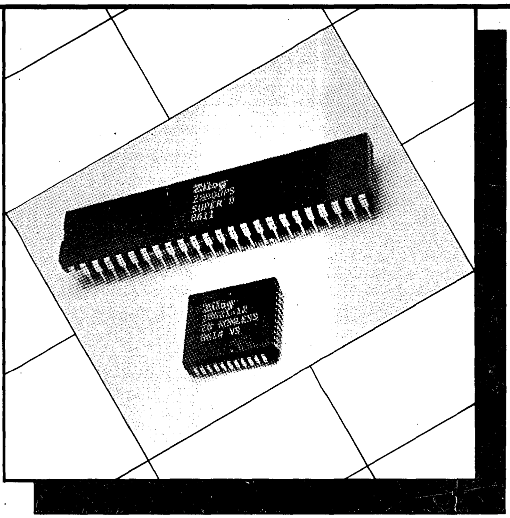
Zilog

June 1988

Z8® Family Design Handbook



Zilog

June 1988

Z8® Family Design Handbook

INTRODUCTION

Zilog was founded in 1974, and within its first year brought to market the most popular and best selling microprocessor in the world, the Z80 8-bit microprocessor.

With the unparalleled success of the Z80 CPU, the name Zilog became synonomous with quality, design integrity, and complete company support elements that remain integral to Zilog today.

Headquartered in Campbell, California, Zilog draws upon the services and skills of the most talented high technology minds in the industry. Zilog's Nampa, Idaho manufacturing facility, and assembly plant in the Philippines are the best of their size today. They provide Zilog customers with a total solution, from engineering, to production, to worldwide on-time delivery of the growing family of Zilog microprocessor and peripheral products.

Z8 Family Design Handbook

Table of Contents

Z8 NMOS MCU Microcomputers

Z8 CMOS MCU Microcomputers

Z8 Application Notes and Technical Articles

Super8 MCU Microcomputer

Super8 Application Notes and Technical Articles

Zilog Product Specification

Z8600 Z8®
Microcomputer

June 1987

FEATURES

□ Complete microcomputer, 2K bytes of ROM, 128 bytes of RAM, and 22 I/O lines.

□ 144-byte register file, including 124 general-purpose registers, four I/O port registers, and 14 status and control registers.

□ Vectored, priority interrupts for I/O and counter/timers.

□ Two programmable 8-bit counter/timers, each with a 6-bit programmable prescaler.

□ Register Pointer so that short, fast instructions can access any one of the nine working register groups.

□ **On–chip oscillator that accepts crystal or external clock drive.**

□ 8 MHz

□ Single +5 power supply—all pins TTL-compatible.

□ **Average instruction execution time of 2.2 µs, minimum 1.5 µs.**

GENERAL DESCRIPTION

The Z8600 microcomputer introduces a new level of sophistication to single-chip architecture. Compared to earlier single-chip microcomputers, the Z8600 offers:

□ faster execution

□ more efficient use of memory

□ more sophisticated interrupt, input/output, and bit manipulation capabilities

□ easier system expansion

Under program control, the MCU can be tailored to the needs of its user. It can be configured as a stand-alone microcomputer with 2K bytes of internal ROM. In all configurations, a large number of pins remain available for I/O.

The MCU is offered in a 28 pin Dual-In-Line-Package (DIP) (Figures 1 and 2).

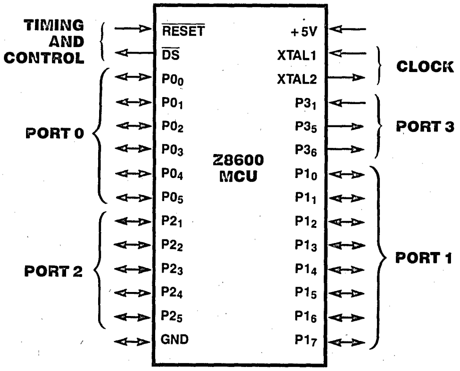


Figure 1. Pin Functions

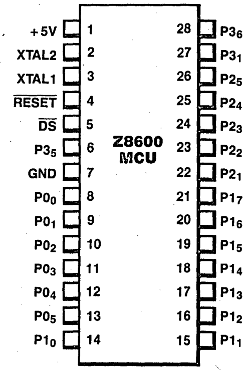


Figure 2. Pin Assignments

PIN DESCRIPTIONS

$\overline{\text{DS}}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each memory transfer.

$P0_0$-$P0_5$, $P1_0$-$P1_7$, $P2_1$-$P2_5$, $P3_1$, $P3_5$, $P3_6$. *I/O Port lines* (bidirectional, TTL-compatible). These 22 I/O lines are grouped in four ports that can be configured under program control for I/O.

$\overline{\text{RESET}}$. *Reset* (input, active Low). $\overline{\text{RESET}}$ initializes the MCU. When $\overline{\text{RESET}}$ is deactivated, program execution begins from internal program location $000C_H$.

XTAL1, XTAL2. *Crystal 1, Crystal 2* (time-base input and output). These pins connect a parallel-resonant 8 MHz crystal to the on-chip clock oscillator and buffer.

ARCHITECTURE

The MCU's architecture is characterized by a flexible I/O scheme, an efficient register and address space structure, and a number of ancillary features that are helpful in many applications. (Figure 3).

Microcomputer applications demand powerful I/O capabilities. The MCU fulfills this with 22 pins dedicated to input and output. These lines are grouped in four ports and are configurable under software control to provide timing, status signals, and parallel I/O.

Two basic internal address spaces are available to support this wide range of configurations: program memory and the register file. The 144-byte random-access register file is composed of 124 general-purpose registers, four I/O port registers, and 14 control and status registers.

To unburden the program from coping with real-time problems such as counting/timing, two counter/timers with a large number of user-selectable modes are offered on-chip.

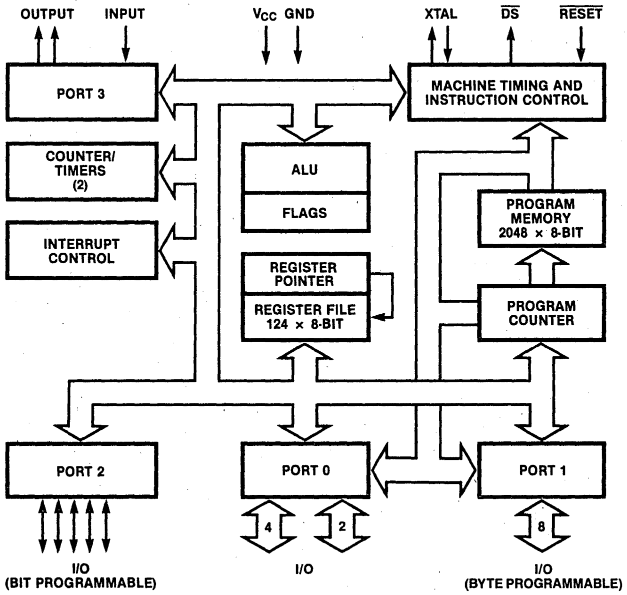


Figure 3. Functional Block Diagram

ADDRESS SPACES

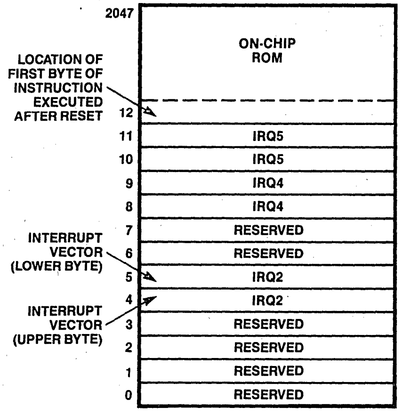
Program Memory. The 16-bit program counter addresses 2K bytes of program memory space as shown in Figure 4.

The first 12 bytes of program memory are reserved for the interrupt vectors. These locations contain three 16-bit vectors that correspond to the three available interrupts.

Register File. The 144-byte register file includes four I/O port registers (R_0-R_3), 124 general-purpose registers (R_4-R_{127}) and **14** control and status registers (R_{241}-R_{255}). These registers are assigned the address locations shown in Figure 5.

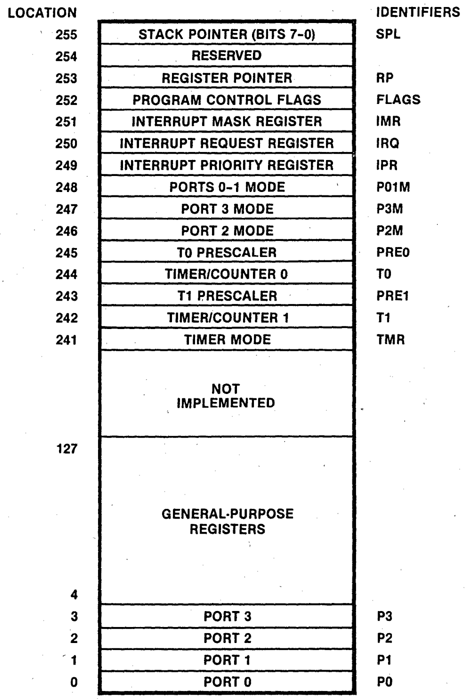
Instructions can access registers directly or indirectly with an 8-bit address field. The MCU also allows short 4-bit register addressing using the Register Pointer (one of the control registers). In the 4-bit mode, the register file is divided into nine working-register groups, each occupying 16 contiguous locations (Figure 6). The Register Pointer addresses the starting location of the active working-register group.

Stacks. An 8-bit Stack Pointer (R_{255}) is used for the internal stack that resides within the 124 general-purpose registers (R_4-R_{127}).


LOCATION OF FIRST BYTE OF INSTRUCTION EXECUTED AFTER RESET
INTERRUPT VECTOR (LOWER BYTE)
INTERRUPT VECTOR (UPPER BYTE)

2047	ON-CHIP ROM
12	
11	IRQ5
10	IRQ5
9	IRQ4
8	IRQ4
7	RESERVED
6	RESERVED
5	IRQ2
4	IRQ2
3	RESERVED
2	RESERVED
1	RESERVED
0	RESERVED

Figure 4. Program Memory Map



LOCATION		IDENTIFIERS
255	STACK POINTER (BITS 7-0)	SPL
254	RESERVED	
253	REGISTER POINTER	RP
252	PROGRAM CONTROL FLAGS	FLAGS
251	INTERRUPT MASK REGISTER	IMR
250	INTERRUPT REQUEST REGISTER	IRQ
249	INTERRUPT PRIORITY REGISTER	IPR
248	PORTS 0-1 MODE	P01M
247	PORT 3 MODE	P3M
246	PORT 2 MODE	P2M
245	T0 PRESCALER	PRE0
244	TIMER/COUNTER 0	T0
243	T1 PRESCALER	PRE1
242	TIMER/COUNTER 1	T1
241	TIMER MODE	TMR
	NOT IMPLEMENTED	
127 4	GENERAL-PURPOSE REGISTERS	
3	PORT 3	P3
2	PORT 2	P2
1	PORT 1	P1
0	PORT 0	P0

Figure 5. Register File

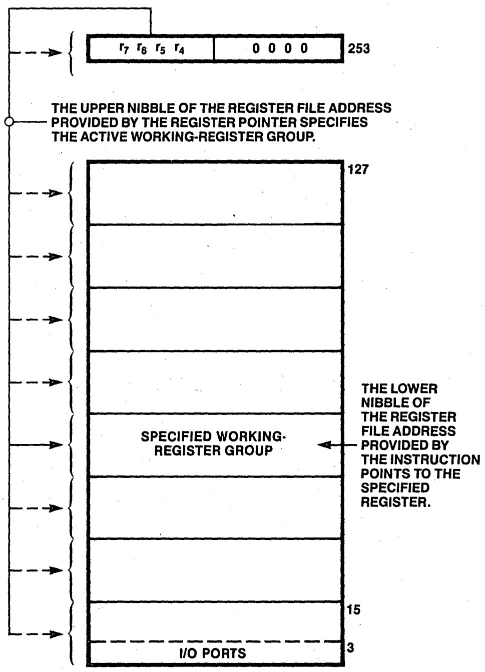

r7 r6 r5 r4
0 0 0 0
253
THE UPPER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE REGISTER POINTER SPECIFIES THE ACTIVE WORKING-REGISTER GROUP.
127
SPECIFIED WORKING-REGISTER GROUP
THE LOWER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE INSTRUCTION POINTS TO THE SPECIFIED REGISTER.
15
I/O PORTS
3

Figure 6. Register Pointer

COUNTER/TIMERS

The MCU contains two 8-bit programmable counter/timers (T_0 and T_1), each driven by its own 6-bit programmable prescaler. The T_1 prescaler can be driven by internal or external clock sources; however, the T_0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrements the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request—IRQ_4 (T_0) or IRQ_5 (T_1)—is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single-pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read any time without disturbing their value or count mode.

The clock source for T_1 is user-definable and can be the internal microprocessor clock (4 MHz maximum) divided by four, or an external signal input via Port 3. The Timer Mode register configures the external timer input as an external clock (1 MHz maximum), a trigger input that can be retriggerable or non-retriggerable, or as a gate input for the internal clock. The counter/timers can be programmably cascaded by connecting the T_0 output to the input of T_1. Port 3 line $P3_6$ also serves as a timer output (T_{OUT}) through which T_0, T_1 or the internal clock can be output.

I/O PORTS

The MCU has 22 lines dedicated to input and output grouped in four ports. Under software control, the ports can be programmed to provide address outputs, timing, status signals, and parallel I/O. All ports have active pull-ups and pull-downs compatible with TTL loads.

Port 0 can be programmed as an I/O port.

Port 1 can be programmed as a byte I/O port.

Port 2 can be programmed independently as input or output and is always available for I/O operations. In addition, Port 2 can be configured to provide open-drain outputs.

Port 3 can be configured as I/O or control lines. $P3_1$ is a general purpose input or can be used for an external interrupt request signal (IRQ_2). $P3_5$ and $P3_6$ are general purpose outputs. $P3_6$ is also used for timer input (T_{IN}) and output (T_{OUT}) signals.

INTERRUPTS

The MCU allows three different interrupts from three sources, the Port 3 line $P3_1$ and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask register globally or individually enables or disables the three interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All interrupts are vectored. When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the Program Counter and status flags, and branches to the program memory vector locations reserved for that interrupt. This memory location and the next byte contain the 16-bit address of the interrupt service routine for that particular interrupt request.

Polled interrupt systems are also supported. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the Interrupt Request register polled to determine which of the interrupt requests needs service.

CLOCK

The on-chip oscillator has a high-gain parallel-resonant amplifier for connection to a crystal or to any suitable external clock source (XTAL1 = Input, XTAL2 = Output).

Crystal source is connected across XTAL1 and XTAL2 using the recommended capacitors (C1 $\leqslant$ 15 pf) from each pin to ground. The specifications are as follows:

- AT cut, parallel resonant
- Fundamental type, 8 MHz maximum
- Series resistance, Rs $\leqslant$ 100Ω

INSTRUCTION SET NOTATION

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "←". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*	Set or cleared according to operation
—	Unaffected
X	Undefined

CONDITION CODES

Value	Mnemonic	Meaning	Flags Set
1000		Always true	—
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	—

INSTRUCTION FORMATS

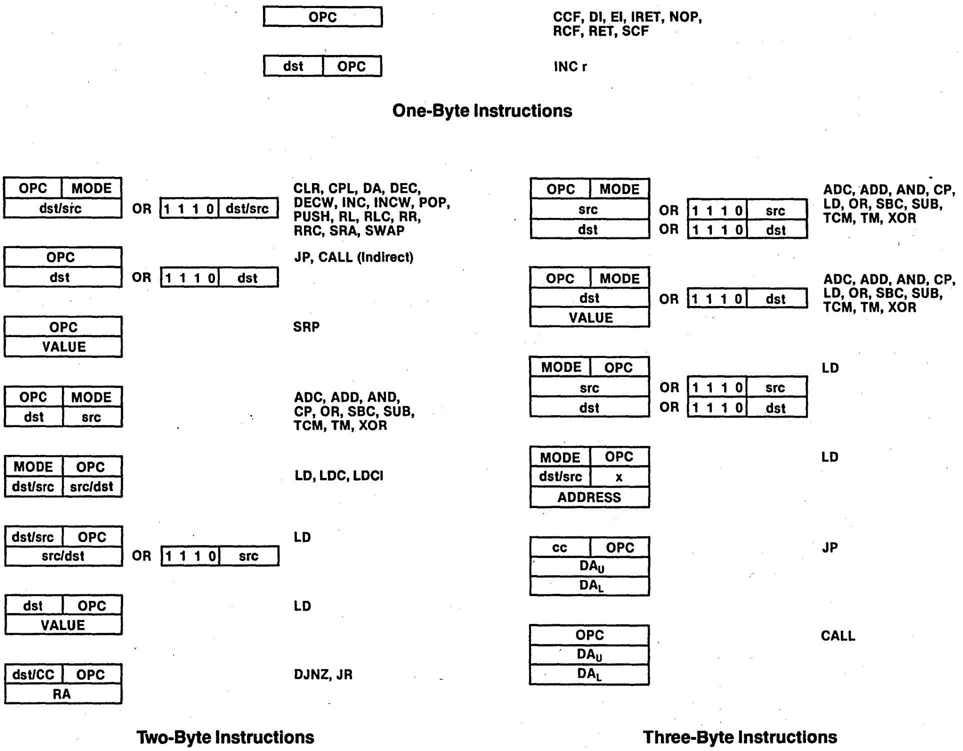


Figure 7. Instruction Formats

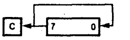
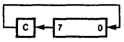
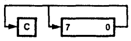
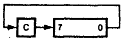
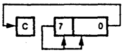
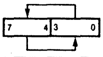
INSTRUCTION SUMMARY

Instruction and Operation	Addr Mode		Opcode Byte	Flags Affected					
	dst	src	(Hex)	C	Z	S	V	D	H
ADC dst,src dst ← dst + src + C	(Note 1)		1□	*	*	*	*	0	*
ADD dst,src dst ← dst + src	(Note 1)		0□	*	*	*	*	0	*
AND dst,src dst ← dst AND src	(Note 1)		5□	—	*	*	0	—	—
CALL dst SP ← SP − 2 @SP ← PC; PC ← dst	DA IRR		D6 D4	—	—	—	—	—	—
CCF C ← NOT C			EF	*	—	—	—	—	—
CLR dst dst ← 0	R IR		B0 B1	—	—	—	—	—	—
COM dst dst ← NOT dst	R IR		60 61	—	*	*	0	—	—

Instruction and Operation	Addr Mode		Opcode Byte	Flags Affected					
	dst	src	(Hex)	C	Z	S	V	D	H
CP dst,src dst − src	(Note 1)		A□	*	*	*	*	—	—
DA dst dst ← DA dst	R IR		40 41	*	*	*	X	—	—
DEC dst dst ← dst − 1	R IR		00 01	—	*	*	*	—	—
DECW dst dst ← dst − 1	RR IR		80 81	—	*	*	*	—	—
DI IMR (7) ← 0			8F	—	—	—	—	—	—
DJNZ r,dst r ← r − 1 if r ≠ 0 PC ← PC + dst Range: +127, −128	RA		rA r = 0 − F	—	—	—	—	—	—

INSTRUCTION SUMMARY (Continued)

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
EI			9F	—	—	—	—	—	—
IMR (7) ← 1									
INC dst	r		rE	—	*	*	*	—	—
dst ← dst + 1			r = 0 − F						
	R		20						
	IR		21						
INCW dst	RR		A0	—	*	*	*	—	—
dst ← dst + 1	IR		A1						
IRET			BF	*	*	*	*	*	*
FLAGS ← @SP; SP ← SP + 1									
PC ← @SP; SP ← SP + 2; IMR (7) ← 1									
JP cc,dst	DA		cD	—	—	—	—	—	—
if cc is true			c = 0 − F						
PC ← dst	IRR		30						
JR cc,dst	RA		cB	—	—	—	—	—	—
if cc is true,			c = 0 − F						
PC ← PC + dst									
Range: +127, −128									
LD dst,src	r	Im	rC	—	—	—	—	—	—
dst ← src	r	R	r8						
	R	r	r9						
			r = 0 − F						
	r	X	C7						
	X	r	D7						
	r	Ir	E3						
	Ir	r	F3						
	R	R	E4						
	R	IR	E5						
	R	IM	E6						
	IR	IM	E7						
	IR	R	F5						
LDC dst,src	r	Irr	C2	—	—	—	—	—	—
dst ← src	Irr	r	D2						
LDCI dst,src	Ir	Irr	C3	—	—	—	—	—	—
dst ← src	Irr	Ir	D3						
r ← r + 1; rr ← rr + 1									
NOP			FF	—	—	—	—	—	—
OR dst,src	(Note 1)		4□	—	*	*	0	—	—
dst ← dst OR src									
POP dst		R	50	—	—	—	—	—	—
dst ← @SP;		IR							
SP ← SP + 1									
PUSH src		R	70	—	—	—	—	—	—
SP ← SP − 1; @SP ← src		IR	71						
RCF			CF	0	—	—	—	—	—
C ← 0									
RET			AF	—	—	—	—	—	—
PC ← @SP; SP ← SP + 2									

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
RL dst 	R		90	*	*	*	*	—	—
	IR		91						
RLC dst 	R		10	*	*	*	*	—	—
	IR		11						
RR dst 	R		E0	*	*	*	*	—	—
	IR		E1						
RRC dst 	R		C0	*	*	*	*	—	—
	IR		C1						
SBC dst,src	(Note 1)		3□	*	*	*	*	1	*
dst ← dst ← src ← C									
SCF			DF	1	—	—	—	—	—
C ← 1									
SRA dst 	R		D0	*	*	*	0	—	—
	IR		D1						
SRP src		Im	31	—	—	—	—	—	—
RP ← src									
SUB dst,src	(Note 1)		2□	*	*	*	*	1	*
dst ← dst ← src									
SWAP dst 	R		F0	X	*	*	X	—	—
	IR		F1						
TCM dst,src	(Note 1)		6□	—	*	*	0	—	—
(NOT dst) AND src									
TM dst,src	(Note 1)		7□	—	*	*	0	—	—
dst AND src									
XOR dst,src	(Note 1)		B□	—	*	*	0	—	—
dst ← dst XOR src									

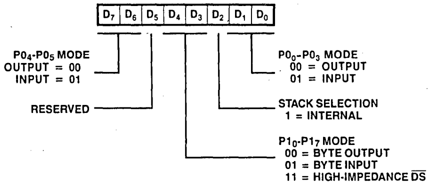
NOTE 1: These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the right of the applicable addressing mode pair.

For example, the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

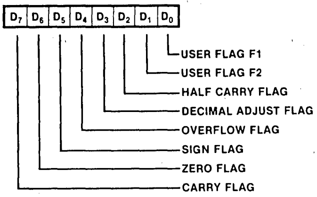
Addr Mode		Lower Opcode Nibble
dst	src	
r	r	2
r	Ir	3
R	R	4
R	IR	5
R	IM	6
IR	IM	7

REGISTERS (Continued)

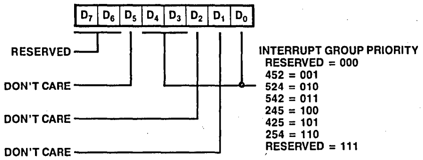
R248 P01M
PORT 0 AND 1 MODE REGISTER
(F8$_H$; Write Only)



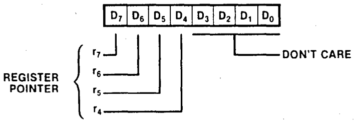
R252 FLAGS
FLAG REGISTER
(FC$_H$; Read/Write)



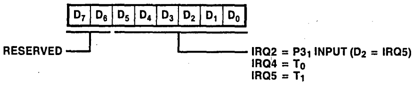
R249 IPR
INTERRUPT PRIORITY REGISTER
(F9$_H$; Write Only)



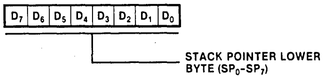
R253 RP
REGISTER POINTER
(FD$_H$; Read/Write)



R250 IRQ
INTERRUPT REQUEST REGISTER
(FA$_H$; Read/Write)



R255 SPL
STACK POINTER
(FF$_H$; Read/Write)



R251 IMR
INTERRUPT MASK REGISTER
(FB$_H$; Read/Write)

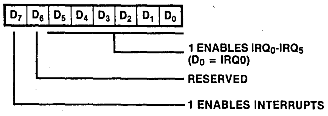


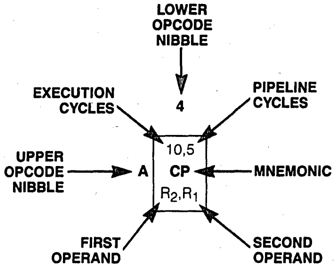
Figure 8. Control Registers (Continued)

OPCODE MAP

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6,5 **DEC** R_1	6,5 **DEC** IR_1	6,5 **ADD** r_1,r_2	6,5 **ADD** r_1,Ir_2	10,5 **ADD** R_2,R_1	10,5 **ADD** IR_2,R_1	10,5 **ADD** R_1,IM	10,5 **ADD** IR_1,IM	6,5 **LD** r_1,R_2	6,5 **LD** r_2,R_1	12/10,5 **DJNZ** r_1,RA	12/10,0 **JR** cc,RA	6,5 **LD** r_1,IM	12/10,0 **JP** cc,DA	6,5 **INC** r1	
1	6,5 **RLC** R_1	6,5 **RLC** IR_1	6,5 **ADC** r_1,r_2	6,5 **ADC** r_1,Ir_2	10,5 **ADC** R_2,R_1	10,5 **ADC** IR_2,R_1	10,5 **ADC** R_1,IM	10,5 **ADC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
2	6,5 **INC** R_1	6,5 **INC** IR_1	6,5 **SUB** r_1,r_2	6,5 **SUB** r_1,Ir_2	10,5 **SUB** R_2,R_1	10,5 **SUB** IR_2,R_1	10,5 **SUB** R_1,IM	10,5 **SUB** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
3	8,0 **JP** IRR_1	6,1 **SRP** IM	6,5 **SBC** r_1,r_2	6,5 **SBC** r_1,Ir_2	10,5 **SBC** R_2,R_1	10,5 **SBC** IR_2,R_1	10,5 **SBC** R_1,IM	10,5 **SBC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
4	8,5 **DA** R_1	8,5 **DA** IR_1	6,5 **OR** r_1,r_2	6,5 **OR** r_1,Ir_2	10,5 **OR** R_2,R_1	10,5 **OR** IR_2,R_1	10,5 **OR** R_1,IM	10,5 **OR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
5	10,5 **POP** R_1	10,5 **POP** IR_1	6,5 **AND** r_1,r_2	6,5 **AND** r_1,Ir_2	10,5 **AND** R_2,R_1	10,5 **AND** IR_2,R_1	10,5 **AND** R_1,IM	10,5 **AND** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
6	6,5 **COM** R_1	6,5 **COM** IR_1	6,5 **TCM** r_1,r_2	6,5 **TCM** r_1,Ir_2	10,5 **TCM** R_2,R_1	10,5 **TCM** IR_2,R_1	10,5 **TCM** R_1,IM	10,5 **TCM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
7	10/12,1 **PUSH** R_2	12/14,1 **PUSH** IR_2	6,5 **TM** r_1,r_2	6,5 **TM** r_1,Ir_2	10,5 **TM** R_2,R_1	10,5 **TM** IR_2,R_1	10,5 **TM** R_1,IM	10,5 **TM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
8	10,5 **DECW** RR_1	10,5 **DECW** IR_1							↓	↓	↓	↓	↓	↓	↓	6.1 **DI**
9	6,5 **RL** R_1	6,5 **RL** IR_1							↓	↓	↓	↓	↓	↓	↓	6.1 **EI**
A	10,5 **INCW** RR_1	10,5 **INCW** IR_1	6,5 **CP** r_1,r_2	6,5 **CP** r_1,Ir_2	10,5 **CP** R_2,R_1	10,5 **CP** IR_2,R_1	10,5 **CP** R_1,IM	10,5 **CP** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	14.0 **RET**
B	6,5 **CLR** R_1	6,5 **CLR** IR_1	6,5 **XOR** r_1,r_2	6,5 **XOR** r_1,Ir_2	10,5 **XOR** R_2,R_1	10,5 **XOR** IR_2,R_1	10,5 **XOR** R_1,IM	10,5 **XOR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	16.0 **IRET**
C	6,5 **RRC** R_1	6,5 **RRC** IR_1	12,0 **LDC** r_1,Irr_2	18,0 **LDCI** Ir_1,Irr_2				10,5 **LD** r_1,x,R_2	↓	↓	↓	↓	↓	↓	↓	6,5 **RCF**
D	6,5 **SRA** R_1	6,5 **SRA** IR_1	12,0 **LDC** r_2,Irr_1	18,0 **LDCI** Ir_2,Irr_1	20,0 **CALL*** IRR_1		20,0 **CALL** DA	10,5 **LD** r_2,x,R_1	↓	↓	↓	↓	↓	↓	↓	6,5 **SCF**
E	6,5 **RR** R_1	6,5 **RR** IR_1		6,5 **LD** r_1,IR_2	10,5 **LD** R_2,R_1	10,5 **LD** IR_2,R_1	10,5 **LD** R_1,IM	10,5 **LD** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	6,5 **CCF**
F	8,5 **SWAP** R_1	8,5 **SWAP** IR_1		6,5 **LD** Ir_1,r_2		10,5 **LD** R_2,IR_1			↓	↓	↓	↓	↓	↓	↓	6,0 **NOP**
	2				3				2					3	1	

Bytes per Instruction



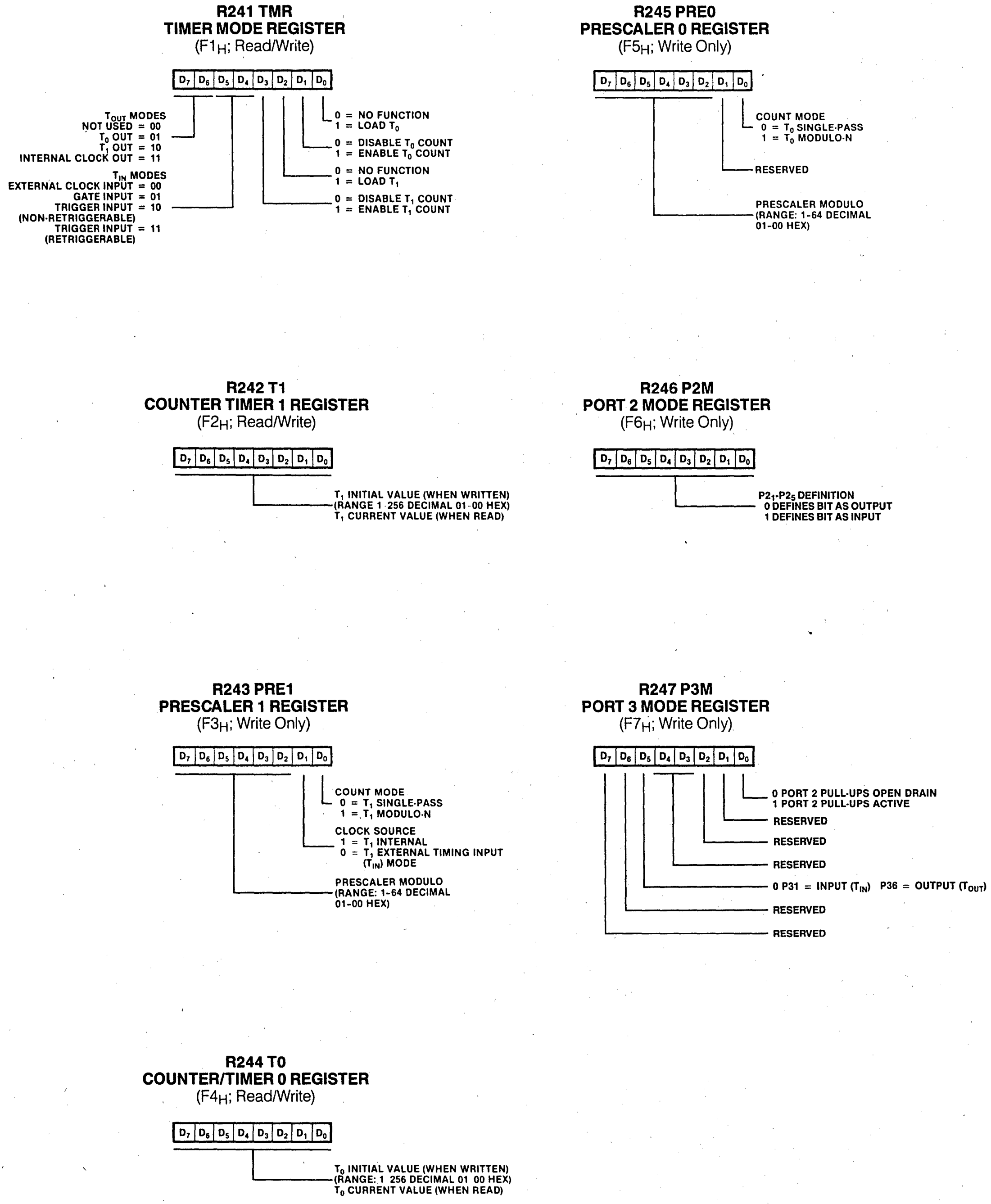
Legend:
R = 8-bit address
r = 4-bit address
R_1 or r_1 = Dst address
R_2 or r_2 = Src address

Sequence:
Opcode, First Operand, Second Operand

NOTE: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

REGISTERS



R241 TMR
TIMER MODE REGISTER
($F1_H$; Read/Write)

R242 T1
COUNTER TIMER 1 REGISTER
($F2_H$; Read/Write)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_1 INITIAL VALUE (WHEN WRITTEN)
(RANGE 1-256 DECIMAL 01-00 HEX)
T_1 CURRENT VALUE (WHEN READ)

R243 PRE1
PRESCALER 1 REGISTER
($F3_H$; Write Only)

R244 T0
COUNTER/TIMER 0 REGISTER
($F4_H$; Read/Write)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_0 INITIAL VALUE (WHEN WRITTEN)
(RANGE: 1-256 DECIMAL 01-00 HEX)
T_0 CURRENT VALUE (WHEN READ)

R245 PRE0
PRESCALER 0 REGISTER
($F5_H$; Write Only)

R246 P2M
PORT 2 MODE REGISTER
($F6_H$; Write Only)

R247 P3M
PORT 3 MODE REGISTER
($F7_H$; Write Only)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
RESERVED
RESERVED
RESERVED
0 P31 = INPUT (T_{IN}) P36 = OUTPUT (T_{OUT})
RESERVED
RESERVED

Figure 8. Control Registers

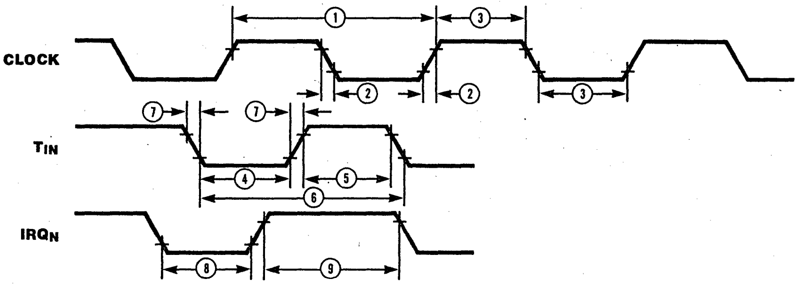


Figure 9. Timing

AC CHARACTERISTICS

Timing Table

			Z8600		
Number	**Symbol**	**Parameter**	**Min**	**Max**	**Notes***
1	TpC	Input Clock Period	125	1000	1
2	TrC,TfC	Clock Input Rise and Fall Times		25	1
3	TwC	Input Clock Width	37		1
4	TwTinL	Timer Input Low Width	100		2
5	TwTinH	Timer Input High Width	3TpC		2
6	TpTin	Timer Input Period	8TpC		2
7	TrTin,TfTin	Timer Input Rise and Fall Times		100	2
8	TwIL	Interrupt Request Input Low Time	100		2,3
9	TwIH	Interrupt Request Input High Time	3TpC		2.3

NOTES:
1. Clock timing references use 3.8V for a logic "1" and 0.8V for a logic "0".
2. Timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
3. Interrupt request via Port 3 ($P3_1$-$P3_3$).
* Units in nanoseconds (ns).

ABSOLUTE MAXIMUM RATINGS

Voltages on all pins with respect
to GND . . . . −0.3V to +7.0V
Operating Ambient
Temperature . . . . .See Ordering Information
Storage Temperature . . . . −65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin.

Standard conditions are:

- $+4.75V \leqslant V_{CC} \leqslant +5.25V$
- GND = 0V
- $0°C \leqslant T_A \leqslant +70°C$

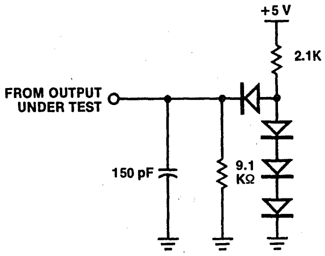


Figure 10. Test Load 1

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8	V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	−0.3	0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{IL}	Input Low Voltage	−0.3	0.8	V	
V_{RH}	Reset Input High Voltage	3.8	V_{CC}	V	
V_{RL}	Reset Input Low Voltage	−0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -250\mu A$
V_{OL}	Output Low Voltage		0.4	V	$I_{OL} = +2.0 mA$
I_{IL}	Input Leakage	−10	10	μA	$0V \leqslant V_{IN} \leqslant +5.25V$
I_{OH}	**Output Drive Current**		**1.5** **2.50**	**mA** **µA**	**$V_{OH} = +2.4V$** **$V_{OH} = +4.0V$**
I_{OL}	Output Leakage	−10	10	μA	$0V \leqslant V_{IN} \leqslant +5.25V$
I_{IR}	Reset Input Current		−50	μA	$V_{CC} = +5.25V, V_{RL} = 0V$
I_{CC}	V_{CC} Supply Current		150	mA	

Zilog

Product Specification

June 1987

Z8601/Z8603
Z8611/Z8613 Z8®

Z8601 Single-Chip MCU with 2K ROM
Z8603 Prototyping Device with 2K EPROM Interface
Z8611 Single-Chip MCU with 4K ROM
Z8613 Prototyping Device with 4K EPROM Interface

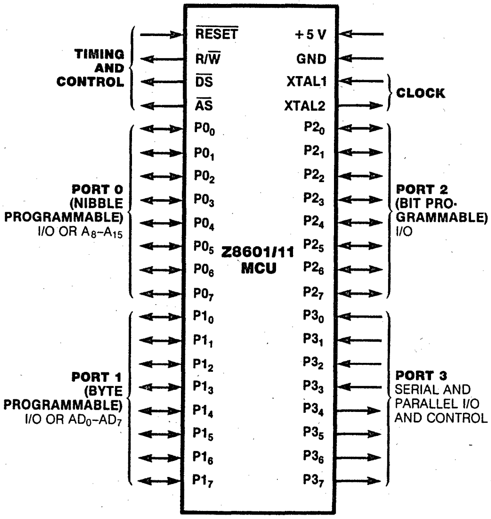
Features

- Complete microcomputer, 2K (8601) or 4K (8611) bytes of ROM, 128 bytes of RAM, 32 I/O lines, and up to 62K (8601) or 60K (8611) bytes addressable external space each for program and data memory.
- 144-byte register file, including 124 general-purpose registers, four I/O port registers, and 16 status and control registers.
- Average instruction execution time of 1.5 μs, maximum of 1 μs.
- Vectored, priority interrupts for I/O, counter/timers, and UART.
- Full-duplex UART and two programmable 8-bit counter/timers, each with a 6-bit programmable prescaler.
- Register Pointer so that short, fast instructions can access any of nine working register groups in 1 μs.
- On-chip oscillator which accepts crystal or external clock drive.
- Single +5 V power supply—all pins TTL compatible.
- 12.5 MHz.

General Description

The Z8 microcomputer introduces a new level of sophistication to single-chip architecture. Compared to earlier single-chip microcomputers, the Z8 offers faster execution; more efficient use of memory; more sophisticated interrupt, input/output and bit-manipulation capabilities; and easier system expansion.

Under program control, the Z8 can be tailored to the needs of its user. It can be configured as a stand-alone microcomputer with 2K or 4K bytes of internal ROM, a traditional microprocessor that manages up to 124K bytes of external memory, or a parallel-processing element in a system with other processors and peripheral controllers linked by the Z-BUS® bus. In all configurations, a large number of pins remain available for I/O.



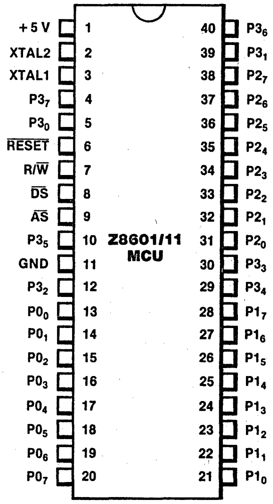


Figure 2a. 40-pin Dual-In-Line Package (DIP), Pin Assignments

Pin Description

$\overline{\text{AS}}$. *Address Strobe* (output, active Low). Address Strobe is pulsed once at the beginning of each machine cycle. Addresses output via Port 1 for all external program or data memory transfers are valid at the trailing edge of $\overline{\text{AS}}$. Under program control, $\overline{\text{AS}}$ can be placed in the high-impedance state along with Ports 0 and 1, Data Strobe and Read/Write.

$\overline{\text{DS}}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each external memory transfer.

P0$_0$-P0$_7$, P1$_0$-P1$_7$, P2$_0$-P2$_7$, P3$_0$-P3$_7$. *I/O Port Lines* (input/outputs, TTL-compatible). These 32 lines are divided into four 8-bit I/O ports that can be configured under program control for I/O or external memory interface.

$\overline{\text{RESET}}$. *Reset* (input, active Low). $\overline{\text{RESET}}$ initializes the Z8. When $\overline{\text{RESET}}$ is deactivated, program execution begins from internal program location 000C$_\text{H}$.

$\overline{\text{ROMless}}$. (input, active LOW). This pin is only available on the 44 pin versions of the Z8601 and Z8611. When connected to GND disables the internal ROM and forces the part to function as a Z8681 ROMless Z8. When left unconnected or pulled high to V$_\text{CC}$ the part will function normally as a Z8601 or Z8611.

R/$\overline{\text{W}}$. *Read/Write* (output). R/$\overline{\text{W}}$ is Low when the Z8 is writing to external program or data memory.

XTAL1, XTAL2. *Crystal 1, Crystal 2* (time-base input and output). These pins connect a parallel resonant 12.5 MHz crystal or an external single-phase 12.5 MHz clock to the on-chip clock oscillator and buffer.

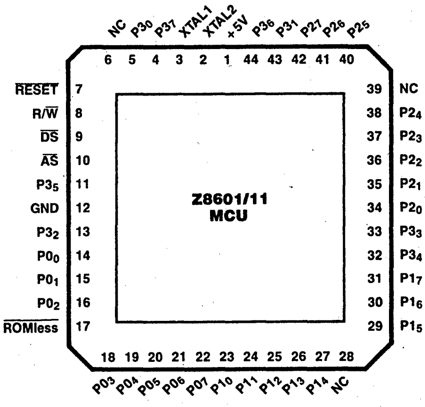


Figure 2b. 44-pin Chip Carrier, Pin Assignments

Architecture

Z8 architecture is characterized by a flexible I/O scheme, an efficient register and address space structure and a number of ancillary features that are helpful in many applications.

Microcomputer applications demand powerful I/O capabilities. The Z8 fulfills this with 32 pins dedicated to input and output. These lines are grouped into four ports of eight lines each and are configurable under software control to provide timing, status signals, serial or parallel I/O with or without handshake, and an address/data bus for interfacing external memory.

Because the multiplexed address/data bus is merged with the I/O-oriented ports, the Z8 can assume many different memory and I/O configurations. These configurations range from a self-contained microcomputer to a microprocessor that can address 124K (Z8601) or 120K (Z8611) bytes of external memory.

Three basic address spaces are available to support this wide range of configurations: program memory (internal and external), data memory (external) and the register file (internal). The 144-byte random-access register file is composed of 124 general-purpose registers, four I/O port registers, and 16 control and status registers.

To unburden the program from coping with real-time problems such as serial data communication and counting/timing, an asynchronous receiver/transmitter (UART) and two counter/timers with a large number of userselectable modes are offered on-chip. Hardware support for the UART is minimized because one of the on-chip timers supplies the bit rate.

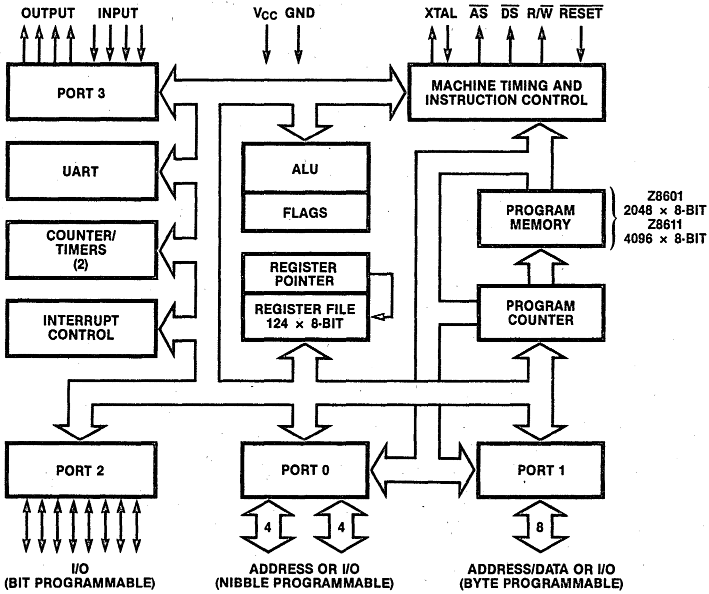


Figure 3. Functional Block Diagram

Address Spaces

Program Memory. The 16-bit program counter addresses 64K bytes of program memory space. Program memory can be located in two areas: one internal and the other external (Figure 4). The first 2048 (Z8601) or 4096 (Z8611) bytes consist of on-chip mask-programmed ROM. At addresses 2048 (Z8601) or 4096 (Z8611) and greater, the Z8 executes external program memory fetches.

The first 12 bytes of program memory are reserved for the interrupt vectors. These locations contain six 16-bit vectors that correspond to the six available interrupts.

Data Memory. The Z8 can address 62K (Z8601) or 60K (Z8611) bytes of external data memory beginning at location 2048 (Z8601) or 4096 (Z8611) (Figure 5). External data memory may be included with or separated from the external program memory space. $\overline{DM}$, an optional I/O function that can be programmed to appear on pin $P3_4$, is used to distinguish between data and program memory space.

Register File. The 144-byte register file includes four I/O port registers (R0–R3), 124 general-purpose registers (R4–R127) and 16 control and status registers (R240–R255). These registers are assigned the address locations shown in Figure 6.

Z8 instructions can access registers directly or indirectly with an 8-bit address field. The Z8 also allows short 4-bit register addressing using the Register Pointer (one of the control registers). In the 4-bit mode, the register file is

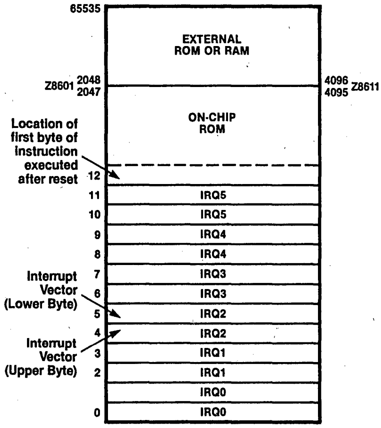


Figure 4. Program Memory Map

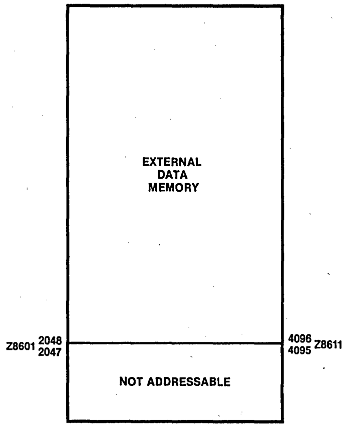


Figure 5. Data Memory Map

LOCATION		IDENTIFIERS
255	STACK POINTER (BITS 7-0)	SPL
254	STACK POINTER (BITS 15-8)	SPH
253	REGISTER POINTER	RP
252	PROGRAM CONTROL FLAGS	FLAGS
251	INTERRUPT MASK REGISTER	IMR
250	INTERRUPT REQUEST REGISTER	IRQ
249	INTERRUPT PRIORITY REGISTER	IPR
248	PORTS 0-1 MODE	P01M
247	PORT 3 MODE	P3M
246	PORT 2 MODE	P2M
245	T0 PRESCALER	PRE0
244	TIMER/COUNTER 0	T0
243	T1 PRESCALER	PRE1
242	TIMER/COUNTER 1	T1
241	TIMER MODE	TMR
240	SERIAL I/O	SIO
	NOT IMPLEMENTED	
127	GENERAL-PURPOSE REGISTERS	
4		
3	PORT 3	P3
2	PORT 2	P2
1	PORT 1	P1
0	PORT 0	P0

Figure 6. The Register File

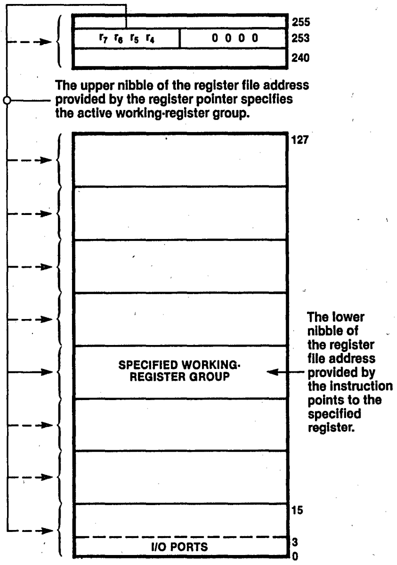

255
r_7 r_6 r_5 r_4
0 0 0 0
253
240
The upper nibble of the register file address provided by the register pointer specifies the active working-register group.
127
SPECIFIED WORKING-REGISTER GROUP
The lower nibble of the register file address provided by the instruction points to the specified register.
15
3
I/O PORTS
0

Figure 7. The Register Pointer

divided into nine working-register groups, each occupying 16 continguous locations (Figure 6). The Register Pointer addresses the starting location of the active working-register group.

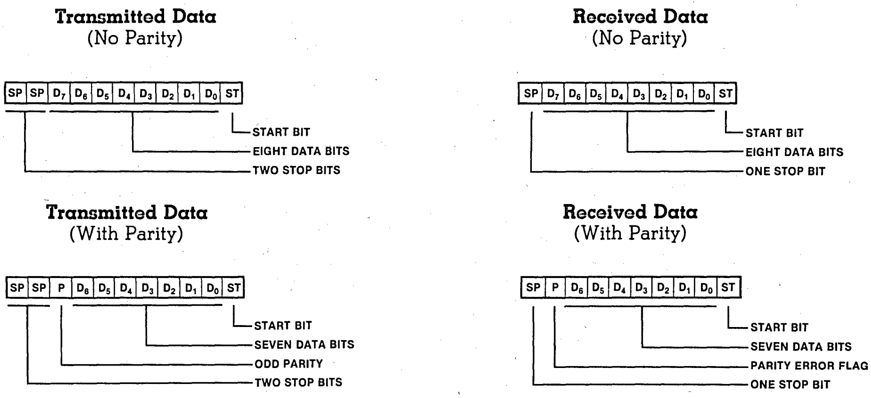
Stacks. Either the internal register file or the external data memory can be used for the stack. A 16-bit Stack Pointer (R254 and R255) is used for the external stack, which can reside anywhere in data memory between locations 2048 (8601) or 4096 (8611) and 65535. An 8-bit Stack Pointer (R255) is used for the internal stack that resides within the 124 general-purpose registers (R4–R127).

Serial Input/ Output

Port 3 lines $P3_0$ and $P3_7$ can be programmed as serial I/O lines for full-duplex serial asynchronous receiver/transmitter operation. The bit rate is controlled by Counter/Timer 0, at 12 MHz.

The Z8 automatically adds a start bit and two stop bits to transmitted data (Figure 8). Odd parity is also available as an option. Eight data bits are always transmitted, regardless of parity selection. If parity is enabled, the eighth bit is the odd parity bit. An interrupt request (IRQ_4) is generated on all transmitted characters.

Received data must have a start bit, eight data bits and at least one stop bit. If parity is on, bit 7 of the received data is replaced by a parity error flag. Received characters generate the IRQ_3 interrupt request.


Transmitted Data
(No Parity)

Received Data
(No Parity)

Transmitted Data
(With Parity)

Received Data
(With Parity)

Figure 8. Serial Data Formats

Counter/ Timers

The Z8 contains two 8-bit programmable counter/timers (T_0 and T_1), each driven by its own 6-bit programmable prescaler. The T_1 prescaler can be driven by internal or external clock sources; however, the T_0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrements the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request—IRQ_4 (t_0) or IRQ_5 (T_1)—is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single-pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read any time without disturbing their value or count mode.

The clock source for T_1 is user-definable and can be the internal microprocessor clock divided by four, or an external signal input via Port 3. The Timer Mode register configures the external timer input as an external clock, a trigger input that can be retriggerable or non-retriggerable, or as a gate input for the internal clock. The counter/timers can be programmably cascaded by connecting the T_0 output to the input of T_1. Port 3 line $P3_6$ also serves as a timer output (T_{OUT}) through which T_0, T_1 or the internal clock can be output.

I/O Ports

The Z8 has 32 lines dedicated to input and output. These lines are grouped into four ports of eight lines each and are configurable as input, output or address/data. Under software control, the ports can be programmed to provide address outputs, timing, status signals, serial I/O, and parallel I/O with or without handshake. All ports have active pull-ups and pull-downs compatible with TTL loads.

Port 1 can be programmed as a byte I/O port or as an address/data port for interfacing external memory. When used as an I/O port, Port 1 may be placed under handshake control. In this configuration, Port 3 lines $P3_3$ and $P3_4$ are used as the handshake controls RDY_1 and $\overline{DAV}_1$ (Ready and Data Available).

Memory locations greater than 2048 (Z8601) or 4096 (Z8611) are referenced through Port 1. To interface external memory, Port 1 must be programmed for the multiplexed Address/Data mode. If more than 256 external locations are required, Port 0 must output the additional lines.

Port 1 can be placed in the high-impedance state along with Port 0, $\overline{AS}$, $\overline{DS}$ and $R/\overline{W}$, allowing the Z8 to share common resources in multiprocessor and DMA applications. Data transfers can be controlled by assigning $P3_3$ as a Bus Acknowledge input and $P3_4$ as a Bus Request output.

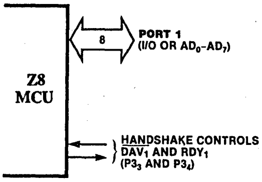


Figure 9a. Port 1

Port 0 can be programmed as a nibble I/O port, or as an address port for interfacing external memory. When used as an I/O port, Port 0 may be placed under handshake control. In this configuration, Port 3 lines $P3_2$ and $P3_5$ are used as the handshake controls $\overline{DAV}_0$ and RDY_0. Handshake signal assignment is dictated by the I/O direction of the upper nibble $P0_4$–$P0_7$.

For external memory references, Port 0 can provide address bits A_8–A_{11} (lower nibble) or A_8–A_{15} (lower and upper nibble) depending on the required address space. If the address range requires 12 bits or less, the upper nibble of Port 0 can be programmed independently as I/O while the lower nibble is used for addressing. When Port 0 nibbles are defined as address bits, they can be set to the highimpedance state along with Port 1 and the control signals $\overline{AS}$, $\overline{DS}$ and $R/\overline{W}$.

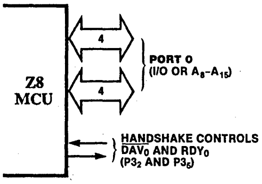


Figure 9b. Port 0

Port 2 bits can be programmed independently as input or output. The port is always available for I/O operations. In addition, Port 2 can be configured to provide open-drain outputs.

Like Ports 0 and 1, Port 2 may also be placed under handshake control. In this configuration, Port 3 lines $P3_1$ and $P3_6$ are used as the handshake controls lines $\overline{DAV}_2$ and RDY_2. The handshake signal assignment for Port 3 lines $P3_1$ and $P3_6$ is dictated by the direction (input or output) assigned to bit 7 of Port 2.

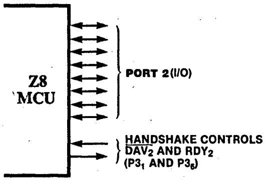


Figure 9c. Port 2

Port 3 lines can be configured as I/O or control lines. In either case, the direction of the eight lines is fixed as four input ($P3_0$–$P3_3$) and four output ($P3_4$–$P3_7$). For serial I/O, lines $P3_0$ and $P3_7$ are programmed as serial in and serial out respectively.

Port 3 can also provide the following control functions: handshake for Ports 0, 1 and 2 ($\overline{DAV}$ and RDY); four external interrupt request signals (IRQ_0–IRQ_3); timer input and output signals (T_{IN} and T_{OUT}) and Data Memory Select ($\overline{DM}$).

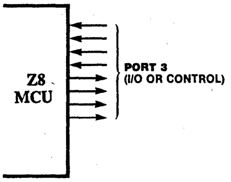


Figure 9d. Port 3

Interrupts

The Z8 allows six different interrupts from eight sources: the four Port 3 lines $P3_0$–$P3_3$, Serial In, Serial Out, and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask register globally or individually enables or disables the six interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All Z8 interrupts are vectored. When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the Program Counter and status flags, and branches to the program memory vector location reserved for that interrupt. This memory location and the next byte contain the 16-bit address of the interrupt service routine for that particular interrupt request.

Polled interrupt systems are also supported. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the Interrupt Request register polled to determine which of the interrupt requests needs service.

Clock

The on-chip oscillator has a high-gain, parallel-resonant amplifier for connection to a crystal or to any suitable external clock source (XTAL1 = Input, XTAL2 = Output).

The crystal source is connected across XTAL1 and XTAL2, using the recommended capacitors ($C_1 \leq 15$ pF) from each pin to ground. The specifications for the crystal are as follows:

- AT cut, parallel resonant
- Fundamental type, 12.5 MHz maximum
- Series resistance, $R_s \leq 100\,\Omega$

Z8603/13 Protopack Emulator

The Z8 Protopack is used for prototype development and preproduction of mask-programmed applications. The Protopack is a ROMless version of the standard Z8601 or Z8611 housed in a pin-compatible 40-pin package (Figure 11).

To provide pin compatibility and interchangeability with the standard maskprogrammed device, the Protopack carries piggy-back a 24-pin socket for a direct interface to program memory (Figure 1). The Z8603 24-pin socket is equipped with 11 ROM address lines, 8 ROM data lines and necessary control lines for interface to 2716 EPROM for the first 2K bytes of program memory. The Z8613 24-pin socket is equipped with 12 ROM address lines, 8 ROM data lines and necessary control lines for interface to 2732 EPROM for the first 4K bytes of program memory.

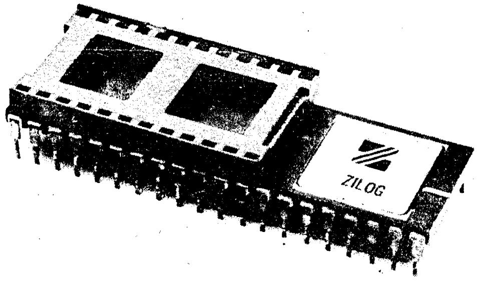


Figure 11. The Z8 Microcomputer Protopack Emulator

Pin compatibility allows the user to design the pc board for a final 40-pin maskprogrammed Z8, and, at the same time, allows the use of the Protopack to build the prototype and pilot production units. When the final program is established, the user can then switch over to the 40-pin mask-programmed Z8 for large volume production. The Protopack is also useful in small volume applica tions where masked ROM setup time, mask charges, etc., are prohibitive and program flexibility is desired.

Compared to the conventional EPROM versions of the single-chip microcomputers, the Protopack approach offers two main advantages:

- Ease of developing various programs during the prototyping stage. For instance, in applications where the same hardware configuration is used with more than one program, the Protopack allows economical program storage in separate EPROMs (or PROMs), whereas the use of separate EPROM-based single-chip microcomputers is more costly.
- Elimination of long lead time in procuring EPROM-based microcomputers.

Instruction Set Notation

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "←". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

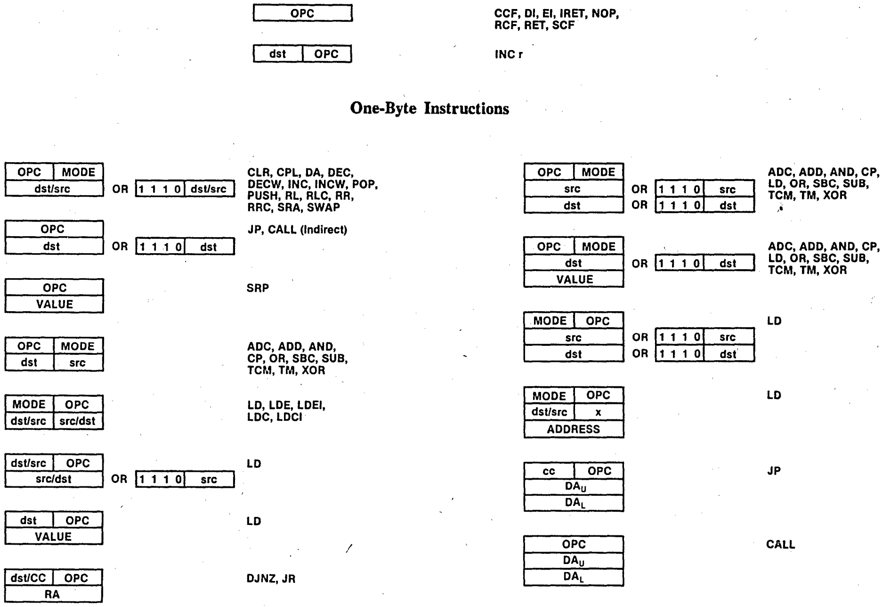
Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*****	Set or cleared according to operation
—	Unaffected
X	Undefined

Condition Codes

Value	Mnemonic	Meaning	Flags Set
1000		Always true	---
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	---

Instruction Formats



One-Byte Instructions

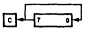
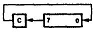
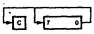
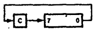
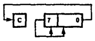
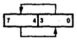
Two-Byte Instructions

Three-Byte Instructions

Figure 12. Instruction Formats

Instruction Summary

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected
	dst	src		C Z S V D H
ADC dst,src	(Note 1)		1□	* * * * 0 *
dst ← dst + src + C				
ADD dst,src	(Note 1)		0□	* * * * 0 *
dst ← dst + src				
AND dst,src	(Note 1)		5□	– * * 0 – –
dst ← dst AND src				
CALL dst	DA		D6	– – – – – –
SP ← SP - 2	IRR		D4	
@SP ← PC; PC ← dst				
CCF			EF	* – – – – –
C ← NOT C				
CLR dst	R		B0	– – – – – –
dst ← 0	IR		B1	
COM dst	R		60	– * * 0 – –
dst ← NOT dst	IR		61	
CP dst,src	(Note 1)		A□	* * * * – –
dst - src				
DA dst	R		40	* * * X – –
dst ← DA dst	IR		41	
DEC dst	R		00	– * * * – –
dst ← dst - 1	IR		01	
DECW dst	RR		80	– * * * – –
dst ← dst - 1	IR		81	
DI				
IMR (7) ← 0			8F	– – – – – –
DJNZ r,dst	RA		rA	– – – – – –
r ← r - 1			r=0-F	
if r ≠ 0				
PC ← PC + dst				
Range: +127, -128				
EI			9F	– – – – – –
IMR (7) ← 1				
INC dst	r		rE	– * * * – –
dst ← dst + 1			r=0-F	
	R		20	
	IR		21	
INCW dst	RR		A0	– * * * – –
dst ← dst + 1	IR		A1	
IRET			BF	* * * * * *
FLAGS ← @SP; SP ← SP + 1				
PC ← @SP; SP ← SP + 2; IMR(7) ← 1				
JP cc,dst	DA		cD	– – – – – –
if cc is true			c=0-F	
PC ← dst	IRR		30	
JR cc,dst	RA		cB	– – – – – –
if cc is true,			c=0-F	
PC ← PC + dst				
Range: +127, -128				
LD dst,src	r	Im	rC	– – – – – –
dst ← src	r	R	r8	
	R	r	r9	
			r=0-F	
	r	X	C7	
	X	r	D7	
	r	Ir	E3	
	Ir	r	F3	
	R	R	E4	
	R	IR	E5	
	R	Im	E6	
	IR	Im	E7	
	IR	R	F5	
LDC dst,src	r	Irr	C2	– – – – – –
dst ← src	Irr	r	D2	
LDCI dst,src	Ir	Irr	C3	– – – – – –
dst ← src	Irr	Ir	D3	
r ← r + 1; rr ← rr + 1				

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected
	dst	src		C Z S V D H
LDE dst,src	r	Irr	82	– – – – – –
dst ← src	Irr	r	92	
LDEI dst,src	Ir	Irr	83	– – – – – –
dst ← src	Irr	Ir	93	
r ← r + 1; rr ← rr + 1				
NOP			FF	– – – – – –
OR dst,src	(Note 1)		4□	– * * 0 – –
dst ← dst OR src				
POP dst	R		50	– – – – – –
dst ← @SP	IR		51	
SP ← SP + 1				
PUSH src		R	70	– – – – – –
SP ← SP - 1; @SP ← src		IR	71	
RCF			CF	0 – – – – –
C ← 0				
RET			AF	– – – – – –
PC ← @SP; SP ← SP + 2				
RL dst 	R		90	* * * * – –
	IR		91	
RLC dst 	R		10	* * * * – –
	IR		11	
RR dst 	R		E0	* * * * – –
	IR		E1	
RRC dst 	R		C0	* * * * – –
	IR		C1	
SBC dst,src	(Note 1)		3□	* * * * 1 *
dst ← dst - src - C				
SCF			DF	1 – – – – –
C ← 1				
SRA dst 	R		D0	* * * 0 – –
	IR		D1	
SRP src		Im	31	– – – – – –
RP ← src				
SUB dst,src	(Note 1)		2□	* * * * 1 *
dst ← dst - src				
SWAP dst 	R		F0	X * * X – –
	IR		F1	
TCM dst,src	(Note 1)		6□	– * * 0 – –
(NOT dst) AND src				
TM dst, src	(Note 1)		7□	– * * 0 – –
dst AND src				
XOR dst,src	(Note 1)		B□	– * * 0 – –
dst ← dst XOR src				

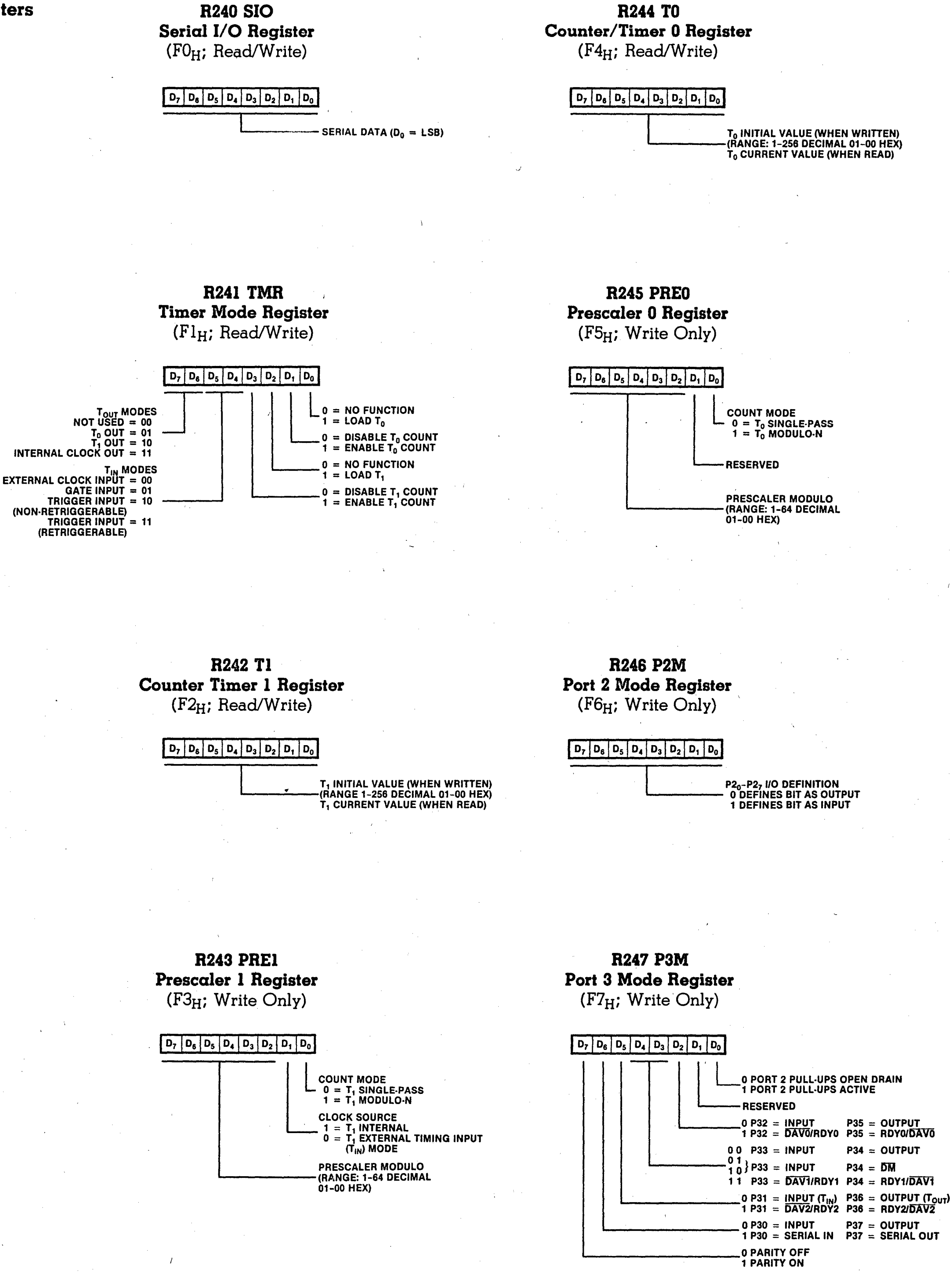
Note 1

These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the right of the applicable addressing mode pair.

For example, to determine the opcode of a ADC instruction use the addressing modes r (destination) and Ir (source). The result is 13.

Addr Mode		Lower Opcode Nibble
dst	src	
r	r	$\boxed{2}$
r	Ir	$\boxed{3}$
R	R	$\boxed{4}$
R	IR	$\boxed{5}$
R	IM	$\boxed{6}$
IR	IM	$\boxed{7}$

Registers



R240 SIO
Serial I/O Register
($F0_H$; Read/Write)

R244 T0
Counter/Timer 0 Register
($F4_H$; Read/Write)

D7 D6 D5 D4 D3 D2 D1 D0
T_0 INITIAL VALUE (WHEN WRITTEN)
(RANGE: 1-256 DECIMAL 01-00 HEX)
T_0 CURRENT VALUE (WHEN READ)

R241 TMR
Timer Mode Register
($F1_H$; Read/Write)

R245 PRE0
Prescaler 0 Register
($F5_H$; Write Only)

R242 T1
Counter Timer 1 Register
($F2_H$; Read/Write)

D7 D6 D5 D4 D3 D2 D1 D0
T_1 INITIAL VALUE (WHEN WRITTEN)
(RANGE 1-256 DECIMAL 01-00 HEX)
T_1 CURRENT VALUE (WHEN READ)

R246 P2M
Port 2 Mode Register
($F6_H$; Write Only)

R243 PRE1
Prescaler 1 Register
($F3_H$; Write Only)

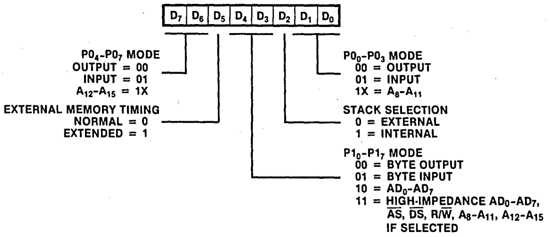
R247 P3M
Port 3 Mode Register
($F7_H$; Write Only)

D7 D6 D5 D4 D3 D2 D1 D0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
RESERVED
0 P32 = INPUT P35 = OUTPUT
1 P32 = $\overline{DAV0}$/RDY0 P35 = RDY0/$\overline{DAV0}$
0 0 P33 = INPUT P34 = OUTPUT
0 1 } P33 = INPUT P34 = $\overline{DM}$
1 0
1 1 P33 = $\overline{DAV1}$/RDY1 P34 = RDY1/$\overline{DAV1}$
0 P31 = INPUT (T_{IN}) P36 = OUTPUT (T_{OUT})
1 P31 = $\overline{DAV2}$/RDY2 P36 = RDY2/$\overline{DAV2}$
0 P30 = INPUT P37 = OUTPUT
1 P30 = SERIAL IN P37 = SERIAL OUT
0 PARITY OFF
1 PARITY ON

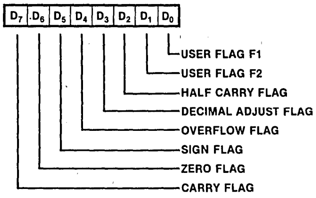
Figure 13. Control Registers

Registers (Continued)

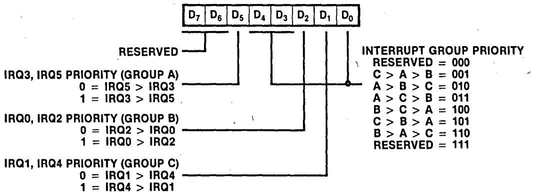
R248 P01M
Port 0 and 1 Mode Register
($F8_H$; Write Only)



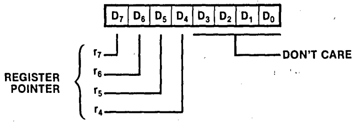
R252 FLAGS
Flag Register
(FC_H; Read/Write)



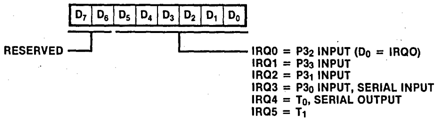
R249 IPR
Interrupt Priority Register
($F9_H$; Write Only)



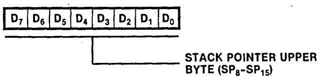
R253 RP
Register Pointer
(FD_H; Read/Write)



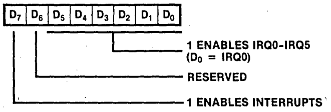
R250 IRQ
Interrupt Request Register
(FA_H; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
RESERVED
IRQ0 = $P3_2$ INPUT (D_0 = IRQ0)
IRQ1 = $P3_3$ INPUT
IRQ2 = $P3_1$ INPUT
IRQ3 = $P3_0$ INPUT, SERIAL INPUT
IRQ4 = T_0, SERIAL OUTPUT
IRQ5 = T_1

R254 SPH
Stack Pointer
(FE_H; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
STACK POINTER UPPER BYTE (SP_8-SP_{15})

R251 IMR
Interrupt Mask Register
(FB_H; Read/Write)



R255 SPL
Stack Pointer
(FF_H; Read/Write)

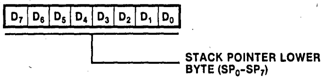
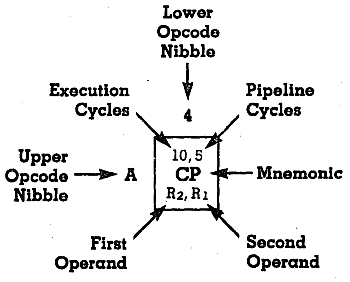

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
STACK POINTER LOWER BYTE (SP_0-SP_7)

Figure 13. Control Registers (Continued)

Opcode Map

	Lower Nibble (Hex)															
Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6,5 **DEC** R_1	6,5 **DEC** IR_1	6,5 **ADD** r_1, r_2	6,5 **ADD** r_1, Ir_2	10,5 **ADD** R_2, R_1	10,5 **ADD** IR_2, R_1	10,5 **ADD** R_1, IM	10,5 **ADD** IR_1, IM	6,5 **LD** r_1, R_2	6,5 **LD** r_2, R_1	12/10,5 **DJNZ** r_1, RA	12/10,0 **JR** cc, RA	6,5 **LD** r_1, IM	12/10,0 **JP** cc, DA	6,5 **INC** r_1	
1	6,5 **RLC** R_1	6,5 **RLC** IR_1	6,5 **ADC** r_1, r_2	6,5 **ADC** r_1, Ir_2	10,5 **ADC** R_2, R_1	10,5 **ADC** IR_2, R_1	10,5 **ADC** R_1, IM	10,5 **ADC** IR_1, IM	↓	↓	↓	↓	↓	↓	↓	
2	6,5 **INC** R_1	6,5 **INC** IR_1	6,5 **SUB** r_1, r_2	6,5 **SUB** r_1, Ir_2	10,5 **SUB** R_2, R_1	10,5 **SUB** IR_2, R_1	10,5 **SUB** R_1, IM	10,5 **SUB** IR_1, IM	↓	↓	↓	↓	↓	↓	↓	
3	8,0 **JP** IRR_1	6,1 **SRP** IM	6,5 **SBC** r_1, r_2	6,5 **SBC** r_1, Ir_2	10,5 **SBC** R_2, R_1	10,5 **SBC** IR_2, R_1	10,5 **SBC** R_1, IM	10,5 **SBC** IR_1, IM	↓	↓	↓	↓	↓	↓	↓	
4	8,5 **DA** R_1	8,5 **DA** IR_1	6,5 **OR** r_1, r_2	6,5 **OR** r_1, Ir_2	10,5 **OR** R_2, R_1	10,5 **OR** IR_2, R_1	10,5 **OR** R_1, IM	10,5 **OR** IR_1, IM	↓	↓	↓	↓	↓	↓	↓	
5	10,5 **POP** R_1	10,5 **POP** IR_1	6,5 **AND** r_1, r_2	6,5 **AND** r_1, Ir_2	10,5 **AND** R_2, R_1	10,5 **AND** IR_2, R_1	10,5 **AND** R_1, IM	10,5 **AND** IR_1, IM	↓	↓	↓	↓	↓	↓	↓	
6	6,5 **COM** R_1	6,5 **COM** IR_1	6,5 **TCM** r_1, r_2	6,5 **TCM** r_1, Ir_2	10,5 **TCM** R_2, R_1	10,5 **TCM** IR_2, R_1	10,5 **TCM** R_1, IM	10,5 **TCM** IR_1, IM	↓	↓	↓	↓	↓	↓	↓	
7	10/12,1 **PUSH** R_2	12/14,1 **PUSH** IR_2	6,5 **TM** r_1, r_2	6,5 **TM** r_1, Ir_2	10,5 **TM** R_2, R_1	10,5 **TM** IR_2, R_1	10,5 **TM** R_1, IM	10,5 **TM** IR_1, IM	↓	↓	↓	↓	↓	↓	↓	
8	10,5 **DECW** RR_1	10,5 **DECW** IR_1	12,0 **LDE** r_1, Irr_2	18,0 **LDEI** Ir_1, Irr_2					↓	↓	↓	↓	↓	↓	↓	6,1 **DI**
9	6,5 **RL** R_1	6,5 **RL** IR_1	12,0 **LDE** Irr_1	18,0 **LDEI** Ir_2, Irr_1					↓	↓	↓	↓	↓	↓	↓	6,1 **EI**
A	10,5 **INCW** RR_1	10,5 **INCW** IR_1	6,5 **CP** r_1, r_2	6,5 **CP** r_1, Ir_2	10,5 **CP** R_2, R_1	10,5 **CP** IR_2, R_1	10,5 **CP** R_1, IM	10,5 **CP** IR_1, IM	↓	↓	↓	↓	↓	↓	↓	14,0 **RET**
B	6,5 **CLR** R_1	6,5 **CLR** IR_1	6,5 **XOR** r_1, r_2	6,5 **XOR** r_1, Ir_2	10,5 **XOR** R_2, R_1	10,5 **XOR** IR_2, R_1	10,5 **XOR** R_1, IM	10,5 **XOR** IR_1, IM	↓	↓	↓	↓	↓	↓	↓	16,0 **IRET**
C	6,5 **RRC** R_1	6,5 **RRC** IR_1	12,0 **LDC** r_1, Irr_2	18,0 **LDCI** Ir_1, Irr_2				10,5 **LD** r_1, x, R_2	↓	↓	↓	↓	↓	↓	↓	6,5 **RCF**
D	6,5 **SRA** R_1	6,5 **SRA** IR_1	12,0 **LDC** r_2, Irr_1	18,0 **LDCI** Ir_2, Irr_1	20,0 **CALL*** IRR_1		20,0 **CALL** DA	10,5 **LD** r_2, x, R_1	↓	↓	↓	↓	↓	↓	↓	6,5 **SCF**
E	6,5 **RR** R_1	6,5 **RR** IR_1		6,5 **LD** r_1, Ir_2	10,5 **LD** R_2, R_1	10,5 **LD** IR_2, R_1	10,5 **LD** R_1, IM	10,5 **LD** IR_1, IM	↓	↓	↓	↓	↓	↓	↓	6,5 **CCF**
F	8,5 **SWAP** R_1	8,5 **SWAP** IR_1		6,5 **LD** Ir_1, r_2		10,5 **LD** R_2, IR_1			↓	↓	↓	↓	↓	↓	↓	6,0 **NOP**
Bytes per Instruction	2				3				2					3	1	



Legend:
R = 8-Bit Address
r = 4-Bit Address
R_1 or r_1 = Dst Address
R_2 or r_2 = Src Address

Sequence:
Opcode, First Operand, Second Operand

Note: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

Absolute Maximum Ratings

Voltages on all pins
with respect to GND . . . . -0.3 V to +7.0 V
Operating Ambient
Temperature . . . . See Ordering Information
Storage Temperature . . . . -65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Standard Test Conditions

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the reference pin.

Standard conditions are:

□ $+4.75\ V \le V_{CC} \le +5.25\ V$

□ GND = 0 V

□ $0°C \le T_A \le +70°C$

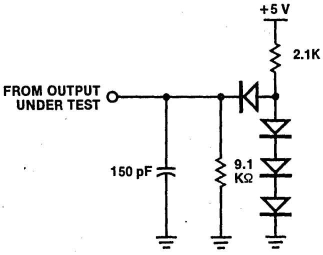


Figure 14. Test Load 1

DC Characteristics

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8	V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	-0.3	0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{RH}	Reset Input High Voltage	3.8	V_{CC}	V	
V_{RL}	Reset Input Low Voltage	-0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -250\ \mu A$
V_{OL}	Output Low Voltage		0.4	V	$I_{OL} = +2.0\ mA$
I_{IL}	Input Leakage	-10	10	μA	$0\ V \le V_{IN} \le +5.25\ V$
I_{OL}	Output Leakage	-10	10	μA	$0\ V \le V_{IN} \le +5.25\ V$
I_{IR}	Reset Input Current		-50	μA	$V_{CC} = +5.25\ V,\ V_{RL} = 0\ V$
I_{CC}	V_{CC} Supply Current		150	mA	

AC Characteristics

External I/O or Memory Read and Write Timing

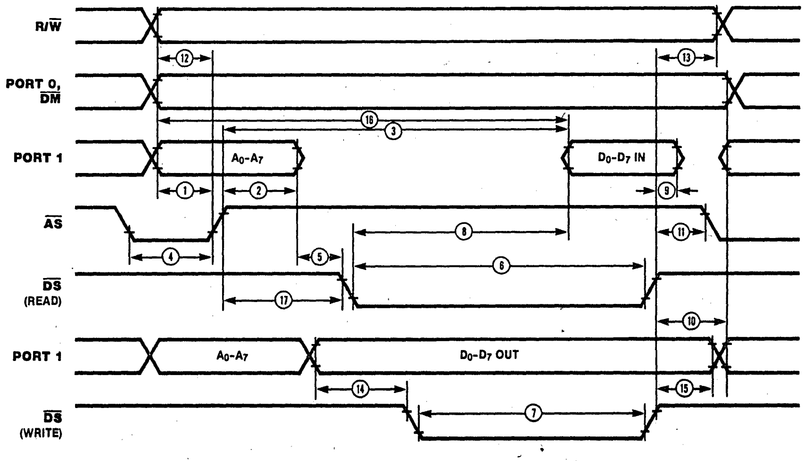


Figure 15. External I/O or Memory Read/Write

No.	Symbol	Parameter	Min	Max	Notes*†°
1	TdA(AS)	Address Valid to $\overline{\text{AS}}$ ↑ Delay	35		2,3
2	TdAS(A)	$\overline{\text{AS}}$ ↑ to Address Float Delay	45		2,3
3	TdAS(DR)	$\overline{\text{AS}}$ ↑ to Read Data Required Valid		220	1,2,3
4	TwAS	$\overline{\text{AS}}$ Low Width	55		1,2,3
5	TdAz(DS)	Address Float to $\overline{\text{DS}}$ ↓	0		
6	TwDSR	$\overline{\text{DS}}$ (Read) Low Width	185		1,2,3
7	TwDSW	$\overline{\text{DS}}$ (Write) Low Width	110		1,2,3
8	TdDSR(DR)	$\overline{\text{DS}}$ ↓ to Read Data Required Valid		130	1,2,3
9	ThDR(DS)	Read Data to $\overline{\text{DS}}$ ↑ Hold Time	0		
10	TdDS(A)	$\overline{\text{DS}}$ ↑ to Address Active Delay	45		2,3
11	TdDS(AS)	$\overline{\text{DS}}$ ↑ to $\overline{\text{AS}}$ ↓ Delay	55		2.3
12	TdR/W(AS)	R/$\overline{\text{W}}$ Valid to $\overline{\text{AS}}$ ↑ Delay	30		2,3
13	TdDS(R/W)	$\overline{\text{DS}}$ ↑ to R/$\overline{\text{W}}$ Not Valid	35		2,3
14	TdDW(DSW)	Write Data Valid to $\overline{\text{DS}}$ (Write) ↓ Delay	35		2,3
15	TdDS(DW)	$\overline{\text{DS}}$ ↑ to Write Data Not Valid Delay	45		2,3
16	TdA(DR)	Address Valid to Read Data Required Valid		255	1,2,3
17	TdAS(DS)	$\overline{\text{AS}}$ ↑ to $\overline{\text{DS}}$ ↓ Delay	55		2,3

NOTES:

1. When using extended memory timing add 2 TpC.
2. Timing numbers given are for minimum TpC.
3. See clock cycle time dependent characteristics table.

† Test Load 1.

° All timing references use 2.0 V for a logic "1" and 0.8 V for a logic "0".

* All units in nanoseconds (ns).

Additional Timing Table

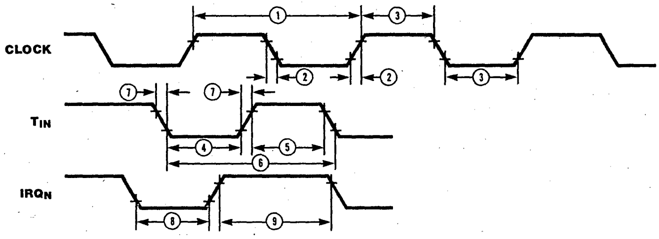


Figure 16. Additional Timing

No.	Symbol	Parameter	Min	Max	Notes*
1	TpC	Input Clock Period	80	1000	1
2	TrC,TfC	Clock Input Rise And Fall Times		15	1
3	TwC	Input Clock Width	26		1
4	TwTinL	Time Input Low Width	70		2
5	TwTinH	Timer Input High Width	3TpC		2
6	TpTin	Timer Input Period	8TpC		2
7	TrTin,TfTin	Timer Input Rise And Fall Times		100	2
8a	TwIL	Interrupt Request Input Low Time	70		2,3
8b	TwIL	Interrupt Request Input Low Time	3TpC		2,4
9	TwIH	Interrupt Request Input High Time	3TpC		2,3

NOTES:
1. Clock timing references uses 3.8 V for a logic "1" and 0.8 V for a logic "0".
2. Timing reference uses 2.0 V for a logic "1" and 0.8 V for a logic "0".
3. Interrupt request via Port 3 ($P3_1$-$P3_3$).
4. Interrupt request via Port 3 ($P3_0$).

* Units in nanoseconds (ns).

Memory Port Timing

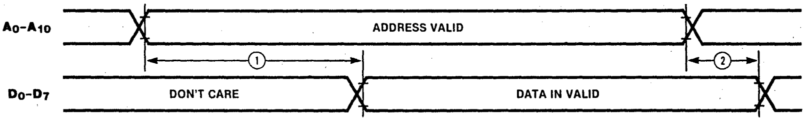


Figure 17. Memory Port Timing

No.	Symbol	Parameter	Min	Max	Notes*
1	TdA(DI)	Address Valid to Data Input Delay		320	1,2
2	ThDI(A)	Data In Hold time	0		1

NOTES:
1. Test Load 2.
2. This is a Clock-Cycle-Dependent parameter. For clock frequencies other than the maximum, use the following formula: 5 TpC – 95

*Units are nanoseconds unless otherwise specified.

Handshake Timing

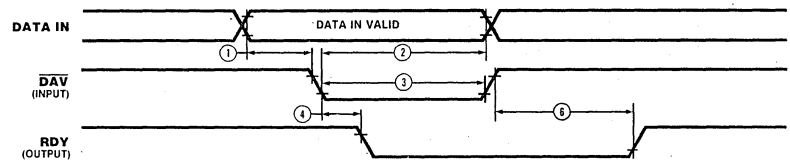


Figure 18a. Input Handshake

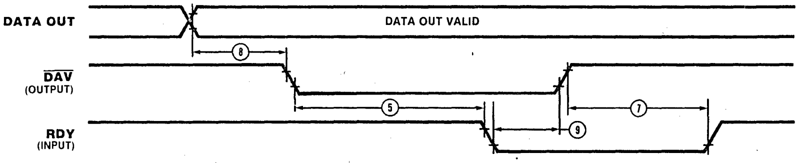


Figure 18b. Output Handshake

No.	Symbol	Parameter	Min	Max	Notes*
1	TsDI(DAV)	Data In Setup Time	0		
2	ThDI(DAV)	Data In Hold time	160		
3	TwDAV	Data Available Width	120		
4	TdDAVIf(RDY)	$\overline{\text{DAV}}$ ↓ Input to RDY ↓ Delay		120	1,2
5	TdDAVOf(RDY)	$\overline{\text{DAV}}$ ↓ Output to RDY ↓ Delay	0		1,3
6	TdDAVIr(RDY)	$\overline{\text{DAV}}$ ↑ Input to RDY ↑ Delay		120	1,2
7	TdDAV0r(RDY)	$\overline{\text{DAV}}$ ↑ Output to RDY ↑ Delay	0		1,3
8	TdDO(DAV)	Data Out to $\overline{\text{DAV}}$ ↓ Delay	30		1
9	TdRDY(DAV)	Rdy ↓ Input to $\overline{\text{DAV}}$ ↑ Delay	0	140	1

NOTES:
1. Test load 1
2. Input handshake
3. Output handshake
† All timing references use 2.0 V for a logic "1" and 0.8 V for a logic "0".

* Units in nanoseconds (ns).

Clock-Cycle-Time-Dependent Characteristics

Number	Symbol	Equation
1	TdA(AS)	TpC-50
2	TdAS(A)	TpC-40
3	TdAS(DR)	4TpC-110*
4	TwAS	TpC-30
5	TwDSR	3TpC-65*
7	TwDSW	2TpC-55*
8	TdDSR(DR)	3TpC-120*
10	Td(DS)A	TpC-40
11	TdDS(AS)	TpC-30
12	TdR/W(AS)	TpC-55
13	TdDS(R/W)	TpC-50
14	TdDW(DSW)	TpC-50
15	TdDS(DW)	TpC-40
16	TdA(DR)	5TpC-160*
17	TdAS(DS)	TpC-30

*Add 2TpC when using extended memory timing.

Zilog **Product Specification**

Z8671 Z8® MCU with BASIC/Debug Interpreter

June 1987

Z8671 MCU

FEATURES

- The Z8671 MCU is a complete microcomputer preprogrammed with a BASIC/Debug interpreter. Interaction between the interpreter and its user is provided through an on-board UART.
- BASIC/Debug can directly address the Z8671's internal registers and all external memory. It provides quick examination and modification of any external memory location or I/O port.
- The BASIC/Debug interpreter can call machine language subroutines to increase execution speed.
- The Z8671's auto start-up capability allows a program to be executed on power-up or Reset without operator intervention.
- Single +5V power supply—all I/O pins TTL-compatible.
- 8 MHz

GENERAL DESCRIPTION

The Z8671 Single-Chip Microcomputer (MCU) is one of a line of preprogrammed chips—in this case with a BASIC/Debug interpreter in ROM—offered by Zilog. As a member of the Z8 Family of microcomputers, it offers the same abundance of resources as the other Z8 microcomputers.

Because the BASIC/Debug interpreter is already part of the chip circuit, programming is made much easier. The Z8671 MCU thus offers a combination of software and hardware that is ideal for many industrial control applications. The Z8671 MCU allows fast hardware tests and bit-by-bit examination and modification of memory location, I/O ports,

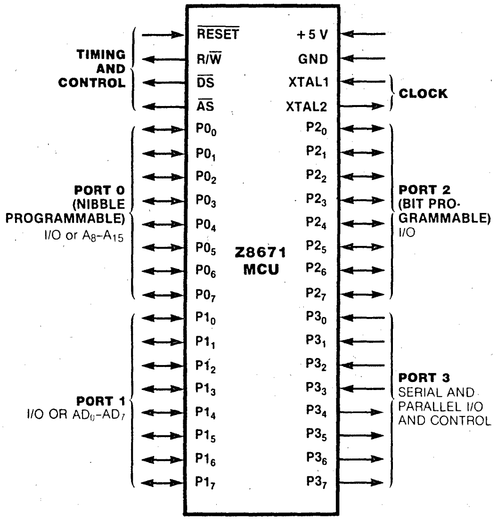

TIMING AND CONTROL
RESET
R/W
DS
AS
+5 V
GND
XTAL1
XTAL2
CLOCK
PORT 0 (NIBBLE PROGRAMMABLE)
I/O or A8-A15
P0₀ P0₁ P0₂ P0₃ P0₄ P0₅ P0₆ P0₇
Z8671 MCU
P2₀ P2₁ P2₂ P2₃ P2₄ P2₅ P2₆ P2₇
PORT 2 (BIT PROGRAMMABLE) I/O
PORT 1
I/O OR AD0-AD7
P1₀ P1₁ P1₂ P1₃ P1₄ P1₅ P1₆ P1₇
P3₀ P3₁ P3₂ P3₃ P3₄ P3₅ P3₆ P3₇
PORT 3
SERIAL AND PARALLEL I/O AND CONTROL

Figure 1. Pin Functions

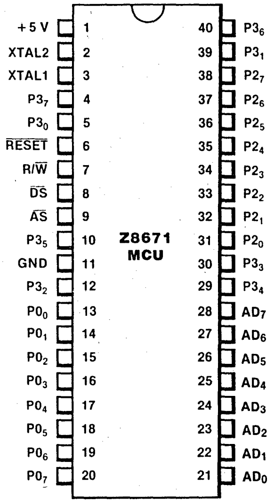


Figure 2a. 40-pin Dual-In-Line Package (DIP), Pin Assignments

or registers. It also allows bit manipulation and logical operations. A self-contained line editor supports interactive debugging, further speeding up program development.

The BASIC/Debug interpreter, a subset of Dartmouth BASIC, operates with three kinds of memory: on-chip registers and external ROM or RAM. The BASIC/Debug interpreter is located in the 2K bytes of on-chip ROM.

Additional features of the Z8671 MCU include the ability to call machine language subroutines to increase execution speed and the ability to have a program execute on power-up or Reset, without operator intervention.

Maximum memory addressing capabilities include 62K bytes of external program memory and 62K bytes of data memory with program storage beginning at location 800_H. This provides up to 124K bytes of useable memory space. Very few 8-bit microcomputers can directly access this amount of memory.

Each Z8671 Microcomputer has 32 I/O lines, a 144-byte register file, an on-board UART, and two counter/timers.

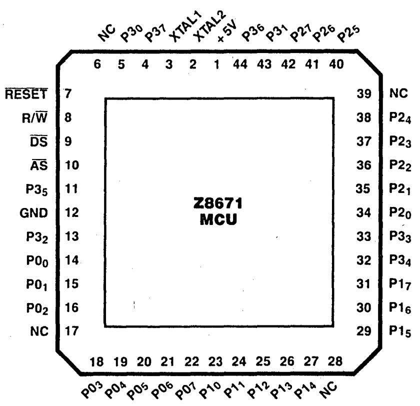


Figure 2b. 44-pin Chip Carrier, Pin Assignments

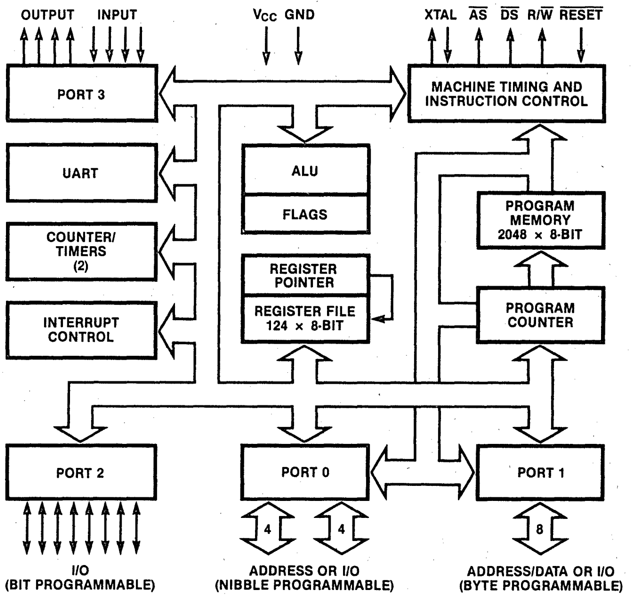

OUTPUT
INPUT
VCC
GND
XTAL
AS
DS
R/W
RESET
PORT 3
MACHINE TIMING AND INSTRUCTION CONTROL
UART
ALU
FLAGS
COUNTER/ TIMERS (2)
PROGRAM MEMORY 2048 × 8-BIT
REGISTER POINTER
REGISTER FILE 124 × 8-BIT
INTERRUPT CONTROL
PROGRAM COUNTER
PORT 2
PORT 0
PORT 1
4
4
8
I/O (BIT PROGRAMMABLE)
ADDRESS OR I/O (NIBBLE PROGRAMMABLE)
ADDRESS/DATA OR I/O (BYTE PROGRAMMABLE)

Figure 3. Functional Block Diagram

ARCHITECTURE

Z8671 architecture is characterized by a flexible I/O scheme, an efficient register and address space structure, and a number of ancillary features that are helpful in many applications.

Microcomputer applications demand powerful I/O capabilities. The Z8671 fulfills this with 32 pins dedicated to input and output. These lines are grouped into four ports of eight lines each and are configurable under software control to provide timing, status signals, serial or parallel I/O with or without handshake, and an address/data bus for interfacing external memory.

Because the multiplexed address/data bus is merged with the I/O-oriented ports, the Z8671 can assume many different memory and I/O configurations. These configurations range from a self-contained microcomputer to a microprocessor that can address 124K bytes of external memory.

Three basic address spaces are available to support this wide range of configurations: program memory (internal and external), data memory (external) and the register file (internal). The 144-byte random-access register file is composed of 124 general-purpose registers, four I/O port registers, and 16 control and status registers.

To unburden the program from coping with real-time problems such as serial data communication and counting/timing, an asynchronous receiver/transmitter (UART) and two counter/timers with a large number of userselectable modes are offered on-chip. Hardware support for the UART is minimized because one of the on-chip timers supplies the bit rate.

PIN DESCRIPTION

$\overline{\text{AS}}$. *Address Strobe* (output, active Low). Address Strobe is pulsed once at the beginning of each machine cycle. Addresses output via Port 1 for all external program or data memory transfers are valid at the trailing edge of $\overline{\text{AS}}$. Under program control, $\overline{\text{AS}}$ can be placed in the high-impedance state along with Ports 0 and 1, Data Strobe, and Read/Write.

$\overline{\text{DS}}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each external memory transfer.

$P0_0$-$P0_7$, $P1_0$-$P1_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. *I/O Port Lines* (input/outputs, TTL-compatible). These 32 lines are divided into four 8-bit I/O ports that can be configured under program control for I/O or external memory interface.

$\overline{\text{RESET}}$. *Reset* (input, active Low). $\overline{\text{RESET}}$ initializes the Z8671. When $\overline{\text{RESET}}$ is deactivated, program execution begins from internal program location $000C_H$.

R/$\overline{\text{W}}$. *Read/Write* (output). R/$\overline{\text{W}}$ is Low when the Z8671 is writing to external program or data memory.

XTAL1, XTAL2. *Crystal 1, Crystal 2* (time-base input and output). These pins connect a parallel-resonant crystal (8 MHz maximum) or an external single-phase clock (8 MHz maximum) to the on-chip clock oscillator and buffer.

ADDRESS SPACES

Program Memory. The Z8671's 16-bit program counter can address 64K bytes of program memory space. Program memory consists of 2K bytes of internal ROM and up to 62K bytes of external ROM, EPROM, or RAM. The first 12 bytes of program memory are reserved for interrupt vectors (Figure 4). These locations contain six 16-bit vectors that correspond to the six available interrupts. The BASIC/Debug interpreter is located in the 2K bytes of internal ROM. The interpreter begins at address 12 and extends to 2047.

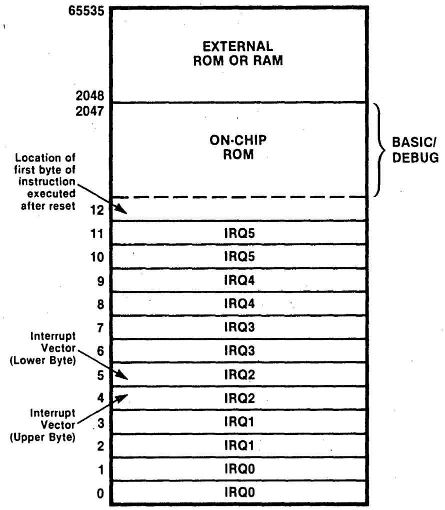


Figure 4. Program Memory Map

Data Memory. The Z8671 can address up to 62K bytes of external data memory beginning at location 2048 (Figure 5). External data memory may be included with, or separated from, the external program memory space. DM, an optional I/O function that can be programmed to appear on pin $P3_4$, is used to distinguish data and program memory space.

Register File. The 144-byte register file may be accessed by BASIC programs as memory locations 0-127 and 240-255. The register file includes four I/O port registers (R0-R3), 124 general-purpose registers (R4-R127), and 16 control and status registers (Figure 6).

The BASIC/Debug Interpreter uses many of the general-purpose registers as pointers, scratch workspace, and internal variables. Consequently, these registers cannot be used by a machine language subroutine or other user programs. On power-up/Reset, BASIC/Debug searches for external RAM memory and checks for an auto start-up program. In a non-destructive method, memory is tested at relative location $xxFD_H$. When BASIC/Debug discovers RAM in the system, it initializes the pointer registers to mark the boundaries between areas of memory that are assigned specific uses. The top page of RAM is allocated for the line buffer, variable storage, and the GOSUB stack. Figure 7a illustrates the contents of the general-purpose registers in the Z8671 system with external RAM. When BASIC/Debug tests memory and finds no RAM, it uses an internal stack and shares register space with the input line buffer and variables. Figure 7b illustrates the contents of the general-purpose registers in the Z8671 system without external RAM.

Stacks. Either the internal register file or the external data memory can be used for the stack. A 16-bit Stack Pointer (R254 and R255) is used for the external stack, which can reside anywhere in data memory between location 2048 and 65535. An 8-bit Stack Pointer (R255) is used for the internal stack that resides within the 124 general-purpose registers (R4-R127).

Register Addressing. Z8671 instructions can directly or indirectly access registers with an 8-bit address field. The Z8671 also allows short 4-bit register addressing using the Register Pointer, which is one of the control registers. In the 4-bit mode, the register file is divided into nine working-register groups, each group consisting of 16 contiguous registers (Figure 8). The Register Pointer addresses the starting location of the active working-register group.

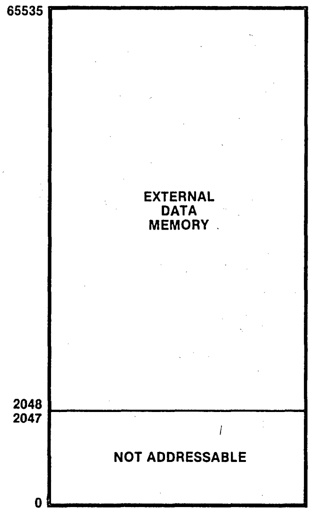


Figure 5. Data Memory Map

LOCATION		IDENTIFIERS
255	STACK POINTER (BITS 7-0)	SPL
254	STACK POINTER (BITS 15-8)	SPH
253	REGISTER POINTER	RP
252	PROGRAM CONTROL FLAGS	FLAGS
251	INTERRUPT MASK REGISTER	IMR
250	INTERRUPT REQUEST REGISTER	IRQ
249	INTERRUPT PRIORITY REGISTER	IPR
248	PORTS 0-1 MODE	P01M
247	PORT 3 MODE	P3M
246	PORT 2 MODE	P2M
245	T0 PRESCALER	PRE0
244	TIMER/COUNTER 0	T0
243	T1 PRESCALER	PRE1
242	TIMER/COUNTER 1	T1
241	TIMER MODE	TMR
240	SERIAL I/O	SIO
	NOT IMPLEMENTED	

Figure 6. Control and Status Registers

Register	Contents
127-104	SHARED BY EXPRESSION STACK AND LINE BUFFER
103-86	GOSUB STACK
85-64	SHARED BY GOSUB AND VARIABLES
63-34	VARIABLES
33	FREE, AVAILABLE FOR USR ROUTINES
32	COUNTER
31	USED INTERNALLY
30	SCRATCH
29-28	POINTER TO CONSTANT BLOCK
27-24	USED INTERNALLY
23-22	LINE NUMBER
21-20	ARGUMENT FOR SUBROUTINE CALL
19-18	ARGUMENT/RESULT FOR SUBROUTINE CALL
17-16	SCRATCH
15-14	POINTER TO NEXT CHARACTER
13-12	POINTER TO LINE BUFFER
11-10	POINTER TO GOSUB
9-8	POINTER TO BASIC PROGRAM
7-6	POINTER TO GOSUB
5-4	FREE
3-0	I/O PORTS

Figure 7a. General-Purpose Registers with External RAM

Register	Contents
127-64	EXPRESSION EVALUATION STACK
63-34	FREE
33-32	COUNTER
31	USED INTERNALLY
30	SCRATCH
29-28	POINTER TO CONSTANT BLOCK
27-24	USED INTERNALLY
23-22	LINE NUMBER
21-20	ARGUMENT FOR SUBROUTINE
19-18	ARGUMENT/ROUTINE FOR SUBROUTINE CALL
17-16	SCRATCH
15-14	POINTER TO INPUT LINE BUFFER
13-12	POINTER TO END OF LINE BUFFER
11-10	POINTER TO STACK BOTTOM
9-8	ADDRESS OF USER PROGRAM
7-6	POINTER TO GOSUB STACK
5-4	POINTER TO END OF PROGRAM
3-0	I/O PORTS

Figure 7b. General-Purpose Registers without External RAM

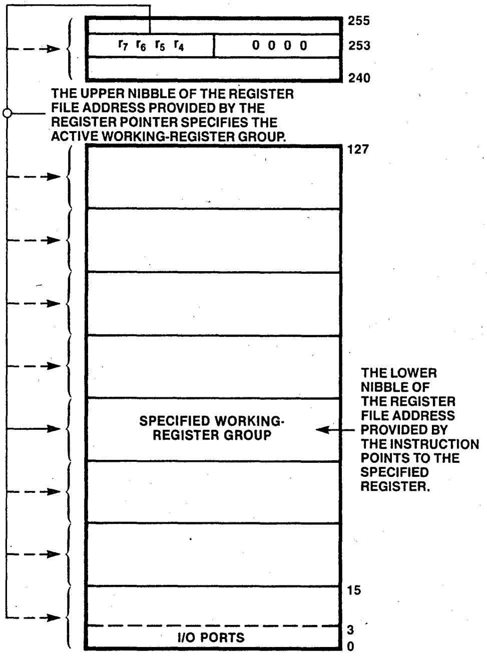

255
r_7 r_6 r_5 r_4
0 0 0 0
253
240
THE UPPER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE REGISTER POINTER SPECIFIES THE ACTIVE WORKING-REGISTER GROUP.
127
SPECIFIED WORKING-REGISTER GROUP
THE LOWER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE INSTRUCTION POINTS TO THE SPECIFIED REGISTER.
15
3
0
I/O PORTS

Figure 8. The Register Pointer

PROGRAM EXECUTION

Automatic Start-up. The Z8671 has an automatic start-up capability which allows a program stored in ROM to be executed without operator intervention. Automatic execution occurs on power-on or Reset when the program is stored at address 1020_H.

Execution Modes. The Z8671's BASIC/Debug Interpreter operates in two execution modes: Run and Immediate. Programs are edited and interactively debugged in the Immediate mode. Some BASIC/Debug commands are used almost exclusively in this mode. The Run mode is entered from the Immediate mode by entering the command RUN. If there is a program in RAM, it is executed. The system returns to the Immediate mode when program execution is complete or interrupted by an error.

INTERACTIVE DEBUGGING

Interactive debugging is accomplished with the self-contained line editor which operates in the Immediate mode. In addition to changing program lines, the editor can correct an immediate command before it is executed. It also allows the correction of typing and other errors as a program is entered.

BASIC/Debug allows interruptions and changes during a program run to correct errors and add new instructions without disturbing the sequential execution of the program. A program run is interrupted with the use of the escape key. The run is restarted with a GOTO command, followed by the appropriate line number, after the desired changes are entered. The same procedure is used to enter corrections after BASIC/Debug returns an error.

COMMANDS

BASIC/Debug recognizes 15 command keywords. For detailed instructions of command usage, refer to the *BASIC/Debug Software Reference Manual* (#03-3149-02).

FO — The GO command unconditionally branches to a machine language subroutine. This statement is similar to the USR function except that no value is returned by the assembly language routine.

GOSUB — GOSUB unconditionally branches to a subroutine at a line number specified by the user.

GOTO — GOTO unconditionally changes the sequence of program execution (branches to a line number).

IF/THEN — This command is used for conditional operations and branches.

INPUT/IN — These commands request information from the user with the prompt "?", then read the input values (which must be separated by commas) from the keyboard, and store them in the indicated variables. INPUT discards any values remaining in the buffer from previous IN, INPUT, or RUN statements, and requests new data from the operator. IN uses any values left in the buffer first, then requests new data.

LET — LET assigns the value of an expression to a variable or memory location.

LIST — This command is used in the interactive mode to generate a listing of program lines stored in memory on the terminal device.

NEW — The NEW command resets pointer R10-11 to the beginning of user memory, thereby marking the space as empty and ready to store a new program.

PRINT — PRINT lists its arguments, which may be text messages or numerical values, on the output terminal.

REM — This command is used to insert explanatory messages into the program.

RETURN — This command returns control to the line following a GOSUB statement.

RUN — RUN initiates sequential execution of all instructions in the current program.

STOP — STOP ends program execution and clears the GOSUB stack.

FUNCTIONS

BASIC/Debug supports two functions: AND and USR.

The AND function performs a logical AND. It can be used to mask, turn off, or isolate bits. This function is used in the following format:

AND (expression, expression)

The two expressions are evaluated, and their bit patterns are ANDed together. If only one value is included in the parentheses, it is ANDed with itself. A logical OR can also be performed by complementing the AND function. This is accomplished by subtracting each expression from −1. For example, the function below is equivalent to the OR of A and B.

−1-AND(−1-A, −1-B)

The USR function calls a machine language subroutine and returns a value. This is useful for applications in which a subroutine can be performed more quickly and efficiently in machine language than in BASIC/Debug.

The address of the first instruction of the subroutine is the first argument of the USR function. The address can be followed by one or two values to be processed by the subroutine. In the following example, BASIC/Debug executes the subroutine located at address 2000 using values literal 256 and variable C.

USR(%2000,256,C)

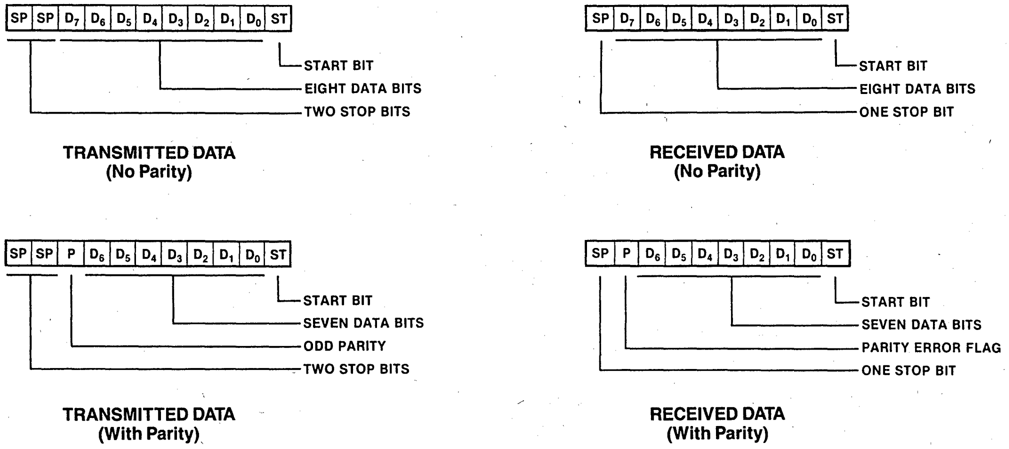
The resulting value is stored in Registers 18-19.

SERIAL INPUT/OUTPUT

Port 3 lines P3$_0$ and P3$_7$ can be programmed as serial I/O lines for full-duplex serial asynchronous receiver/transmitter operation. The bit rate is controlled by Counter/Timer 0, with a maximum rate of 62.5K bits/second.

The Z8671 automatically adds a start bit and two stop bits to transmitted data (Figure 9). Odd parity is also available as an option. Eight data bits are always transmitted, regardless of parity selection. If parity is enabled, the eighth data bit is used as the odd parity bit. An interrupt request (IRQ4) is generated on all transmitted characters.

Received data must have a start bit, eight data bits, and at least one stop bit. If parity is on, bit 7 of the received data is replaced by a parity error flag. Received characters generate the IRQ3 interrupt request.



TRANSMITTED DATA (No Parity)

RECEIVED DATA (No Parity)

TRANSMITTED DATA (With Parity)

RECEIVED DATA (With Parity)

Figure 9. Serial Data Formats

I/O PORTS

The Z8671 has 32 lines dedicated to input and output. These lines are grouped into four ports of eight lines each and are configurable as input, output or address/data. Under software control, the ports can be programmed to provide address outputs, timing, status signals, serial I/O, and parallel I/O with or without handshake. All ports have active pull-ups and pull-downs compatible with TTL loads.

Port 1 can be programmed as a byte I/O port or as an address/data port for interfacing external memory. When used as an I/O port, Port 1 may be placed under handshake control. In this configuration, Port 3 lines $P3_3$ and $P3_4$ are used as the handshake controls RDY1 and $\overline{DAV1}$ (Ready and Data Available).

Memory locations greater than 2048 are referenced through Port 1. To interface external memory, Port 1 must be programmed for the multiplexed Address/Data mode. If more than 256 external locations are required, Port 0 must output the additional lines.

Port 1 can be placed in the high-impedance state along with Port 0, $\overline{AS}$, $\overline{DS}$ and R/$\overline{W}$, allowing the Z8671 to share common resources in multiprocessor and DMA applications. Data transfers can be controlled by assigning $P3_3$ as a Bus Acknowledge input and $P3_4$ as a Bus Request output.

Port 0 can be programmed as a nibble I/O port, or as an address port for interfacing external memory. When used as an I/O port, Port 0 may be placed under handshake control. In this configuration, Port 3 lines $P3_2$ and $P3_5$ are used as the handshake controls $\overline{DAV0}$ and RDY0. Handshake signal assignment is dictated by the I/O direction of the upper nibble $P0_4$-$P0_7$.

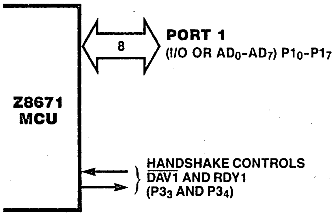
For external memory references, Port 0 can provide address bits A_8-A_{11} (lower nibble) or A_8-A_{15} (lower and upper nibble) depending on the required address space. If the address range requires 12 bits or less, the upper nibble of Port 0 can be programmed independently as I/O while the lower nibble is used for addressing. When Port 0 nibbles are defined as address bits, they can be set to the high-impedance state along with Port 1 and the control signals $\overline{AS}$, $\overline{DS}$ and R/$\overline{W}$.

Port 2 bits can be programmed independently as input or output. The port is always available for I/O operations. In addition, Port 2 can be configured to provide open-drain outputs.

Like Ports 0 and 1, Port 2 may also be placed under handshake control. In this configuration, Port 3 lines $P3_1$ and $P3_6$ are used as the handshake controls lines $\overline{DAV2}$ and RDY2. The handshake signal assignment for Port 3 lines $P3_1$ and $P3_6$ is dictated by the direction (input or output) assigned to bit 7 of Port 2.

Port 3 lines can be configured as I/O or control lines. In either case, the direction of the eight lines is fixed as four input ($P3_0$-$P3_3$) and four output ($P3_4$-$P3_7$). For serial I/O, lines $P3_0$ and $P3_7$ are programmed as serial in and serial out respectively.

Port 3 can also provide the following control functions: handshake for Ports 0, 1 and 2 ($\overline{DAV}$ and RDY); four external interrupt request signals (IRQ0-IRQ3); timer input and output signals (T_{IN} and T_{OUT}) and Data Memory Select ($\overline{DM}$).



Z8671 MCU
8
PORT 1
(I/O OR AD_0-AD_7) $P1_0$-$P1_7$
HANDSHAKE CONTROLS
$\overline{DAV1}$ AND RDY1
($P3_3$ AND $P3_4$)

Figure 10a. Port 1

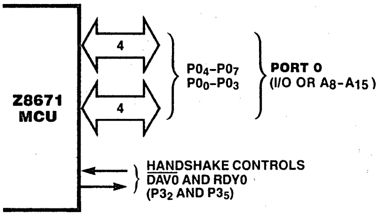


Figure 10b. Port 0

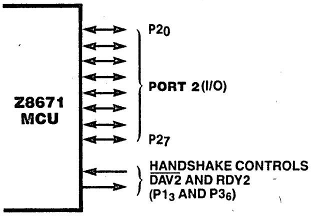


Figure 10c. Port 2

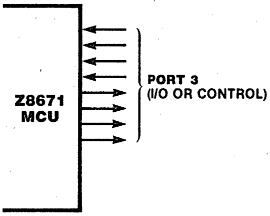


Figure 10d. Port 3

COUNTER/TIMERS

The Z8671 contains two 8-bit programmable counter/timers (T0 and T1), each driven by its own 6-bit programmable prescaler. The T1 prescaler can be driven by internal or external clock sources; however, the T0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrements the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request—IRQ4 (T_0) or IRQ5 (T_1)—is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single-pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read any time without disturbing their value or count mode.

The clock source for T1 is user-definable; it can be either the internal microprocessor clock (4 MHz maximum) divided by four, or an external signal input via Port 3. The Timer Mode register configures the external timer input as an external clock, a trigger input that can be retriggerable or nonretriggerable, or as a gate input for the internal clock. The counter/timers can be programmably cascaded by connecting the T0 output to the input of T1. Port 3 line $P3_6$ also serves as a timer output (T_{OUT}) through which T0, T1 or the internal clock can be output.

Z8671 MCU

INTERRUPTS

The Z8671 allows six different interrupts from eight sources: the four Port 3 lines $P3_0$-$P3_3$, Serial In, Serial Out, and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask register globally or individually enables or disables the six interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All Z8671 interrupts are vectored; however, the internal UART operates in a polling fashion. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the Interrupt Request register polled to determine which of the interrupt requests needs service.

The BASIC/Debug Interpreter does not process interrupts. Interrupts are vectored through locations in internal ROM which point to addresses $1000\text{-}1011_H$. To process interrupts, jump instructions can be entered to the interrupt handling routines at the appropriate addresses as shown in Table 1.

Table 1. Interrupt Jump Instructions

Hex Address	Contains Jump Instruction and Subroutine Address for:
1000-1002	IRQ0
1003-1005	IRQ1
1006-1008	IRQ2
1009-100B	IRQ3
100C-100E	IRQ4
100F-1011	IRQ5

CLOCK

The on-chip oscillator has a high-gain, parallel-resonant amplifier for connection to a crystal or to any suitable external clock source (XTAL1 = Input, XTAL2 = Output).

The crystal source is connected across XTAL1 and XTAL2, using the recommended capacitance (C_L = 15 pf maximum) from each pin to ground. The specifications for the crystal are as follows:

- AT cut, parallel resonant
- Fundamental type, 8 maximum
- Series resistance, $R \leqslant 100\ \Omega$
- 8 MHz maximum

INSTRUCTION SET NOTATION

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "9". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*****	Set or cleared according to operation
—	Unaffected
X	Undefined

CONDITION CODES

Value	Mnemonic	Meaning	Flags Set
1000		Always true	—
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	—

INSTRUCTION FORMATS

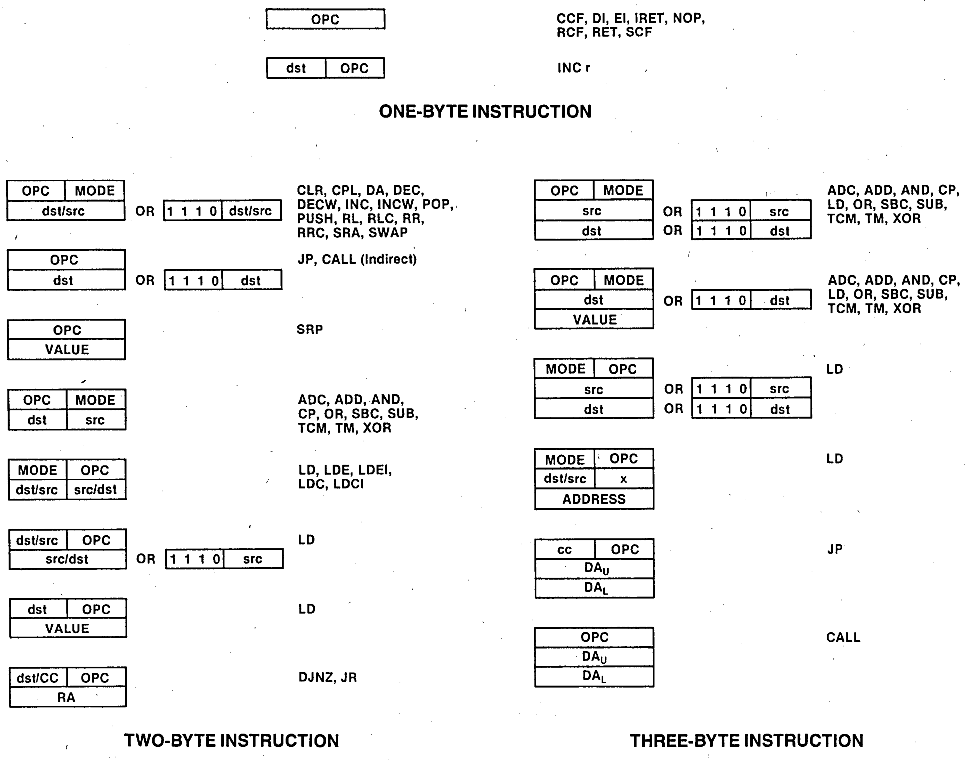
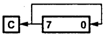
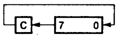
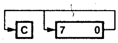


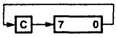
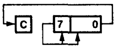
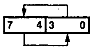
Figure 11. Instruction Formats

INSTRUCTION SUMMARY

Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
ADC dst,src	(Note 1)		1□	*	*	*	*	0	*
dst ← dst + src + C									
ADD dst,src	(Note 1)		0□	*	*	*	*	0	*
dst ← dst + src									
AND dst,src	(Note 1)		5□	—	*	*	0	—	—
dst ← dst AND src									
CALL dst	DA		D6	—	—	—	—	—	—
SP ← SP − 2	IRR		D4						
@SP ← PC; PC ← dst									
CCF			EF	*	—	—	—	—	—
C ← NOT C									
CLR dst	R		B0	—	—	—	—	—	—
dst ← 0	IR		B1						
COM dst	R		60	—	*	*	0	—	—
dst ← NOT dst	IR		61						
CP dst,src	(Note 1)		A□	*	*	*	*	—	—
dst − src									
DA dst	R		40	*	*	*	X	—	—
dst ← DA dst	IR		41						
DEC dst	R		00	—	*	*	*	—	—
dst ← dst − 1	IR		01						
DECW dst	RR		80	—	*	*	*	—	—
dst ← dst − 1	IR		81						
DI									
IMR (7) ← 0			8F	—	—	—	—	—	—
DJNZ r,dst	RA		rA	—	—	—	—	—	—
r ← r − 1			r = 0 − F						
if r ≠ 0									
PC ← PC + dst									
Range: +127, −128									
EI			9F	—	—	—	—	—	—
IMR (7) ← 1									
INC dst	r		rE	—	*	*	*	—	—
dst ← dst + 1			r = 0 − F						
	R		20						
	IR		21						
INCW dst	RR		A0	—	*	*	*	—	—
dst ← dst + 1	IR		A1						
IRET			BF	*	*	*	*	*	*
FLAGS ← @SP; SP ← SP + 1									
PC ← @SP; SP ← SP + 2; IMR (7) ← 1									
JP cc,dst	DA		cD	—	—	—	—	—	—
if cc is true			c = 0 − F						
PC ← dst	IRR		30						

Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
JR cc,dst	RA		cB	—	—	—	—	—	—
if cc is true,			c = 0 − F						
PC ← PC + dst									
Range: +127, −128									
LD dst,src	r	Im	rC	—	—	—	—	—	—
dst ← src	r	R	r8						
	R	r	r9						
			r = 0 − F						
	r	X	C7						
	X	r	D7						
	r	Ir	E3						
	Ir	r	F3						
	R	R	E4						
	R	IR	E5						
	R	IM	E6						
	IR	IM	E7						
	IR	R	F5						
LDC dst,src	r	Irr	C2	—	—	—	—	—	—
dst ← src	Irr	r	D2						
LDCI dst,src	Ir	Irr	C3	—	—	—	—	—	—
dst ← src	Irr	Ir	D3						
r ← r + 1; rr ← rr + 1									
LDE dst,src	r	Irr	82	—	—	—	—	—	—
dst ← src	Irr	r	92						
LDEI dst,src	Ir	Irr	83	—	—	—	—	—	—
dst ← src	Irr	Ir	93						
r ← r + 1; rr ← rr + 1									
NOP			FF	—	—	—	—	—	—
OR dst,src	(Note 1)		4□	—	*	*	0	—	—
dst ← dst OR src									
POP dst	R		50	—	—	—	—	—	—
dst ← @SP;	IR		51						
SP ← SP + 1									
PUSH src		R	70	—	—	—	—	—	—
SP ← SP − 1; @SP ← src		IR	71						
RCF			CF	0	—	—	—	—	—
C ← 0									
RET			AF	—	—	—	—	—	—
PC ← @SP; SP ← SP + 2									
RL dst 	R		90	*	*	*	*	—	—
	IR		91						
RLC dst 	R		10	*	*	*	*	—	—
	IR		11						
RR dst 	R		E0	*	*	*	*	—	—
	IR		E1						

INSTRUCTION SUMMARY (Continued)

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
RRC dst 	R IR		C0 C1	*	*	*	*	—	—
SBC dst,src dst ← dst ← src ← C	(Note 1)		3□	*	*	*	*	1	*
SCF C ← 1			DF	1	—	—	—	—	—
SRA dst 	R IR		D0 D1	*	*	*	0	—	—
SRP src RP ← src		Im	31	—	—	—	—	—	—
SUB dst,src dst ← dst ← src	(Note 1)		2□	*	*	*	*	1	*
SWAP dst 	R IR		F0 F1	X	*	*	X	—	—
TCM dst,src (NOT dst) AND src	(Note 1)		6□	—	*	*	0	—	—
TM dst,src dst AND src	(Note 1)		7□	—	*	*	0	—	—

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
XOR dst,src dst ← dst XOR src	(Note 1)		B□	—	*	*	0	—	—

NOTE: These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the left of the applicable addressing mode pair.

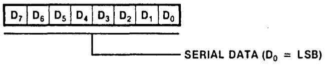
For example, the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

Addr Mode dst	Addr Mode src	Lower Opcode Nibble
r	r	2
r	Ir	3
R	R	4
R	IR	5
R	IM	6
IR	IM	7

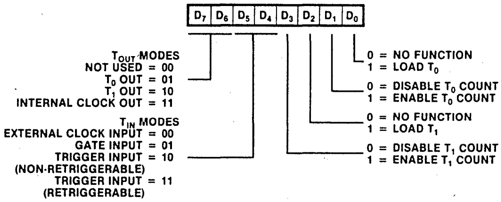
Z8671 MCU

REGISTERS

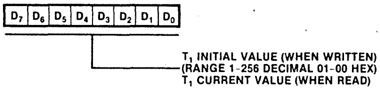
R240 SIO
Serial I/O Register
($F0_H$; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
SERIAL DATA (D_0 = LSB)

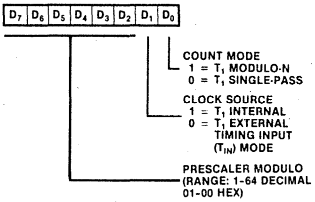
R241 TMR
Time Mode Register
($F1_H$; Read/Write)



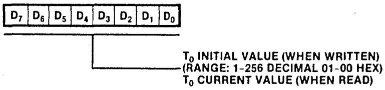
R242 T1
Counter Timer 1 Register
($F2_H$; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_1 INITIAL VALUE (WHEN WRITTEN)
(RANGE 1-256 DECIMAL 01-00 HEX)
T_1 CURRENT VALUE (WHEN READ)

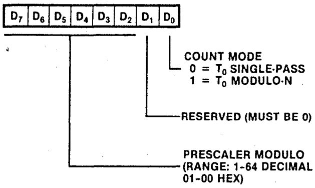
R243 PRE1
Prescaler 1 Register
($F3_H$; Write Only)



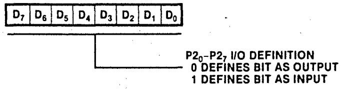
R244 TO
Counter/Timer 0 Register
($F4_H$; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_0 INITIAL VALUE (WHEN WRITTEN)
(RANGE: 1-256 DECIMAL 01-00 HEX)
T_0 CURRENT VALUE (WHEN READ)

R245 PRE0
Prescaler 0 Register
($F5_H$; Write Only)



R246 P2M
Port 2 Mode Register
($F6_H$; Write Only)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
$P2_0$-$P2_7$ I/O DEFINITION
0 DEFINES BIT AS OUTPUT
1 DEFINES BIT AS INPUT

R247 P3M
Port 3 Mode Register
($F7_H$; Write Only)

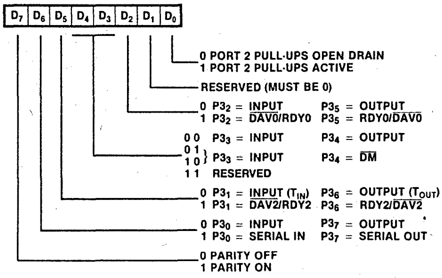
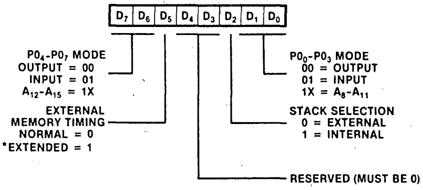

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
RESERVED (MUST BE 0)
0 $P3_2$ = INPUT $P3_5$ = OUTPUT
1 $P3_2$ = $\overline{DAV0}$/RDY0 $P3_5$ = RDY0/$\overline{DAV0}$
0 0 $P3_3$ = INPUT $P3_4$ = OUTPUT
0 1 } 1 0 $P3_3$ = INPUT $P3_4$ = $\overline{DM}$
1 1 RESERVED
0 $P3_1$ = INPUT (T_{IN}) $P3_6$ = OUTPUT (T_{OUT})
1 $P3_1$ = $\overline{DAV2}$/RDY2 $P3_6$ = RDY2/$\overline{DAV2}$
0 $P3_0$ = INPUT $P3_7$ = OUTPUT
1 $P3_0$ = SERIAL IN $P3_7$ = SERIAL OUT
0 PARITY OFF
1 PARITY ON

Figure 12. Control Registers

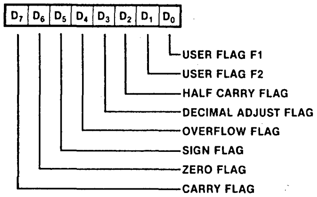
REGISTERS
(Continued)

R248 P01M
Port 0 Register
(F8H; Write Only)

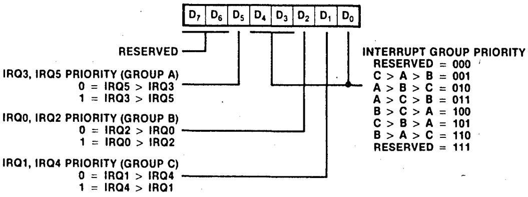


*ALWAYS EXTENDED TIMING AFTER RESET

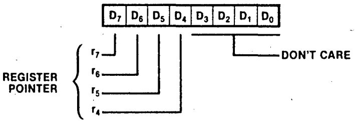
R252 FLAGS
Flag Register
(FCH; Read/Write)



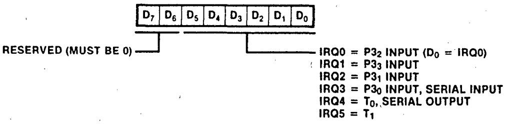
R249 IPR
Interrupt Priority Register
(F9H; Write Only)



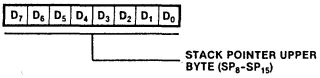
R253 RP
Register Pointer
(FDH; Read/Write)



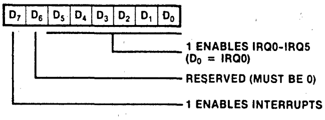
R250 IRQ
Interrupt Request Register
(FAH; Read/Write)



R254 SPH
Stack Pointer
(FEH; Read/Write)



R251 IMR
Interrupt Mask Register
(FBH; Read/Write)



R255 SPL
Stack Pointer
(FFH; Read/Write)

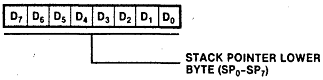


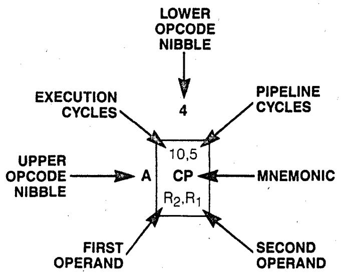
Figure 12. Control Registers (Continued)

Z8671 MCU

OPCODE MAP

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6,5 **DEC** R_1	6,5 **DEC** IR_1	6,5 **ADD** r_1,r_2	6,5 **ADD** r_1,Ir_2	10,5 **ADD** R_2,R_1	10,5 **ADD** IR_2,R_1	10,5 **ADD** R_1,IM	10,5 **ADD** IR_1,IM	6,5 **LD** r_1,R_2	6,5 **LD** r_2,R_1	12/10,5 **DJNZ** r_1,RA	12/10,0 **JR** cc,RA	6,5 **LD** r_1,IM	12/10,0 **JP** cc,DA	6,5 **INC** r1	
1	6,5 **RLC** R_1	6,5 **RLC** IR_1	6,5 **ADC** r_1,r_2	6,5 **ADC** r_1,Ir_2	10,5 **ADC** R_2,R_1	10,5 **ADC** IR_2,R_1	10,5 **ADC** R_1,IM	10,5 **ADC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
2	6,5 **INC** R_1	6,5 **INC** IR_1	6,5 **SUB** r_1,r_2	6,5 **SUB** r_1,Ir_2	10,5 **SUB** R_2,R_1	10,5 **SUB** IR_2,R_1	10,5 **SUB** R_1,IM	10,5 **SUB** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
3	8,0 **JP** IRR_1	6,1 **SRP** IM	6,5 **SBC** r_1,r_2	6,5 **SBC** r_1,Ir_2	10,5 **SBC** R_2,R_1	10,5 **SBC** IR_2,R_1	10,5 **SBC** R_1,IM	10,5 **SBC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
4	8,5 **DA** R_1	8,5 **DA** IR_1	6,5 **OR** r_1,r_2	6,5 **OR** r_1,Ir_2	10,5 **OR** R_2,R_1	10,5 **OR** IR_2,R_1	10,5 **OR** R_1,IM	10,5 **OR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
5	10,5 **POP** R_1	10,5 **POP** IR_1	6,5 **AND** r_1,r_2	6,5 **AND** r_1,Ir_2	10,5 **AND** R_2,R_1	10,5 **AND** IR_2,R_1	10,5 **AND** R_1,IM	10,5 **AND** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
6	6,5 **COM** R_1	6,5 **COM** IR_1	6,5 **TCM** r_1,r_2	6,5 **TCM** r_1,Ir_2	10,5 **TCM** R_2,R_1	10,5 **TCM** IR_2,R_1	10,5 **TCM** R_1,IM	10,5 **TCM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
7	10/12,1 **PUSH** R_2	12/14,1 **PUSH** IR_2	6,5 **TM** r_1,r_2	6,5 **TM** r_1,Ir_2	10,5 **TM** R_2,R_1	10,5 **TM** IR_2,R_1	10,5 **TM** R_1,IM	10,5 **TM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
8	10,5 **DECW** RR_1	10,5 **DECW** IR_1	12,0 **LDE** r_1,Irr_2	18,0 **LDEI** Ir_1,Irr_2					↓	↓	↓	↓	↓	↓	↓	6,1 **DI**
9	6,5 **RL** R_1	6,5 **RL** IR_1	12,0 **LDE** r_2,Irr_1	18,0 **LDEI** Ir_2,Irr_1					↓	↓	↓	↓	↓	↓	↓	6,1 **EI**
A	10,5 **INCW** RR_1	10,5 **INCW** IR_1	6,5 **CP** r_1,r_2	6,5 **CP** r_1,Ir_2	10,5 **CP** R_2,R_1	10,5 **CP** IR_2,R_1	10,5 **CP** R_1,IM	10,5 **CP** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	14,0 **RET**
B	6,5 **CLR** R_1	6,5 **CLR** IR_1	6,5 **XOR** r_1,r_2	6,5 **XOR** r_1,Ir_2	10,5 **XOR** R_2,R_1	10,5 **XOR** IR_2,R_1	10,5 **XOR** R_1,IM	10,5 **XOR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	16,0 **IRET**
C	6,5 **RRC** R_1	6,5 **RRC** IR_1	12,0 **LDC** r_1,Irr_2	18,0 **LDCI** Ir_1,Irr_2				10,5 **LD** r_1,x,R_2	↓	↓	↓	↓	↓	↓	↓	6,5 **RCF**
D	6,5 **SRA** R_1	6,5 **SRA** IR_1	12,0 **LDC** r_2,Irr_1	18,0 **LDCI** Ir_2,Irr_1	20,0 **CALL*** IRR_1		20,0 **CALL** DA	10,5 **LD** r_2,x,R_1	↓	↓	↓	↓	↓	↓	↓	6,5 **SCF**
E	6,5 **RR** R_1	6,5 **RR** IR_1		6,5 **LD** r_1,IR_2	10,5 **LD** R_2,R_1	10,5 **LD** IR_2,R_1	10,5 **LD** R_1,IM	10,5 **LD** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	6,5 **CCF**
F	8,5 **SWAP** R_1	8,5 **SWAP** IR_1		6,5 **LD** Ir_1,r_2		10,5 **LD** R_2,IR_1			↓	↓	↓	↓	↓	↓	↓	6,0 **NOP**
Bytes per Instruction	2				3				2					3	1	



Legend:
R = 8-bit address
r = 4-bit address
R_1 or r_1 = Dst address
R_2 or r_2 = Src address

Sequence:
Opcode, First Operand, Second Operand

NOTE: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

ABSOLUTE MAXIMUM RATINGS

Voltages on all pins with respect
to GND . . . . –0.3V to +7.0V
Operating Ambient
Temperature . . . . .See Ordering Information
Storage Temperature . . . . –65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin.

Standard conditions are:

- $+4.75V \leqslant V_{CC} \leqslant +5.25V$
- GND = 0V
- $0°C \leqslant T_A \leqslant +70°C$

The Ordering Information section lists package temperature ranges and product numbers. Package drawings are in the Package Information section. Refer to the Literature List for additional documentation.

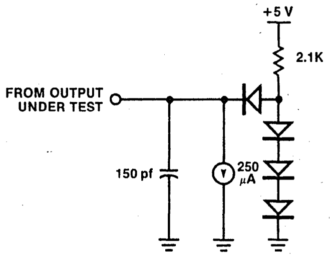


Figure 13. Test Load 1

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8	V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	–0.3	0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{IL}	Input Low Voltage	–0.3	0.8	V	
V_{RH}	Reset Input High Voltage	3.8	V_{CC}	V	
V_{RL}	Reset Input Low Voltage	–0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -250\,\mu A$
V_{OL}	Output Low Voltage		0.4	V	$I_{OL} = +2.0\,mA$
I_{IL}	Input Leakage	–10	10	μA	$0V \leqslant V_{IN} \leqslant +5.25V$
I_{OL}	Output Leakage	–10	10	μA	$0V \leqslant V_{IN} \leqslant +5.25V$
I_{IR}	Reset Input Current		–50	μA	$V_{CC} = +5.25V, V_{RL} = 0V$
I_{CC}	V_{CC} Supply Current		180	mA	

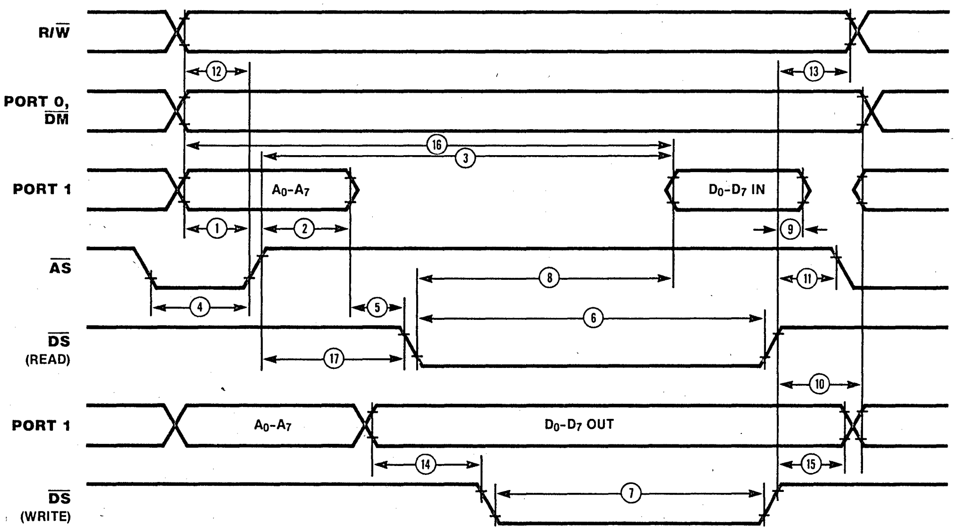


Figure 16. External I/O or Memory Read/Write

AC CHARACTERISTICS

External I/O or Memory Read/Write Timing

No.	Symbol	Parameter	Min	Max	Notes*†°
1	TdA(AS)	Address Valid to $\overline{\text{AS}}$ ↑ Delay	35		2,3
2	TdAS(A)	$\overline{\text{AS}}$ ↑ to Address Float Delay	45		2,3
3	TdAS(DR)	$\overline{\text{AS}}$ ↑ to Read Data Required Valid		220	1,2,3
4	TwAS	$\overline{\text{AS}}$ Low Width	55		1,2,3
5	TdAz(DS)	Address Float to $\overline{\text{DS}}$ ↓	0		
6	TwDSR	$\overline{\text{DS}}$ (Read) Low Width	185		1,2,3
7	TwDSW	$\overline{\text{DS}}$ (Write) Low Width	110		1,2,3
8	TdDSR(DR)	$\overline{\text{DS}}$ ↓ to Read Data Required Valid		130	1,2,3
9	ThDR(DS)	Read Data to $\overline{\text{DS}}$ ↑ Hold Time	0		
10	TdDS(A)	$\overline{\text{DS}}$ ↑ to Address Active Delay	45		2,3
11	TdDS(AS)	$\overline{\text{DS}}$ ↑ to $\overline{\text{AS}}$ ↓ Delay	55		2.3
12	TdR/W(AS)	R/$\overline{\text{W}}$ Valid to $\overline{\text{AS}}$ ↑ Delay	30		2,3
13	TdDS(R/W)	$\overline{\text{DS}}$ ↑ to R/$\overline{\text{W}}$ Not Valid	35		2,3
14	TdDW(DSW)	Write Data Valid to $\overline{\text{DS}}$ (Write) ↓ Delay	35		2,3
15	TdDS(DW)	$\overline{\text{DS}}$ ↑ to Write Data Not Valid Delay	45		2,3
16	TdA(DR)	Address Valid to Read Data Required Valid		255	1,2,3
17	TdAS(DS)	$\overline{\text{AS}}$ ↑ to $\overline{\text{DS}}$ ↓ Delay	55		2,3

NOTES:
1. When using extended memory timing add 2 TpC.
2. Timing numbers given are for minimum TpC.
3. See clock cycle time dependent characteristics table.

† Test Load 1.
° All timing references use 2.0 V for a logic "1" and 0.8 V for a logic "0".
* All units in nanoseconds (ns).

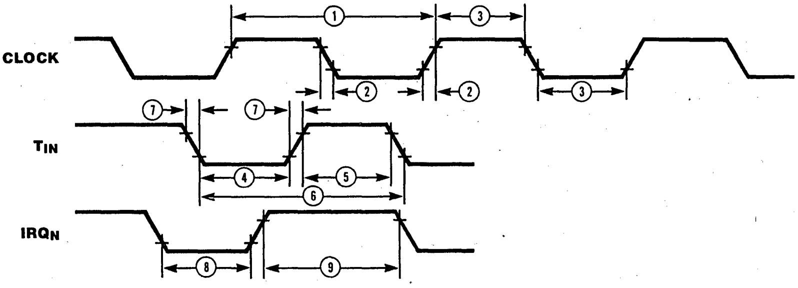


Figure 17. Additional Timing

AC CHARACTERISTICS

Additional Timing

No.	Symbol	Parameter	Min	Max	Notes*
1	TpC	Input Clock Period	80	1000	1
2	TrC,TfC	Clock Input Rise And Fall Times		15	1
3	TwC	Input Clock Width	26		1
4	TwTinL	Time Input Low Width	70		2
5	TwTinH	Timer Input High Width	3TpC		2
6	TpTin	Timer Input Period	8TpC		2
7	TrTin,TfTin	Timer Input Rise And Fall Times		100	2
8a	TwIL	Interrupt Request Input Low Time	70		2,3
8b	TwIL	Interrupt Request Input Low Time	3TpC		2,4
9	TwIH	Interrupt Request Input High Time	3TpC		2,3

NOTES:
1. Clock timing references uses 3.8 V for a logic "1",and 0.8 V for a logic "0".
2. Timing reference uses 2.0 V for a logic "1" and 0.8 V for a logic "0".
3. Interrupt request via Port 3 ($P3_1$-$P3_3$).
4. Interrupt request via Port 3 ($P3_0$).

* Units in nanoseconds (ns).

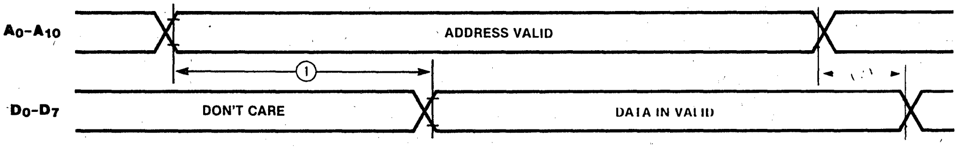


Figure 18. Memory Port Timing

AC CHARACTERISTICS

Memory Port Timing

No.	Symbol	Parameter	Min	Max	Notes*
1	TdA(DI)	Address Valid to Data Input Delay		320	1,2
2	ThDI(A)	Data In Hold time	0		1

NOTES:
1. Test Load 2.
2. This is a Clock-Cycle-Dependent parameter. For clock frequencies other than the maximum, use the following formula: 5 TpC – 95

*Units are nanoseconds unless otherwise specified.

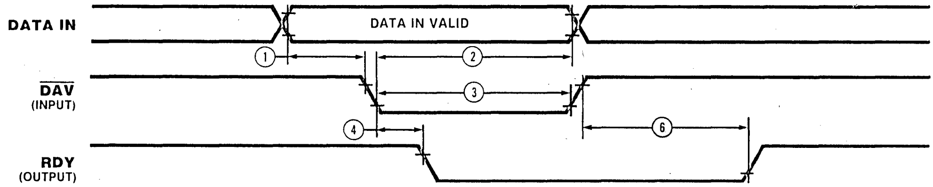


Figure 18a. Input Handshake

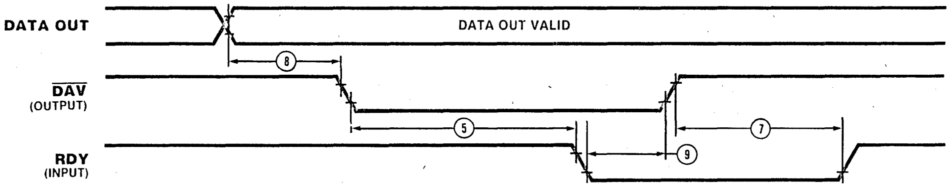


Figure 18b. Output Handshake

AC CHARACTERISTICS

Handshake Timing

No.	Symbol	Paramoter	Min	Max	Notos*
1	TsDI(DAV)	Data In Setup Time	0		
2	ThDI(DAV)	Data In Hold time	160		
3	TwDAV	Data Available Width	120		
4	TdDAVIf(RDY)	$\overline{\text{DAV}}$ ↓ Input to RDY ↓ Delay		120	1,2
5	TdDAVOf(RDY)	$\overline{\text{DAV}}$ ↓ Output to RDY ↓ Delay	0		1,3
6	TdDAVIr(RDY)	$\overline{\text{DAV}}$ ↑ Input to RDY ↑ Delay		120	1,2
7	TdDAV0r(RDY)	$\overline{\text{DAV}}$ ↑ Output to RDY ↑ Delay	0		1,3
8	TdDO(DAV)	Data Out to $\overline{\text{DAV}}$ ↓ Delay	30		1
9	TdRDY(DAV)	Rdy ↓ Input to $\overline{\text{DAV}}$ ↑ Delay	0	140	1

NOTES:
1. Test load 1
2. Input handshake
3. Output handshake
† All timing references use 2.0 V for a logic "1" and 0.8 V for a logic "0".

* Units in nanoseconds (ns).

CLOCK CYCLE TIME-DEPENDENT CHARACTERISTICS

Number	Symbol	Z8671-8 Equation	Number	Symbol	Z8671-8 Equation
1	TdA(AS)	TpC − 75	13	TdDS(R/W)	TpC − 65
2	TdAS(A)	TpC − 55	14	TdDW(DSW)	TpC − 75
3	TdAS(DR)	4TpC − 140*	15	TdDS(DW)	TpC − 55
4	TwAS	TpC − 45	16	TdA(DR)	5TpC − 215*
6	TwDSR	3TpC − 125*	17	TdAS(DS)	TpC − 45
7	TwDSW	2TpC − 90*			
8	TdDSR(DR)	3TpC − 175*			
10	Td(DS)A	TpC − 55			
11	TdDS(AS)	TpC − 55			
12	TdR/W(AS)	TpC − 75			

* Add 2TpC when using extended memory timing

Zilog **Product Specification**

Z8681/82 Z8®
ROMless MCU

June 1987

FEATURES

- Complete microcomputer, 24 I/O lines, and up to 64K bytes of addressable external space each for program and data memory.
- 143-byte register file, including 124 general-purpose registers, 3 I/O port registers, and 16 status and control registers.
- Vectored, priority interrupts for I/O, counter/timers, and UART.
- On-chip oscillator that accepts crystal or external clock drive.
- Full-duplex UART and two programmable 8-bit counter/timers, each with a 6-bit programmable prescaler.
- Register Pointer so that short, fast instructions can access any one of the nine working-register groups.
- Single +5V power supply—all I/O pins TTL compatible.
- **Z8681/82 available in 8 MHz. Z8681 also available in 12 and 16 MHz.**

GENERAL DESCRIPTION

The Z8681 and Z8682 are ROMless versions of the Z8 single-chip microcomputer. The Z8682 is usually more cost effective. These products differ only slightly and can be used interchangeably with proper system design to provide maximum flexibility in meeting price and delivery needs.

The Z8681/82 offers all the outstanding features of the Z8 family architecture except an on-chip program ROM. Use of external memory rather than a preprogrammed ROM enables this Z8 microcomputer to be used in low volume applications or where code flexibility is required.

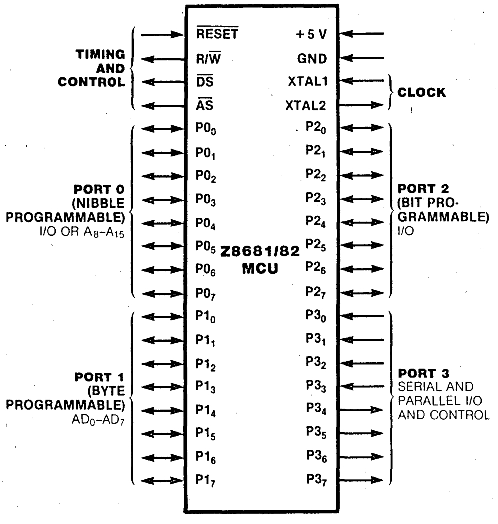


Figure 1. Pin Functions

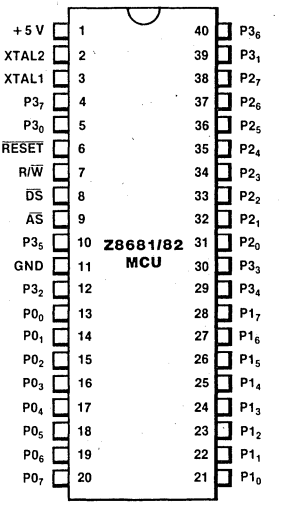


Figure 2a. 40-pin Dual-In-Line Package (DIP), Pin Assignments

The Z8681/82 can provide up to 16 output address lines, thus permitting an address space of up to 64K bytes of data or program memory. Eight address outputs (AD$_0$-AD$_7$) are provided by a multiplexed, 8-bit, Address/Data bus. The remaining 8 bits can be provided by the software configuration of Port 0 to output address bits A$_8$-A$_{15}$.

Available address space can be doubled (up to 128K bytes for the Z8681 and 124K bytes for the Z8682) by programming bit 4 of Port 3 (P3$_4$) to act as a data memory select output ($\overline{\text{DM}}$). The two states of $\overline{\text{DM}}$ together with the 16 address outputs can define separate data and memory address spaces of up to 64K/62Kbytes each.

There are 143 bytes of RAM located on-chip and organized as a register file of 124 general-purpose registers, 16 control and status registers, and three I/O port registers. This register file can be divided into nine groups of 16 working registers each. Configuring the register file in this manner allows the use of short format instructions; in addition, any of the individual registers can be accessed directly.

The pin functions and the pin assignments of the Z8681/82 40– and 44–pin packages are illustrated in Figures 1 and 2, respectively.

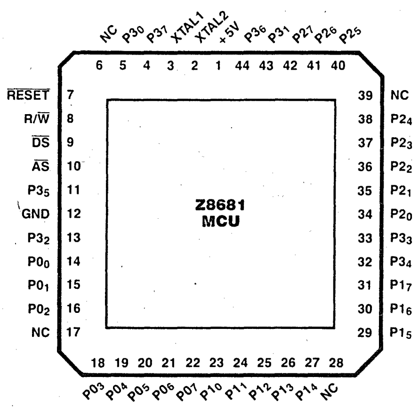


Figure 2b. 44-pin Chip Carrier, Pin Assignments

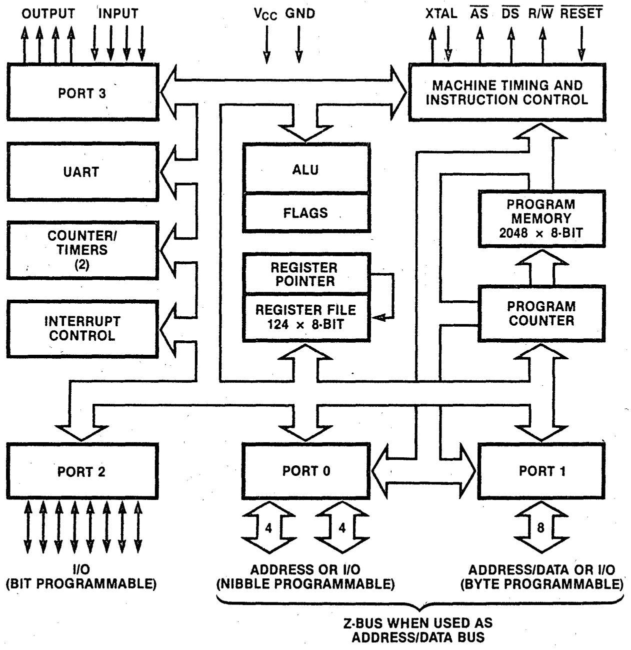

OUTPUT
INPUT
V$_{CC}$
GND
XTAL
$\overline{\text{AS}}$
$\overline{\text{DS}}$
R/$\overline{\text{W}}$
$\overline{\text{RESET}}$
PORT 3
MACHINE TIMING AND INSTRUCTION CONTROL
UART
ALU
FLAGS
COUNTER/ TIMERS (2)
PROGRAM MEMORY 2048 × 8-BIT
REGISTER POINTER
INTERRUPT CONTROL
REGISTER FILE 124 × 8-BIT
PROGRAM COUNTER
PORT 2
PORT 0
PORT 1
4
4
8
I/O (BIT PROGRAMMABLE)
ADDRESS OR I/O (NIBBLE PROGRAMMABLE)
ADDRESS/DATA OR I/O (BYTE PROGRAMMABLE)
Z-BUS WHEN USED AS ADDRESS/DATA BUS

Figure 3. Functional Block Diagram

ARCHITECTURE

Z8681/82 architecture is characterized by a flexible I/O scheme, an efficient register and address space structure and a number of ancillary features that are helpful in many applications.

Microcomputer applications demand powerful I/O capabilities. The Z8681/82 fulfills this with 24 pins available for input and output. These lines are grouped into three ports of eight lines each and are configurable under software control to provide timing, status signals, serial or parallel I/O with or without handshake, and an Address bus for interfacing external memory.

Three basic address spaces are available: program memory, data memory and the register file (internal). The 143-byte random-access register file is composed of 124 general-purpose registers, three I/O port registers, and 16 control and status registers.

To unburden the program from coping with real-time problems such as serial data communication and counting/timing, an asynchronous receiver/transmitter (UART) and two counter/timers with a large number of user-selectable modes are offered on-chip. Hardware support for the UART is minimized because one of the on-chip timers supplies the bit rate. Figure 3 shows the Z8681/82 block diagram.

PIN DESCRIPTION

$\overline{\text{AS}}$. *Address Strobe* (output, active Low). Address Strobe is pulsed once at the beginning of each machine cycle. Addresses output via Port 1 for all external program or data memory transfers are valid at the trailing edge of $\overline{\text{AS}}$.

$\overline{\text{DS}}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each external memory transfer.

$P0_0$-$P0_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. *I/O Port Lines* (input/outputs, TTL-compatible). These 24 lines are divided into three 8-bit I/O ports that can be configured under program control for I/O or external memory interface (Figure 3).

$P1_0$-$P1_7$. *Address/Data Port* (bidirectional). Multiplexed address (A_0-A_7) and data (D_0-D_7) lines used to interface with program and data memory.

$\overline{\text{RESET}}$. *Reset* (input, active Low). $\overline{\text{RESET}}$ initializes the Z8681/82. After RESET the Z8681 is in the extended memory mode. When $\overline{\text{RESET}}$ is deactivated, program execution begins from program location $000C_H$ for the Z8681 and 0812_H for the Z8682.

R/$\overline{\text{W}}$. *Read/Write* (output). R/$\overline{\text{W}}$ is Low when the Z8681/82 is writing to external program or data memory.

XTAL1, XTAL2. *Crystal 1, Crystal 2* (time-base input and output). These pins connect a parallel-resonant crystal to the on-chip clock oscillator and buffer.

SUMMARY OF Z8681 AND Z8682 DIFFERENCES

Feature	Z8681	Z8682
Address of first instruction executed after Reset	12	2066
Addressable memory space	0–64K	2K–64K
Address of interrupt vectors	0–11	2048–2065
Reset input high voltage	TTL levels *	7.35–8.0V
Port 0 configuration after Reset	Input, float after reset. Can be programmed as Address bits.	Output, configured as Address bit A_8–A_{15}.
External memory timing start-up configurations	Extended Timing	Normal Timing
Interrupt vectors	2 byte vectors point directly to service routines.	2 byte vectors in internal ROM point to 3 byte Jump instructions, which point to service routines.
Interrupt response time	26 clocks	36 clocks

*8.0V V_{IN} max.

ADDRESS SPACES

Program Memory*. The Z8681/82 addresses 64K/62K bytes of external program memory space (Figure 4).

For the Z8681, the first 12 bytes of program memory are reserved for the interrupt vectors. These locations contain six 16-bit vectors that correspond to the six available interrupts. Program execution begins at location $000C_H$ after a reset.

The Z8682 has six 24-bit interrupt vectors beginning at address 0800_H. The vectors consist of Jump Absolute instructions. After a reset, program execution begins at location 0812_H for the Z8682.

Data Memory*. The Z8681/82 can address 64K/62K bytes of external data memory. External data memory may be included with or separated from the external program memory space. $\overline{DM}$, an optional I/O function that can be programmed to appear on pin $P3_4$, is used to distinguish between data and program memory space.

Register File. The 143-byte register file includes three I/O port registers (R0, R2, R3), 124 general-purpose registers (R4-R127) and 16 control and status registers (R240-R255). These registers are assigned the address locations shown in Figure 5.

Z8681/82 instructions can access registers directly or indirectly with an 8-bit address field. This also allows short 4-bit register addressing using the Register Pointer (one of the control registers). In the 4-bit mode, the register file is divided into nine working-register groups, each occupying 16 contiguous locations (Figure 5). The Register Pointer addresses the starting location of the active working-register group (Figure 6).

Stacks. Either the internal register file or the external data memory can be used for the stack. A 16-bit Stack Pointer (R254 and R255) is used for the external stack, which can reside anywhere in data memory. An 8-bit Stack Pointer (R255) is used for the internal stack that resides within the 124 general-purpose registers (R4-R127).

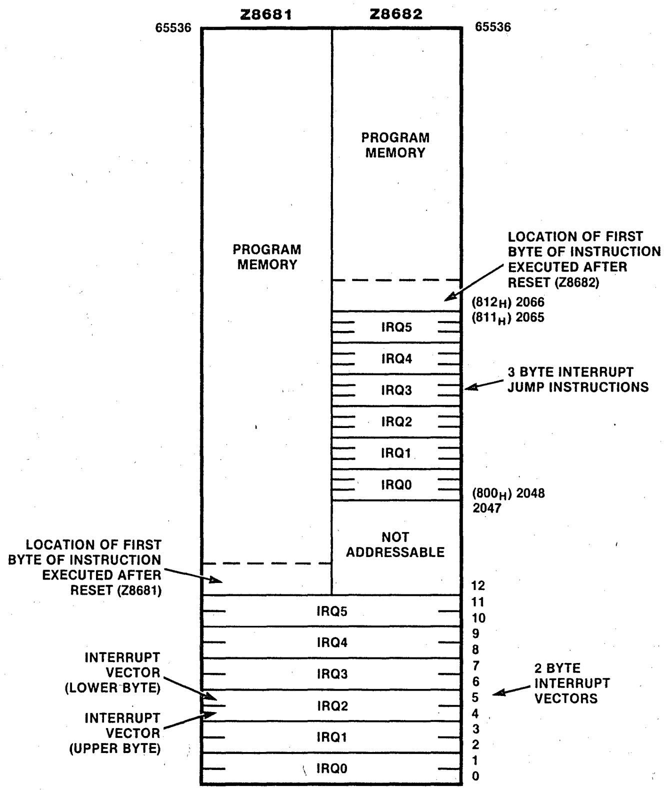

Z8681
Z8682
65536
65536
PROGRAM MEMORY
PROGRAM MEMORY
LOCATION OF FIRST BYTE OF INSTRUCTION EXECUTED AFTER RESET (Z8682)
(812H) 2066
(811H) 2065
IRQ5
IRQ4
IRQ3
IRQ2
IRQ1
IRQ0
3 BYTE INTERRUPT JUMP INSTRUCTIONS
(800H) 2048
2047
NOT ADDRESSABLE
LOCATION OF FIRST BYTE OF INSTRUCTION EXECUTED AFTER RESET (Z8681)
12
11
10
9
8
7
6
5
4
3
2
1
0
IRQ5
IRQ4
IRQ3
IRQ2
IRQ1
IRQ0
INTERRUPT VECTOR (LOWER BYTE)
INTERRUPT VECTOR (UPPER BYTE)
2 BYTE INTERRUPT VECTORS

Figure 4. Z8681/82 Program Memory Map

*This feature differs in the Z8681 and Z8682.

DEC		HEX	IDENTIFIERS
255	STACK POINTER (BITS 7-0)	FF	SPL
254	STACK POINTER (BITS 15-8)	FE	SPH
253	REGISTER POINTER	FD	RP
252	PROGRAM CONTROL FLAGS	FC	FLAGS
251	INTERRUPT MASK REGISTER	FB	IMR
250	INTERRUPT REQUEST REGISTER	FA	IRQ
249	INTERRUPT PRIORITY REGISTER	F9	IPR
248	PORTS 0-1 MODE	F8	P01M
247	PORT 3 MODE	F7	P3M
246	PORT 2 MODE	F6	P2M
245	T0 PRESCALER	F5	PRE0
244	TIMER/COUNTER 0	F4	T0
243	T1 PRESCALER	F3	PRE1
242	TIMER/COUNTER 1	F2	T1
241	TIMER MODE	F1	TMR
240	SERIAL I/O	F0	SIO
	NOT IMPLEMENTED		
127	GENERAL-PURPOSE REGISTERS	7F	
4		04	
3	PORT 3	03	P3
2	PORT 2	02	P2
1	PORT 1	01	P1
0	PORT 0	00	P0

Figure 5. The Register File

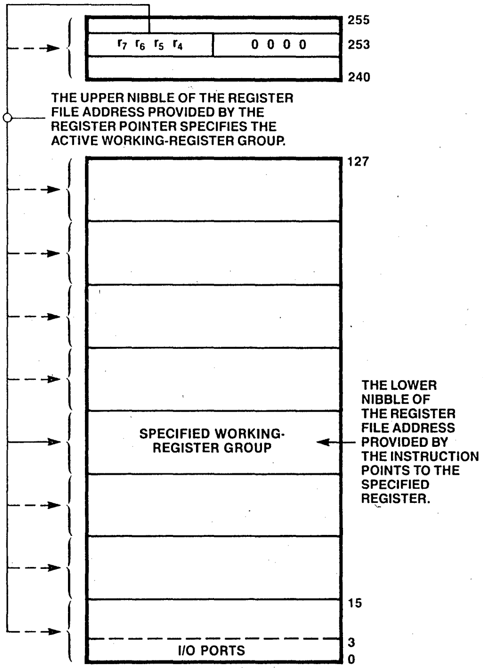

255
r_7 r_6 r_5 r_4
0 0 0 0
253
240
THE UPPER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE REGISTER POINTER SPECIFIES THE ACTIVE WORKING-REGISTER GROUP.
127
SPECIFIED WORKING-REGISTER GROUP
THE LOWER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE INSTRUCTION POINTS TO THE SPECIFIED REGISTER.
15
I/O PORTS
3
0

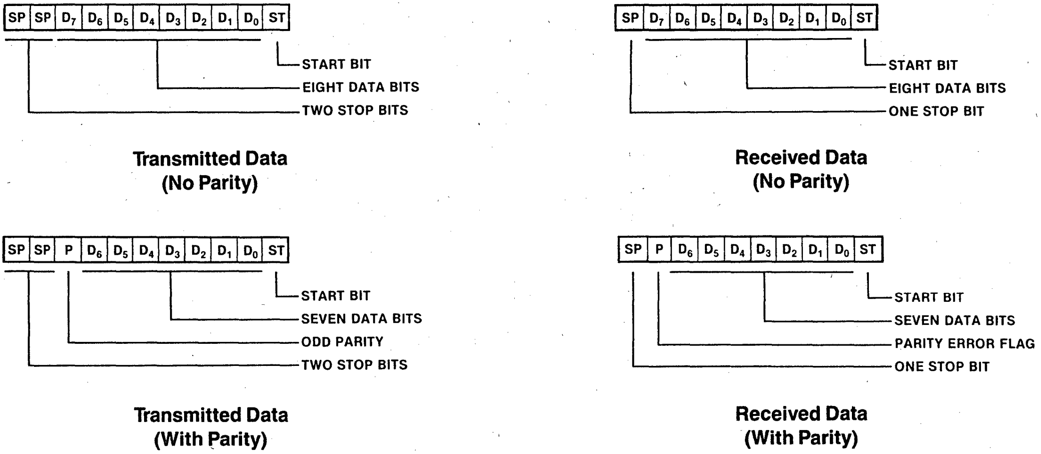
Figure 6. The Register Pointer

SERIAL INPUT/OUTPUT

Port 3 lines $P3_0$ and $P3_7$ can be programmed as serial I/O lines for full-duplex serial asynchronous receiver/transmitter operation. The bit rate is controlled by Counter/Timer 0.

The Z8681/82 automatically adds a start bit and two stop bits to transmitted data (Figure 7). Odd parity is also available as an option. Eight data bits are always transmitted, regardless of parity selection. If parity is enabled, the eighth data bit is used as the odd parity bit. An interrupt request (IRQ4) is generated on all transmitted characters.

Received data must have a start bit, eight data bits, and at least one stop bit. If parity is on, bit 7 of the received data is replaced by a parity error flag. Received characters generate the IRQ3 interrupt request.


SP SP D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 ST
START BIT
EIGHT DATA BITS
TWO STOP BITS
SP D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 ST
START BIT
EIGHT DATA BITS
ONE STOP BIT
SP SP P D_6 D_5 D_4 D_3 D_2 D_1 D_0 ST
START BIT
SEVEN DATA BITS
ODD PARITY
TWO STOP BITS
SP P D_6 D_5 D_4 D_3 D_2 D_1 D_0 ST
START BIT
SEVEN DATA BITS
PARITY ERROR FLAG
ONE STOP BIT

Transmitted Data (No Parity)

Received Data (No Parity)

Transmitted Data (With Parity)

Received Data (With Parity)

Figure 7. Serial Data Formats

COUNTER/TIMERS

The Z8681/82 contains two 8-bit programmable counter/timers (T_0 and T_1), each driven by its own 6-bit programmable prescaler. The T_1 prescaler can be driven by internal or external clock sources; however, the T_0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrements the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request—IRQ4 (T_0) or IRQ5 (T_1)—is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single-pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read any time without disturbing their value or count mode.

The clock source for T_1 is user-definable; it can be either the internal microprocessor clock divided by four, or an external signal input via Port 3. The Timer Mode register configures the external timer input as an external clock, a trigger input that can be retriggerable or nonretriggerable, or as a gate input for the internal clock. The counter/timers can be programmably cascaded by connecting the T_0 output to the input of T_1. Port 3 line $P3_6$ also serves as a timer output (T_{OUT}) through which T_0, T_1 or the internal clock can be output.

I/O PORTS

The Z8681/82 has 24 lines available for input and output. These lines are grouped into three ports of eight lines each and are configurable as input, output or address. Under software control, the ports can be programmed to provide address outputs, timing, status signals, serial I/O, and parallel I/O with or without handshake. All ports have active pull-ups and pull-downs compatible with TTL loads.

Port 1 is a dedicated Z-BUS compatible memory interface. The operations of Port 1 are supported by the Address Strobe ($\overline{AS}$) and Data Strobe ($\overline{DS}$) lines, and by the Read/Write (R/$\overline{W}$) and Data Memory ($\overline{DM}$) control lines. The low-order program and data memory addresses (A_0-A_7) are output through Port 1 (Figure 8) and are multiplexed with data in/out (D_0-D_7). Instruction fetch and data memory read/write operations are done through this port.

Port 1 cannot be used as a register nor can a handshake mode be used with this port.

Both the Z8681 and Z8682 wake up with the 8 bits of Port 1 configured as address outputs for external memory. If more than eight address lines are required with the Z8681, additional lines can be obtained by programming Port 0 bits as address bits. The least-significant four bits of Port 0 can be configured to supply address bits A_8-A_{11} for 4K byte addressing or both nibbles of Port 0 can be configured to supply address bits A_8-A_{15} for 64K byte addressing.

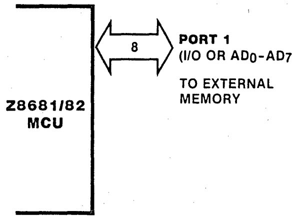


Figure 8. Port 1

Port 0* can be programmed as a nibble I/O port, or as an address port for interfacing external memory (Figure 9). When used as an I/O port, Port 0 can be placed under handshake control. In this configuration, Port 3 lines $P3_2$ and $P3_5$ are used as the handshake controls DAV_0 and RDY_0. Handshake signal assignment is dictated by the I/O direction of the upper nibble $P0_4$-$P0_7$.

For external memory references, Port 0 can provide address bits A_8-A_{11} (lower nibble) or A_8-A_{15} (lower and upper nibbles) depending on the required address space.. If the address range requires 12 bits or less, the upper nibble of Port 0 can be programmed independently as I/O while the lower nibble is used for addressing.

In the Z8681 *, Port 0 lines float after reset; their logic state is unknown until the execution of an initialization routine that configures Port 0. Such an initialization routine must reside within the first 256 bytes of executable code and must be physically mapped into memory by forcing the Port 0 address lines to a known state (Figure 10). The proper port initialization sequence is:

1. Write initial address (A_8-A_{15}) of initialization routine to Port 0 address lines.
2. Configure Port 0 Mode register to output A_8-A_{15} (or A_8-A_{11}).

To permit the use of slow memory, an automatic wait mode of two oscillator clock cycles is configured for the bus timing of the Z8681 after each reset. The initialization routine could include reconfiguration to eliminate this extended timing mode.

*This feature differs in the Z8681 and Z8682.

The following example illustrates the manner in which an initialization routine can be mapped in a Z8681 system with 4K of memory.

Example. In Figure 10, the initialization routine is mapped to the first 256 bytes of program memory. Pull-down resistors maintain the address lines at a logic 0 level when these lines are floating. The leakage current caused by fanout must be taken into consideration when selecting the value of the pulldown resistors. The resistor value must be large enough to allow the Port 0 output driver to pull the line to a logic 1. Generally, pulldown resistors are incompatible with TTL loads. If Port 0 drives into TTL input loads (I_{LOW} = 1.6 mA) the external resistors should be tied to V_{CC} and the initialization routine put in address space $FF00_H$–$FFFF_H$.

*In the Z8682**, Port 0 lines are configured as address lines A_8-A_{15} after a Reset. If one or both nibbles are needed for I/O operation, they must be configured by writing to the Port 0 Mode register. The Z8682 is in the fast memory timing mode after Reset, so the initialization routine must be in fast memory.

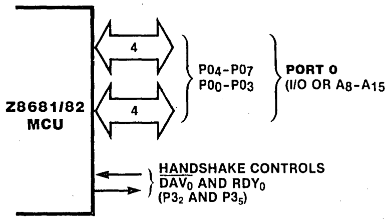


Figure 9. Port 0

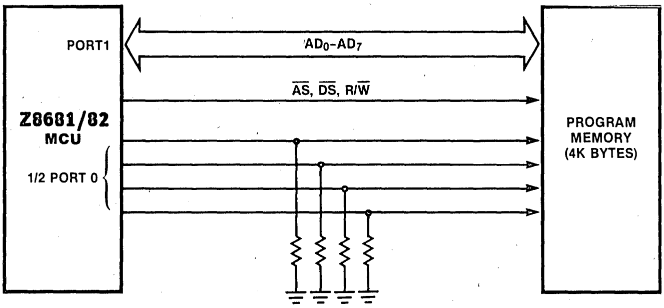


Figure 10. Port 0 Address Lines Tied to Logic 0

Port 2 bits can be programmed independently as input or output (Figure 11). This port is always available for I/O operations. In addition, Port 2 can be configured to provide open-drain outputs.

Like Port 0, Port 2 may also be placed under handshake control. In this configuration, Port 3 lines $P3_1$ and $P3_6$ are used as the handshake controls lines $\overline{DAV}_2$ and RDY_2. The handshake signal assignment for Port 3 lines $P3_1$ and $P3_6$ is dictated by the direction (input or output) assigned to bit 7 of Port 2.

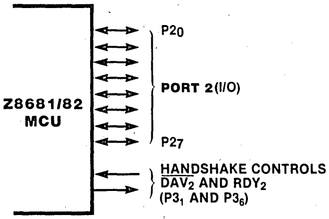


Figure 11. Port 2

Port 3 lines can be configured as I/O or control lines (Figure 12). In either case, the direction of the eight lines is fixed as four input ($P3_0$-$P3_3$) and four output ($P3_4$-$P3_7$). For serial I/O, lines $P3_0$ and $P3_7$ are programmed as serial in and serial out, respectively.

Port 3 can also provide the following control functions: handshake for Ports 0 and 2 ($\overline{DAV}$ and RDY); four external interrupt request signals (IRQ0-IRQ3); timer input and output signals (T_{IN} and T_{OUT}) and Data Memory Select ($\overline{DM}$).

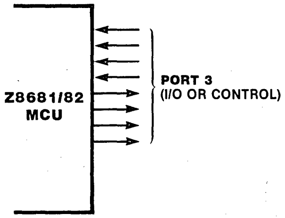


Figure 12. Port 3

*This feature differs in the Z8681 and Z8682.

INTERRUPTS*

The Z8681/82 allows six different interrupts from eight sources: the four Port 3 lines $P3_0$-$P3_3$, Serial In, Serial Out, and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask register globally or individually enables or disables the six interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All Z8681 and Z8682 interrupts are vectored through locations in program memory. When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the Program Counter and status flags, and accesses the program memory vector location reserved for that interrupt. In the Z8681, this memory location and the next byte contain the 16-bit address of the interrupt service routine for that particular interrupt request. The Z8681 takes 26 system clock cycles to enter an interrupt subroutine.

The Z8682 has a small internal ROM that contains six 2-byte interrupt vectors pointing to addresses 2048-2065, where 3-byte jump absolute instructions are located (Figure 4 and Table 1). These jump instructions each contain a 1-byte opcode and a 2-byte starting address for the interrupt service routine. The Z8682 takes 36 system clock cycles to enter an interrupt subroutine.

Table 1. Z8682 Interrupt Processing

Hex Address	Contains Jump Instruction and Subroutine Address For
800-802	IRQ0
803-805	IRQ1
806-808	IRQ2
809-80B	IRQ3
80C-80E	IRQ4
80F-811	IRQ5

Polled interrupt systems are also supported. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the Interrupt Request register polled to determine which of the interrupt requests needs service.

CLOCK

The on-chip oscillator has a high-gain, parallel-resonant amplifier for connection to a crystal or to any suitable external clock source (XTAL1 = Input, XTAL2 = Output).

The crystal source is connected across XTAL1 and XTAL2, using the recommended capacitance (C_L = 15 pf maximum) from each pin to ground. The specifications for the crystal are as follows:

- AT cut, parallel-resonant
- Fundamental type
- Series resistance, $R_S \leq 100\Omega$
- For Z8682, 8 MHz maximum
- **For Z8681–12, 16 MHz maximum**

Z8681/Z8682 INTERCHANGEABILITY

Although the Z8681 and Z8682 have minor differences, a system can be designed for compatibility with both ROMless versions. To achieve interchangeability, the design must take into account the special requirements of each device in the external interface, initialization, and memory mapping.

External Interface. The Z8682 requires a 7.5V positive logic level on the $\overline{\text{RESET}}$ pin for at least 6 clock periods immediately following reset, as shown in Figure 13. The Z8681 requires a 3.8V or higher positive logic level, but is compatible with the Z8682 $\overline{\text{RESET}}$ waveform. Figure 14 shows a simple circuit for generating the 7.5V level.

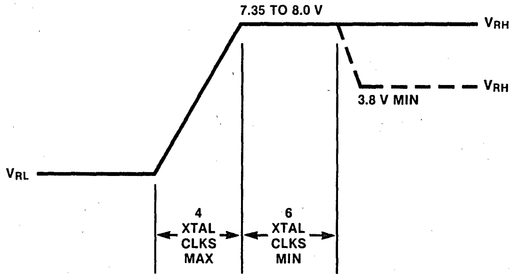


Figure 13. Z8682 RESET Pin Input Waveform

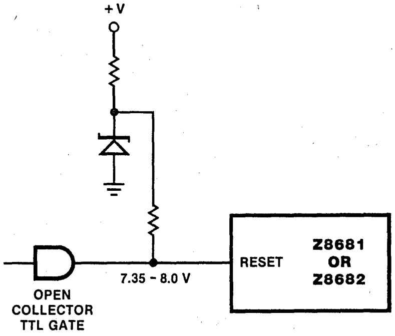


Figure 14. RESET Circuit

*This feature differs in the Z8681 and Z8682.

Initialization. The Z8681 wakes up after reset with Port 0 configured as an input, which means Port 0 lines are floating in a high-impedance state. Because of this pullup or pulldown, resistors must be attached to Port 0 lines to force them to a valid logic level until Port 0 is configured as an address port.

Port 0 initialization is discussed in the section on ports. An example of an initialization routine for Z8681/Z8682 compatibility is shown in Table 2. Only the Z8681 need execute this program.

Table 2. Initialization Routine

Address	Opcodes	Instruction	Comments
000C	E6 00 00	LD PO #%00	Set A_8-A_{15} to 0.
000F	E6 F8 96	LD P01M #%96	Configure Port 0 as A_8-A_{15}. Eliminate extended memory timing.
0012	8D 08 12	JP START ADDRESS	Execute application program.

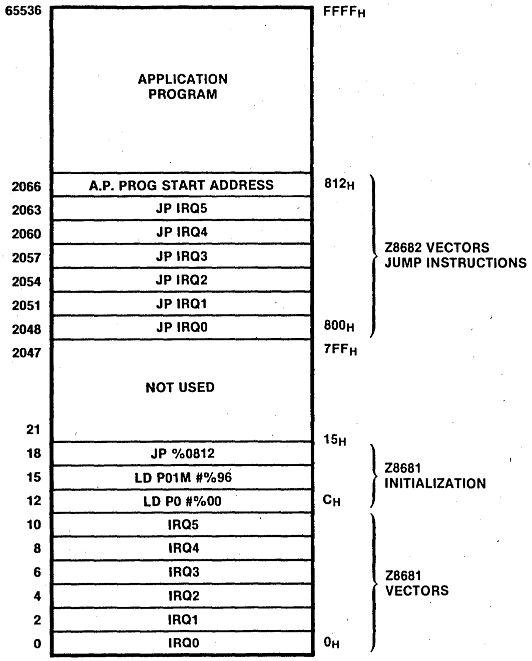
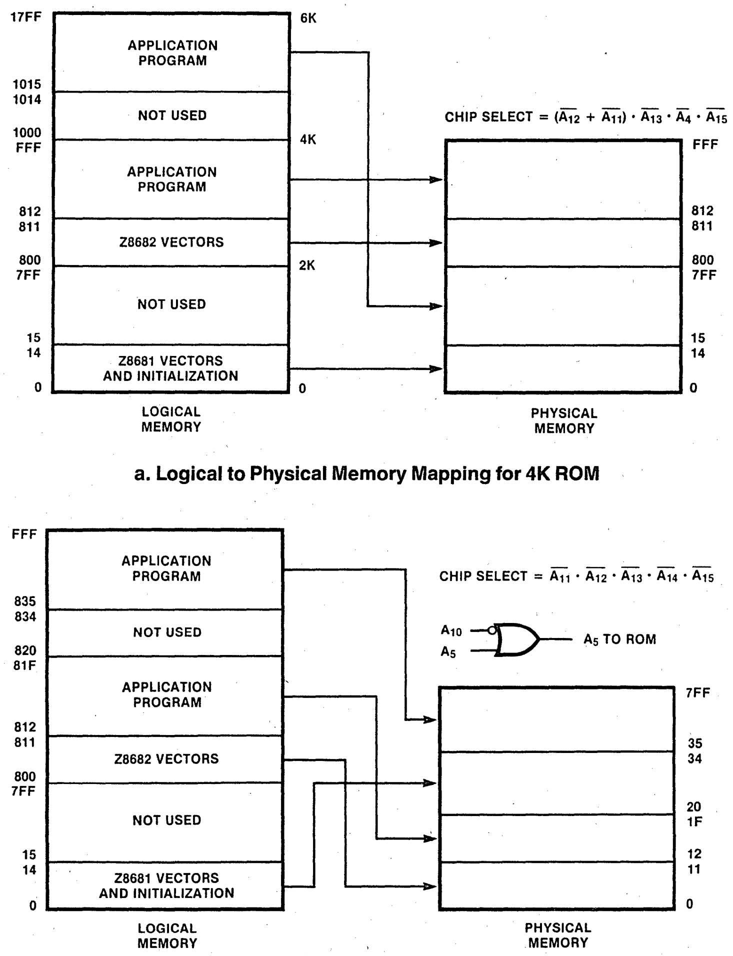


Figure 15. Z8681/82 Logical Program Memory Mapping

Memory Mapping. The Z8681 and Z8682 lower memory boundaries are located at 0 and 2048, respectively. A single program ROM can be used with either product if the logical program memory map shown in Figure 15 is followed. The Z8681 vectors and initialization routine must be starting at address 0 and the Z8682 3-byte vectors (jump instructions) must be at address 2048 and higher. Addresses in the range 21-2047 are not used. Figure 16 shows practical schemes for implementing this memory map using 4K and 2K ROMs.



a. Logical to Physical Memory Mapping for 4K ROM

b. Logical to Physical Memory Mapping for 2K ROM

Figure 16. Practical Schemes for Implementing Z8681 and Z8682 Compatible Memory Map

INSTRUCTION SET NOTATION

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "←". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*****	Set or cleared according to operation
—	Unaffected
X	Undefined

CONDITION CODES

Value	Mnemonic	Meaning	Flags Set
1000		Always true	—
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	—

INSTRUCTION FORMATS

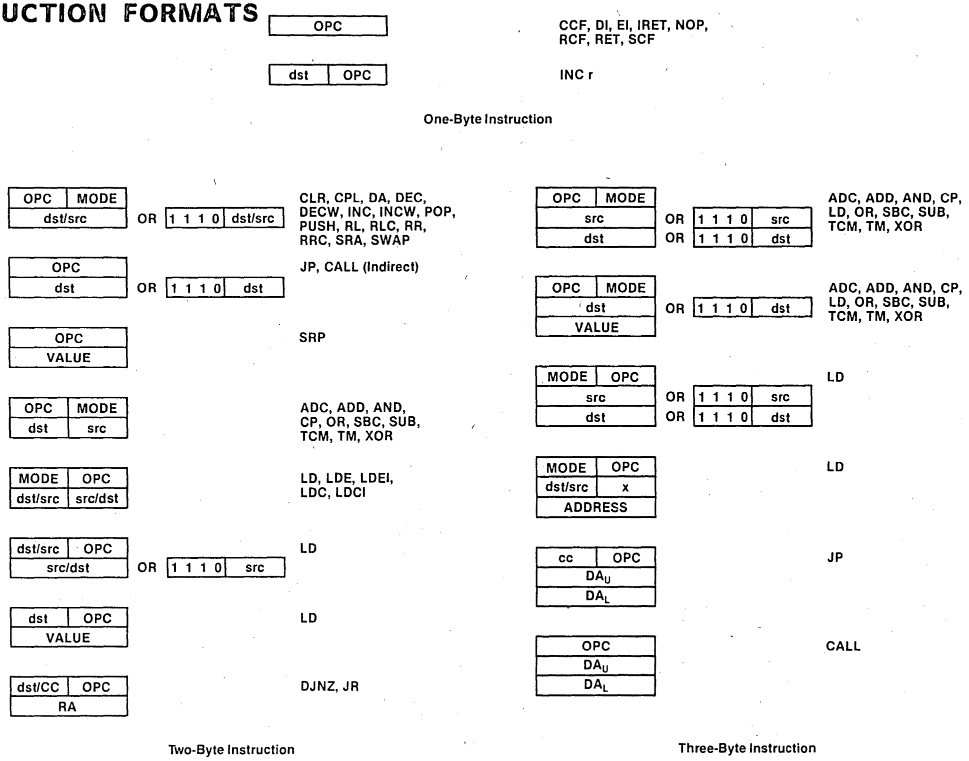
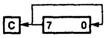


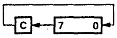
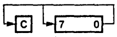
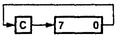
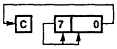
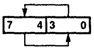
Figure 17. Instruction Formats

INSTRUCTION SUMMARY

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
ADC dst,src dst ← dst + src + C	(Note 1)		1□	*	*	*	*	0	*
ADD dst,src dst ← dst + src	(Note 1)		0□	*	*	*	*	0	*
AND dst,src dst ← dst AND src	(Note 1)		5□	—	*	*	0	—	—
CALL dst SP ← SP − 2 @SP ← PC; PC ← dst	DA IRR		D6 D4	—	—	—	—	—	—
CCF C ← NOT C			EF	*	—	—	—	—	—
CLR dst dst ← 0	R IR		B0 B1	—	—	—	—	—	—
COM dst dst ← NOT dst	R IR		60 61	—	*	*	0	—	—
CP dst,src dst − src	(Note 1)		A□	*	*	*	*	—	—
DA dst dst ← DA dst	R IR		40 41	*	*	*	X	—	—
DEC dst dst ← dst − 1	R IR		00 01	—	*	*	*	—	—
DECW dst dst ← dst − 1	RR IR		80 81	—	*	*	*	—	—
DI IMR (7) ← 0			8F	—	—	—	—	—	—
DJNZ r,dst r ← r − 1 if r ≠ 0 PC ← PC + dst Range: +127, −128	RA		rA r = 0 − F	—	—	—	—	—	—
EI IMR (7) ← 1			9F	—	—	—	—	—	—
INC dst dst ← dst + 1	r R IR		rE r = 0 − F 20 21	—	*	*	*	—	—
INCW dst dst ← dst + 1	RR IR		A0 A1	—	*	*	*	—	—

INSTRUCTION SUMMARY (Continued)

Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
IRET			BF	*	*	*	*	*	*
FLAGS ← @SP; SP ← SP + 1									
PC ← @SP; SP ← SP + 2; IMR (7) ← 1									
JP cc,dst	DA		cD	—	—	—	—	—	—
if cc is true			c = 0 – F						
PC ← dst	IRR		30						
JR cc,dst	RA		cB	—	—	—	—	—	—
if cc is true,			c = 0 – F						
PC ← PC + dst									
Range: +127, –128									
LD dst,src	r	Im	rC	—	—	—	—	—	—
dst ← src	r	R	r8						
	R	r	r9						
			r = 0 – F						
	r	X	C7						
	X	r	D7						
	r	Ir	E3						
	Ir	r	F3						
	R	R	E4						
	R	IR	E5						
	R	IM	E6						
	IR	IM	E7						
	IR	R	F5						
LDC dst,src	r	Irr	C2	—	—	—	—	—	—
dst ← src	Irr	r	D2						
LDCI dst,src	Ir	Irr	C3	—	—	—	—	—	—
dst ← src	Irr	Ir	D3						
r ← r + 1; rr ← rr + 1									
LDE dst,src	r	Irr	82	—	—	—	—	—	—
dst ← src	Irr	r	92						
LDEI dst,src	Ir	Irr	83	—	—	—	—	—	—
dst ← src	Irr	Ir	93						
r ← r + 1; rr ← rr + 1									
NOP			FF	—	—	—	—	—	—
OR dst,src	(Note 1)		4□	—	*	*	0	—	—
dst ← dst OR src									
POP dst	R		50	—	—	—	—	—	—
dst ← @SP;	IR		51						
SP ← SP + 1									
PUSH src		R	70	—	—	—	—	—	—
SP ← SP – 1; @SP ← src		IR	71						
RCF			CF	0	—	—	—	—	—
C ← 0									
RET			AF	—	—	—	—	—	—
PC ← @SP; SP ← SP + 2									
RL dst 	R		90	*	*	*	*	—	—
	IR		91						

Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
RLC dst 	R		10	*	*	*	*	—	—
	IR		11						
RR dst 	R		E0	*	*	*	*	—	—
	IR		E1						
RRC dst 	R		C0	*	*	*	*	—	—
	IR		C1						
SBC dst,src	(Note 1)		3□	*	*	*	*	1	*
dst ← dst ← src ← C									
SCF			DF	1	—	—	—	—	—
C ← 1									
SRA dst 	R		D0	*	*	*	0	—	—
	IR		D1						
SRP src		Im	31	—	—	—	—	—	—
RP ← src									
SUB dst,src	(Note 1)		2□	*	*	*	*	1	*
dst ← dst ← src									
SWAP dst 	R		F0	X	*	*	X	—	—
	IR		F1						
TCM dst,src	(Note 1)		6□	—	*	*	0	—	—
(NOT dst) AND src									
TM dst,src	(Note 1)		7□	—	*	*	0	—	—
dst AND src									
XOR dst,src	(Note 1)		B□	—	*	*	0	—	—
dst ← dst XOR src									

NOTE: These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the left of the applicable addressing mode pair.

For example, the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

Addr Mode		Lower
dst	src	Opcode Nibble
r	r	$\boxed{2}$
r	Ir	$\boxed{3}$
R	R	$\boxed{4}$
R	IR	$\boxed{5}$
R	IM	$\boxed{6}$
IR	IM	$\boxed{7}$

REGISTERS

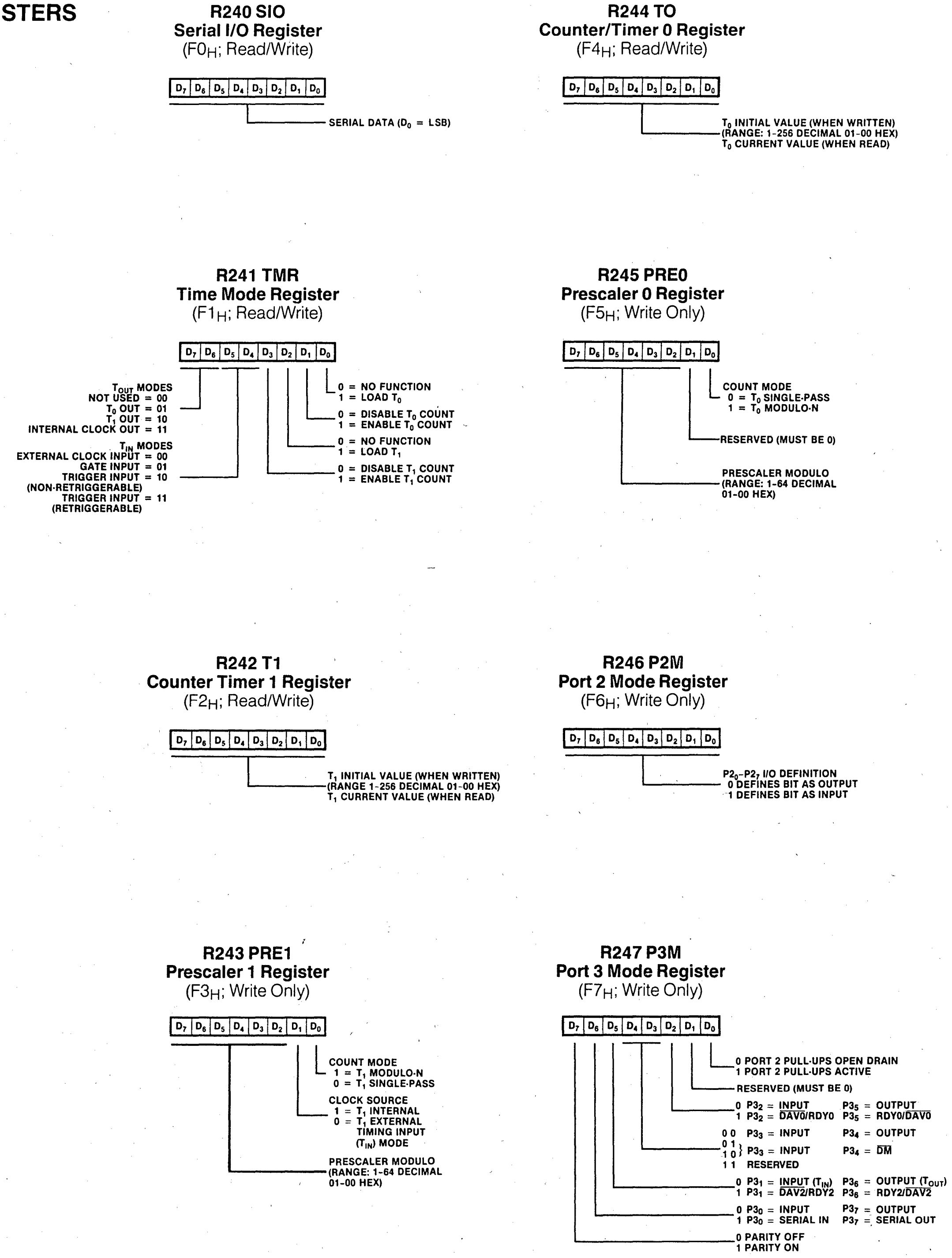
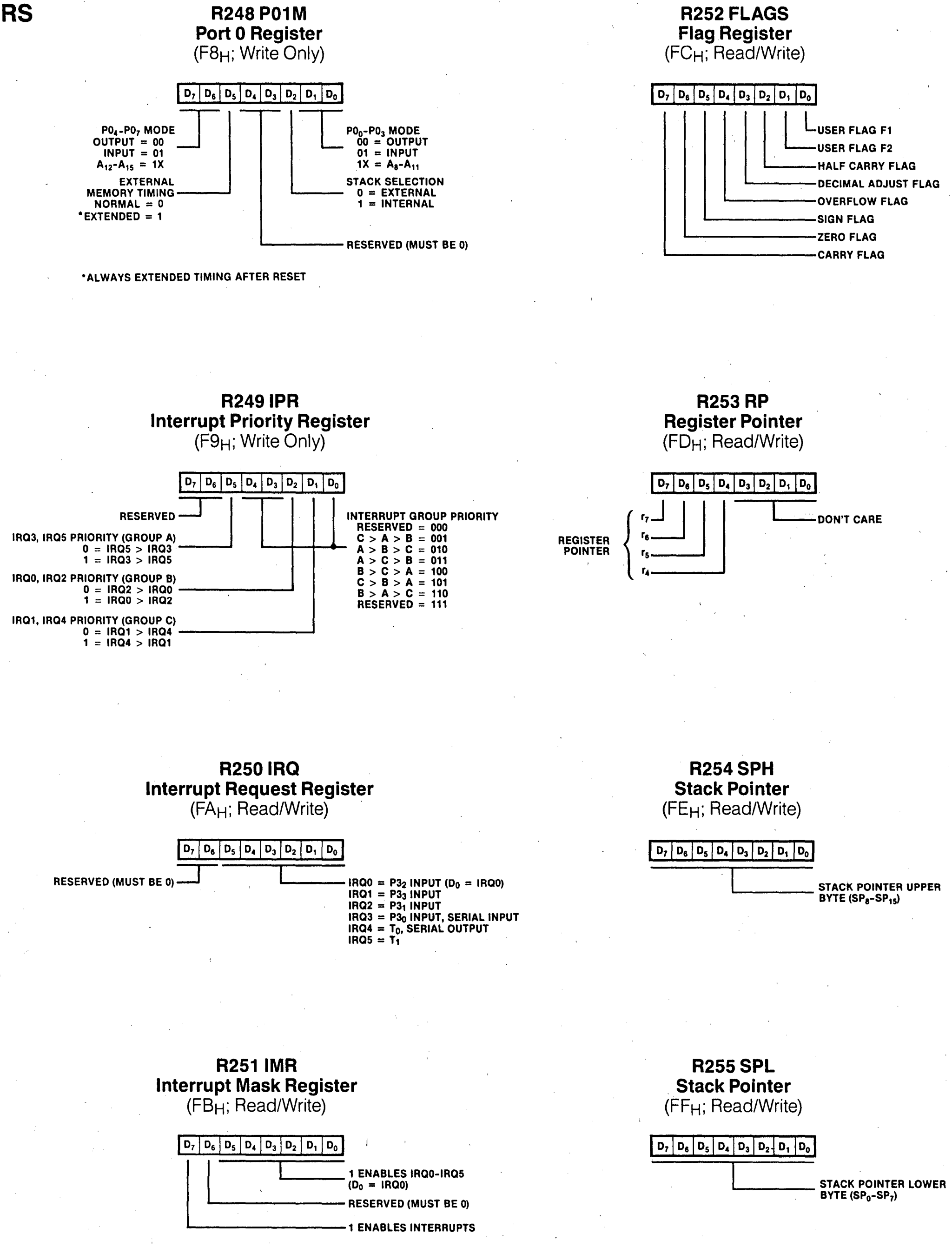


Figure 18. Control Registers

REGISTERS
(Continued)


R248 P01M
Port 0 Register
(F8H; Write Only)

R252 FLAGS
Flag Register
(FCH; Read/Write)

R249 IPR
Interrupt Priority Register
(F9H; Write Only)

R253 RP
Register Pointer
(FDH; Read/Write)

R250 IRQ
Interrupt Request Register
(FAH; Read/Write)

R254 SPH
Stack Pointer
(FEH; Read/Write)

R251 IMR
Interrupt Mask Register
(FBH; Read/Write)

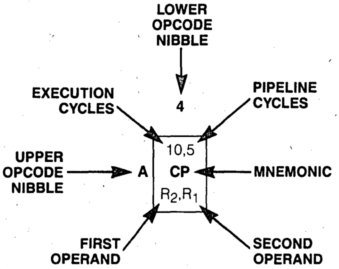
R255 SPL
Stack Pointer
(FFH; Read/Write)

Figure 18. Control Registers (Continued)

Z8681/82 OPCODE MAP

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6,5 **DEC** R_1	6,5 **DEC** IR_1	6,5 **ADD** r_1,r_2	6,5 **ADD** r_1,Ir_2	10,5 **ADD** R_2,R_1	10,5 **ADD** IR_2,R_1	10,5 **ADD** R_1,IM	10,5 **ADD** IR_1,IM	6,5 **LD** r_1,R_2	6,5 **LD** r_2,R_1	12/10,5 **DJNZ** r_1,RA	12/10,0 **JR** cc,RA	6,5 **LD** r_1,IM	12/10,0 **JP** cc,DA	6,5 **INC** r1	
1	6,5 **RLC** R_1	6,5 **RLC** IR_1	6,5 **ADC** r_1,r_2	6,5 **ADC** r_1,Ir_2	10,5 **ADC** R_2,R_1	10,5 **ADC** IR_2,R_1	10,5 **ADC** R_1,IM	10,5 **ADC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
2	6,5 **INC** R_1	6,5 **INC** IR_1	6,5 **SUB** r_1,r_2	6,5 **SUB** r_1,Ir_2	10,5 **SUB** R_2,R_1	10,5 **SUB** IR_2,R_1	10,5 **SUB** R_1,IM	10,5 **SUB** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
3	8,0 **JP** IRR_1	6,1 **SRP** IM	6,5 **SBC** r_1,r_2	6,5 **SBC** r_1,Ir_2	10,5 **SBC** R_2,R_1	10,5 **SBC** IR_2,R_1	10,5 **SBC** R_1,IM	10,5 **SBC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
4	8,5 **DA** R_1	8,5 **DA** IR_1	6,5 **OR** r_1,r_2	6,5 **OR** r_1,Ir_2	10,5 **OR** R_2,R_1	10,5 **OR** IR_2,R_1	10,5 **OR** R_1,IM	10,5 **OR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
5	10,5 **POP** R_1	10,5 **POP** IR_1	6,5 **AND** r_1,r_2	6,5 **AND** r_1,Ir_2	10,5 **AND** R_2,R_1	10,5 **AND** IR_2,R_1	10,5 **AND** R_1,IM	10,5 **AND** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
6	6,5 **COM** R_1	6,5 **COM** IR_1	6,5 **TCM** r_1,r_2	6,5 **TCM** r_1,Ir_2	10,5 **TCM** R_2,R_1	10,5 **TCM** IR_2,R_1	10,5 **TCM** R_1,IM	10,5 **TCM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
7	10/12,1 **PUSH** R_2	12/14,1 **PUSH** IR_2	6,5 **TM** r_1,r_2	6,5 **TM** r_1,Ir_2	10,5 **TM** R_2,R_1	10,5 **TM** IR_2,R_1	10,5 **TM** R_1,IM	10,5 **TM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
8	10,5 **DECW** RR_1	10,5 **DECW** IR_1	12,0 **LDE** r_1,Irr_2	18,0 **LDEI** Ir_1,Irr_2					↓	↓	↓	↓	↓	↓	↓	6,1 **DI**
9	6,5 **RL** R_1	6,5 **RL** IR_1	12,0 **LDE** r_2,Irr_1	18,0 **LDEI** Ir_2,Irr_1					↓	↓	↓	↓	↓	↓	↓	6,1 **EI**
A	10,5 **INCW** RR_1	10,5 **INCW** IR_1	6,5 **CP** r_1,r_2	6,5 **CP** r_1,Ir_2	10,5 **CP** R_2,R_1	10,5 **CP** IR_2,R_1	10,5 **CP** R_1,IM	10,5 **CP** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	14,0 **RET**
B	6,5 **CLR** R_1	6,5 **CLR** IR_1	6,5 **XOR** r_1,r_2	6,5 **XOR** r_1,Ir_2	10,5 **XOR** R_2,R_1	10,5 **XOR** IR_2,R_1	10,5 **XOR** R_1,IM	10,5 **XOR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	16,0 **IRET**
C	6,5 **RRC** R_1	6,5 **RRC** IR_1	12,0 **LDC** r_1,Irr_2	18,0 **LDCI** Ir_1,Irr_2				10,5 **LD** r_1,x,R_2	↓	↓	↓	↓	↓	↓	↓	6,5 **RCF**
D	6,5 **SRA** R_1	6,5 **SRA** IR_1	12,0 **LDC** r_2,Irr_1	18,0 **LDCI** Ir_2,Irr_1	20,0 **CALL*** IRR_1		20,0 **CALL** DA	10,5 **LD** r_2,x,R_1	↓	↓	↓	↓	↓	↓	↓	6,5 **SCF**
E	6,5 **RR** R_1	6,5 **RR** IR_1		6,5 **LD** r_1,IR_2	10,5 **LD** R_2,R_1	10,5 **LD** IR_2,R_1	10,5 **LD** R_1,IM	10,5 **LD** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	6,5 **CCF**
F	8,5 **SWAP** R_1	8,5 **SWAP** IR_1		6,5 **LD** Ir_1,r_2		10,5 **LD** R_2,IR_1			↓	↓	↓	↓	↓	↓	↓	6,0 **NOP**
Bytes per Instruction	2				3				2					3	1	



Legend:
R = 8-bit address
r = 4-bit address
R_1 or r_1 = Dst address
R_2 or r_2 = Src address

Sequence:
Opcode, First Operand, Second Operand

NOTE: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

ABSOLUTE MAXIMUM RATINGS

Voltages on all pins except $\overline{\text{RESET}}$
with respect to GND . . . . −0.3V to +7.0V
Operating Ambient
Temperature . . . . .See Ordering Information
Storage Temperature . . . . −65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin.

Standard conditions are as follows:

- $+4.75V \leq V_{CC} \leq +5.25V$
- GND = 0V
- $0°C \leq T_A \leq +70°C$ for S (Standard temperature)
- $-40°C \leq T_A \leq +$**100**°C for E (Extended temperature)

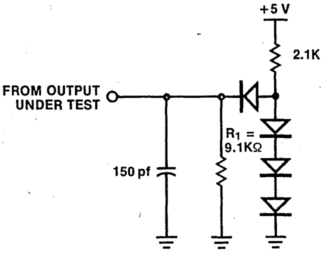


Figure 19. Test Load 1

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8	V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	−0.3	0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{IL}	Input Low Voltage	−0.3	0.8	V	
V_{RH}	Reset Input High Voltage	3.8	V_{CC}	V	See Note
V_{RL}	Reset Input Low Voltage	−0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -250\,\mu A$
V_{OL}	Output Low Voltage		0.4	V	$I_{OL} = +2.0$ mA
I_{IL}	Input Leakage	−10	10	μA	$0V \leq V_{IN} \leq +5.25V$
I_{OL}	Output Leakage	−10	10	μA	$0V \leq V_{IN} \leq +5.25V$
I_{IR}	Reset Input Current		−50	μA	$V_{CC} = +5.25V$, $V_{RL} = 0V$
I_{CC}	V_{CC} Supply Current		**150**	mA	All outputs and I/O pins floating

*The Reset line (pin 6) is used to place the Z8682 in external memory mode. This is accomplished as shown in Figure 13.

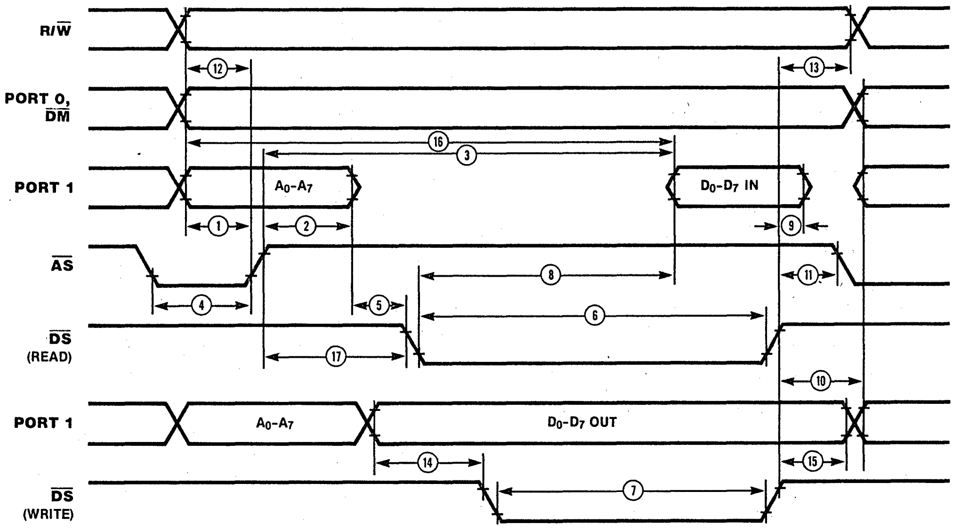


Figure 20. External I/O or Memory Read/Write Timing

AC CHARACTERISTICS

External I/O or Memory Read and Write Timing

			Z8681/82 8 MHz		Z8681 12 MHz		Z8681 16 MHz		
Number	**Symbol**	**Parameter**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Notes**
1	TdA(AS)	Address Valid to $\overline{\text{AS}}$ ↑Delay	50		35		20		2,3
2	TdAS(A)	$\overline{\text{AS}}$ ↑ to Address Float Delay	70		45		30		2,3
3	TdAS(DR)	$\overline{\text{AS}}$ ↑ to Read Data Required Valid		360		220		180	1,2,3
4	TwAS	$\overline{\text{AS}}$ Low Width	80		55		35		2,3
5	TdAz(DS)	Address Float to $\overline{\text{DS}}$ ↓	0		0		0		
6	TwDSR	$\overline{\text{DS}}$ (Read) Low Width	250		185		135		1,2,3
7	TwDSW	$\overline{\text{DS}}$ (Write) Low Width	160		110		80		1,2,3
8	TdDSR(DR)	$\overline{\text{DS}}$ ↓ to Read Data Required Valid		200		130		75	1,2,3
9	ThDR(DS)	Read Data to $\overline{\text{DS}}$ ↑ Hold Time	0		0		0		2,3
10	TdDS(A)	$\overline{\text{DS}}$ ↑ to Address Active Delay	70		45		-		2,3
11	TdDS(AS)	$\overline{\text{DS}}$ ↑ to $\overline{\text{AS}}$ ↓Delay	70		55		30		2,3
12	TdR/W(AS)	R/$\overline{\text{W}}$ Valid to $\overline{\text{AS}}$ ↑ Delay	50		30		20		2,3
13	TdDS(R/W)	$\overline{\text{DS}}$ ↑ to R/$\overline{\text{W}}$ Not Valid	60		35		30		2,3
14	TdDW(DSW)	Write Data Valid to $\overline{\text{DS}}$ (Write) ↓ Delay	50		35		25		2,3
15	TdDS(DW)	$\overline{\text{DS}}$ ↑ to Write Data Not Valid Delay	60		35		30		2,3
16	TdA(DR)	Address Valid to Read Data Required Valid		410		255		200	1,2,3
17	TdAS(DS)	$\overline{\text{AS}}$ ↑ to $\overline{\text{DS}}$ ↓ Delay	80		55		40		2,3

NOTES:
1. When using extended memory timing add 2 TpC.
2. Timing numbers given are for minimum TpC.
3. See clock cycle time dependent characteristics table.
4. 16 MHz timing is preliminary and subject to change.

* All units in nanoseconds (ns).
† Test Load 1
° All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".

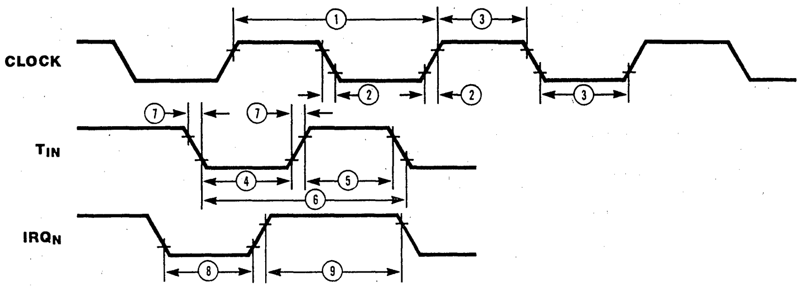


Figure 21. Additional Timing

AC CHARACTERISTICS

Additional Timing Table

			Z8681/82		Z8681		Z8681		
			8 MHz		12 MHz		16 MHz		
Number	Symbol	Parameter	Min	Max	Min	Max	Min	Max	Notes
1	TpC	Input Clock Period	125	1000	83	1000	62.5	1000	1
2	TrC,TfC	Clock Input Rise and Fall Times		25		15		10	1
3	TwC	Input Clock Width	37		70		21		1
4	TwTinL	Timer Input Low Width	100		70		50		2
5	TwTinH	Timer Input High Width	3TpC		3TpC		3TpC		2
6	TpTin	Timer Input Period	8TpC		8TpC		8TpC		2
7	TrTin,TfTin	Timer Input Rise and Fall Times		100		100		100	2
8A	TwIL	Interrupt Request Input Low Time	100		70		50		2,4
8B	TwIL	Interrupt Request Input Low Time	3TpC		3TpC		3TpC		2,5
9	TwIH	Interrupt Request Input High Time	3TpC		3TpC		3TpC		2,3

NOTES:
1. Clock timing references use 3.8V for a logic "1" and 0.8V for a logic "0".
2. Timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
3. Interrupt request via Port 3.
4. Interrupt request via Port 3 (P3$_1$-P3$_3$)
5. Interrupt request via Port 3 (P3$_0$)
6. 16 MHz timing is preliminary and subject to change.
* Units in nanoseconds (ns).

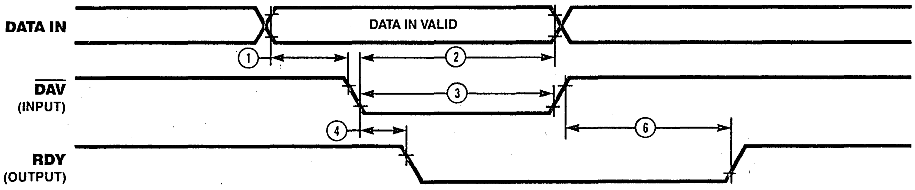


Figure 22a. Input Handshake Timing

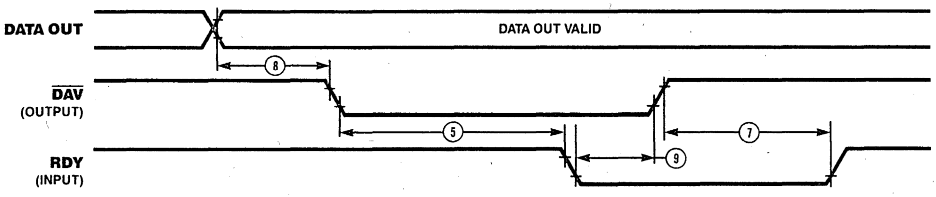


Figure 22b. Output Handshake Timing

AC CHARACTERISTICS

Handshake Timing

			Z8681/82 8 MHz		Z8681 12 MHz		Z8681 16 MHz		
Number	Symbol	Parameter	Min	Max	Min	Max	Min	Max	Notes
1	TsDI(DAV)	Data In Setup Time	0		0		0		
2	ThDI(DAV)	Data In Hold Time	230		160		145		
3	TwDAV	Data Available Width	175		120		110		
4	TdDAVIf(RDY)	DAV ↓ Input to RDY ↓ Delay		175		120		115	1,2
5	TdDAVOf(RDY)	DAV ↓ Output to RDY ↓ Delay	0		0		0		1,3
6	TdDAVIr(RDY)	DAV ↑ Input to RDY ↑ Delay		175		120		115	1,2
7	TdDAVOr(RDY)	DAV ↑ Output to RDY ↑ Delay	0		0		0		1,3
8	TdDO(DAV)	Data Out to DAV ↓ Delay	50		30		30		1
9	TdRDY(DAV)	Rdy ↓ Input to DAV ↑ Delay	0	200	0	140	0	130	1

NOTES:
1. Test load 1
2. Input handshake
3. Output handshake
4. 16 MHz timing is preliminary and subject to change.
† All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
* Units in nanoseconds (ns).

CLOCK CYCLE TIME-DEPENDENT CHARACTERISTICS

Number	Symbol	Z8681/82 8 MHz Equation	Z8681/82 12 MHz Equation
1	TdA(AS)	TpC-75	TpC-50
2	TdAS(A)	TpC-55	TpC-40
3	TdAS(DR)	4TpC-140 *	4TpC-110 *
4	TwAS	TpC-45	TpC-30
6	TwDSR	3TpC-125 *	3TpC-65 *
7	TwDSW	2TpC-90 *	2TpC-55 *
8	TdDSR(DR)	3TpC-175 *	3TpC-120 *
10	Td(DS)A	TpC-55	TpC-40
11	TdDS(AS)	TpC-55	TpC-30
12	TdR/W(AS)	TpC-75	TpC-55
13	TdDS(R/W)	TpC-65	TpC-50
14	TdDW(DSW)	TpC-75	TpC-50
15	TdDS(DW)	TpC-55	TpC-40
16	TdA(DR)	5TpC-215 *	5TpC-160 *
17	TdAS(DS)	TpC-45	TpC-30

* Add 2TpC when using extended memory timing

June 1987

Z8691 Z8®
ROMless Microcomputer

FEATURES

- Complete microcomputer, 24 I/O lines, and up to 64K bytes of addressable external space each for program and data memory.
- 143-byte register file, including 124 general-purpose registers, 3 I/O port registers, and 16 status and control registers.
- Vectored, priority interrupts for I/O, counter/timers, and UART.
- On-chip oscillator that accepts crystal or external clock drive.
- Full-duplex UART and two programmable 8-bit counter/timers, each with a 6-bit programmable prescaler.
- Register Pointer so that short, fast instructions can access any one of the nine working-register groups.
- Single +5V power supply—all I/O pins TTL compatible.
- 8 MHz/12 MHz versions.

GENERAL DESCRIPTION

The Z8691 is a ROMless version of the Z8 single-chip microcomputer. The Z8691 offers all the outstanding features of the Z8 family architecture except an on-chip program ROM. Use of external memory rather than a preprogrammed ROM enables this Z8 microcomputer to be used in low volume applications or where code flexibility is required.

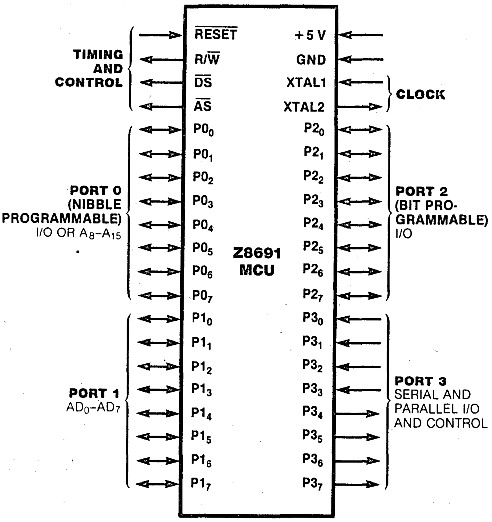

TIMING AND CONTROL
RESET
R/W
DS
AS
+5 V
GND
XTAL1
XTAL2
CLOCK
PORT 0 (NIBBLE PROGRAMMABLE) I/O OR A_8–A_{15}
P0₀ P0₁ P0₂ P0₃ P0₄ P0₅ P0₆ P0₇
Z8691 MCU
PORT 2 (BIT PROGRAMMABLE) I/O
P2₀ P2₁ P2₂ P2₃ P2₄ P2₅ P2₆ P2₇
PORT 1 AD_0–AD_7
P1₀ P1₁ P1₂ P1₃ P1₄ P1₅ P1₆ P1₇
PORT 3 SERIAL AND PARALLEL I/O AND CONTROL
P3₀ P3₁ P3₂ P3₃ P3₄ P3₅ P3₆ P3₇

Figure 1. Pin Functions

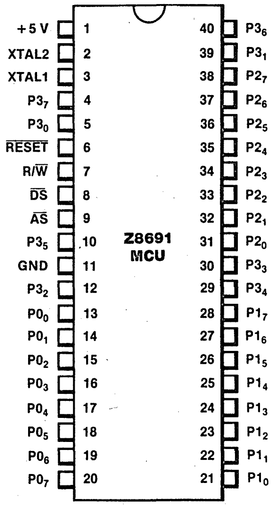


Figure 2a. 40-pin Dual-In-Line Package (DIP), Pin Assignments

The Z8691 can provide up to 16 output address lines, thus permitting an address space of up to 64K bytes of data or program memory. Eight address outputs (AD_0-AD_7) are provided by a multiplexed, 8-bit, Address/Data bus. The remaining 8 bits can be provided by the software configuration of Port 0 to output address bits A_8-A_{15}.

Available address space can be doubled (up to 128K bytes) by programming bit 4 of Port 3 ($P3_4$) to act as a data memory select output ($\overline{DM}$). The two states of $\overline{DM}$ together with the 16 address outputs can define separate data and memory address spaces of up to 64K bytes each.

There are 143 bytes of RAM located on-chip and organized as a register file of 124 general-purpose registers, 16 control and status registers, and three I/O port registers. This register file can be divided into nine groups of 16 working registers each. Configuring the register file in this manner allows the use of short format instructions; in addition, any of the individual registers can be accessed directly.

The pin functions and the pin assignments of the Z8691 40-pin and 44-pin packages are illustrated in Figures 1 and 2, respectively.

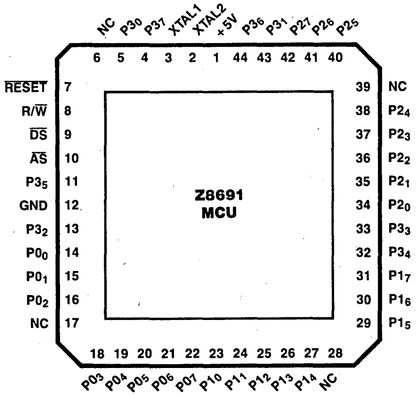


Figure 2b. 44-pin Chip Carrier, Pin Assignments

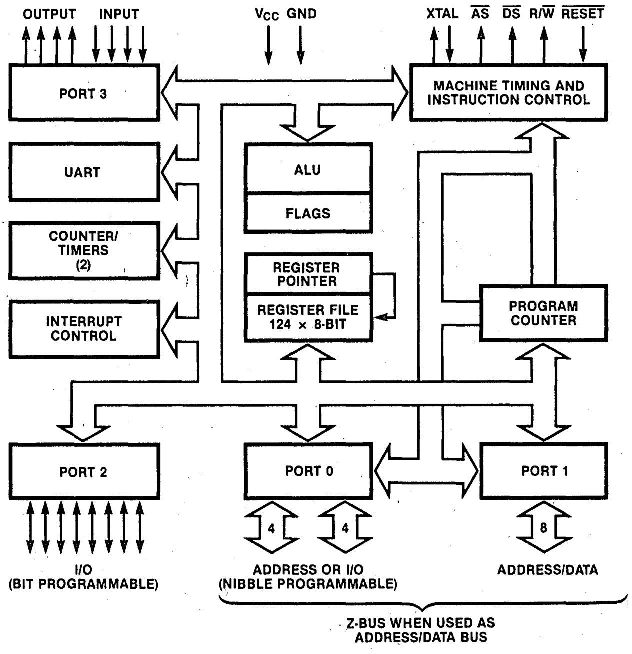


Figure 3. Functional Block Diagram

ARCHITECTURE

Z8691 architecture is characterized by a flexible I/O scheme, an efficient register and address space structure and a number of ancillary features that are helpful in many applications.

Microcomputer applications demand powerful I/O capabilities. The Z8691 fulfills this with 24 pins available for input and output. These lines are grouped into three ports of eight lines each and are configurable under software control to provide timing, status signals, serial or parallel I/O with or without handshake, and an Address bus for interfacing external memory.

Three basic address spaces are available: program memory, data memory and the register file (internal). The 143-byte random-access register file is composed of 124 general-purpose registers, three I/O port registers, and 16 control and status registers.

To unburden the program from coping with real-time problems such as serial data communication and counting/timing, an asynchronous receiver/transmitter (UART) and two counter/timers with a large number of user-selectable modes are offered on-chip. Hardware support for the UART is minimized because one of the on-chip timers supplies the bit rate. Figure 3 shows the Z8691 block diagram.

PIN DESCRIPTION

$\overline{\text{AS}}$. *Address Strobe* (output, active Low). Address Strobe is pulsed once at the beginning of each machine cycle. Addresses output via Port 1 for all external program or data memory transfers are valid at the trailing edge of $\overline{\text{AS}}$.

$\overline{\text{DS}}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each external memory transfer.

$P0_0$-$P0_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. *I/O Port Lines* (input/outputs, TTL-compatible). These 24 lines are divided into three 8-bit I/O ports that can be configured under program control for I/O or external memory interface (Figure 3).

$P1_0$-$P1_7$. *Address/Data Port** (bidirectional). Multiplexed address (A_0-A_7) and data (D_0-D_7) lines used to interface with program and data memory.

$\overline{\text{RESET}}$. *Reset* (input, active Low). $\overline{\text{RESET}}$ initializes the Z8691. After RESET the Z8691 is in the extended memory mode. When $\overline{\text{RESET}}$ is deactivated, program execution begins from program location $000C_H$.

R/$\overline{\text{W}}$. *Read/Write* (output). R/$\overline{\text{W}}$ is Low when the Z8691 is writing to external program or data memory.

XTAL1, XTAL2. *Crystal 1, Crystal 2* (time-base input and output). These pins connect a parallel-resonant crystal to the on-chip clock oscillator and buffer.

ADDRESS SPACES

Program Memory. The Z8691 addresses 64K/62K bytes of external program memory space (Figure 4).

The first 12 bytes of program memory are reserved for the interrupt vectors. These locations contain six 16-bit vectors that correspond to the six available interrupts. Program execution begins at location $000C_H$ after a reset.

Data Memory. The Z8691 can address 64K bytes of external data memory. External data memory may be included with or separated from the external program memory space. $\overline{DM}$, an optional I/O function that can be programmed to appear on pin $P3_4$, is used to distinguish between data and program memory space.

Register File. The 143-byte register file includes three I/O port registers (R0, R2, R3), 124 general-purpose registers (R4-R127) and 16 control and status registers (R240-R255). These registers are assigned the address locations shown in Figure 5.

Z8691 instructions can access registers directly or indirectly with an 8-bit address field. This also allows short 4-bit register addressing using the Register Pointer (one of the control registers). In the 4-bit mode, the register file is divided into nine working-register groups, each occupying 16 contiguous locations (Figure 5). The Register Pointer addresses the starting location of the active working-register group (Figure 6).

Stacks. Either the internal register file or the external data memory can be used for the stack. A 16-bit Stack Pointer (R254 and R255) is used for the external stack, which can reside anywhere in data memory. An 8-bit Stack Pointer (R255) is used for the internal stack that resides within the 124 general-purpose registers (R4-R127).

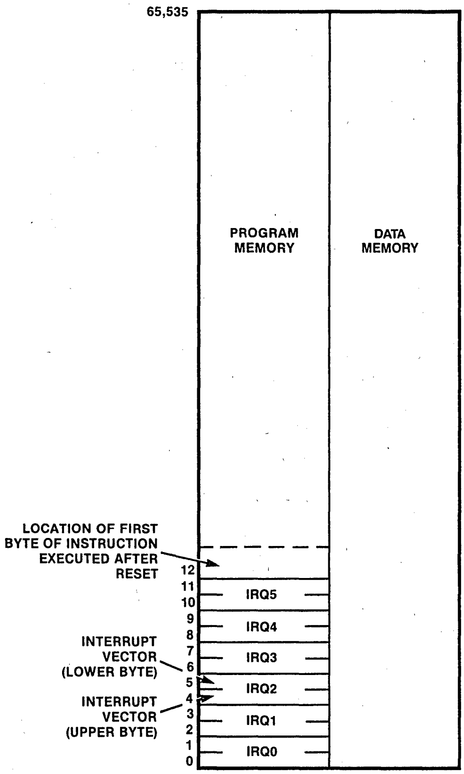

65,535
PROGRAM MEMORY
DATA MEMORY
LOCATION OF FIRST BYTE OF INSTRUCTION EXECUTED AFTER RESET
12
11
10
9
8
7
6
5
4
3
2
1
0
IRQ5
IRQ4
IRQ3
IRQ2
IRQ1
IRQ0
INTERRUPT VECTOR (LOWER BYTE)
INTERRUPT VECTOR (UPPER BYTE)

Figure 4. Program Memory Map

DEC		HEX	IDENTIFIERS
255	STACK POINTER (BITS 7-0)	FF	SPL
254	STACK POINTER (BITS 15-8)	FE	SPH
253	REGISTER POINTER	FD	RP
252	PROGRAM CONTROL FLAGS	FC	FLAGS
251	INTERRUPT MASK REGISTER	FB	IMR
250	INTERRUPT REQUEST REGISTER	FA	IRQ
249	INTERRUPT PRIORITY REGISTER	F9	IPR
248	PORTS 0-1 MODE	F8	P01M
247	PORT 3 MODE	F7	P3M
246	PORT 2 MODE	F6	P2M
245	T0 PRESCALER	F5	PRE0
244	TIMER/COUNTER 0	F4	T0
243	T1 PRESCALER	F3	PRE1
242	TIMER/COUNTER 1	F2	T1
241	TIMER MODE	F1	TMR
240	SERIAL I/O	F0	SIO
	NOT IMPLEMENTED		
127	GENERAL-PURPOSE REGISTERS	7F	
4		04	
3	PORT 3	03	P3
2	PORT 2	02	P2
1	PORT 1	01	P1
0	PORT 0	00	P0

Figure 5. The Register File

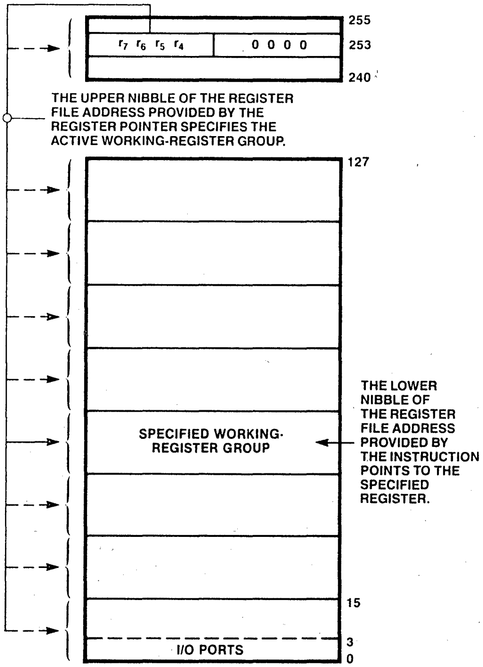

r_7 r_6 r_5 r_4
0 0 0 0
255
253
240
THE UPPER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE REGISTER POINTER SPECIFIES THE ACTIVE WORKING-REGISTER GROUP.
127
SPECIFIED WORKING-REGISTER GROUP
THE LOWER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE INSTRUCTION POINTS TO THE SPECIFIED REGISTER.
15
3
I/O PORTS
0

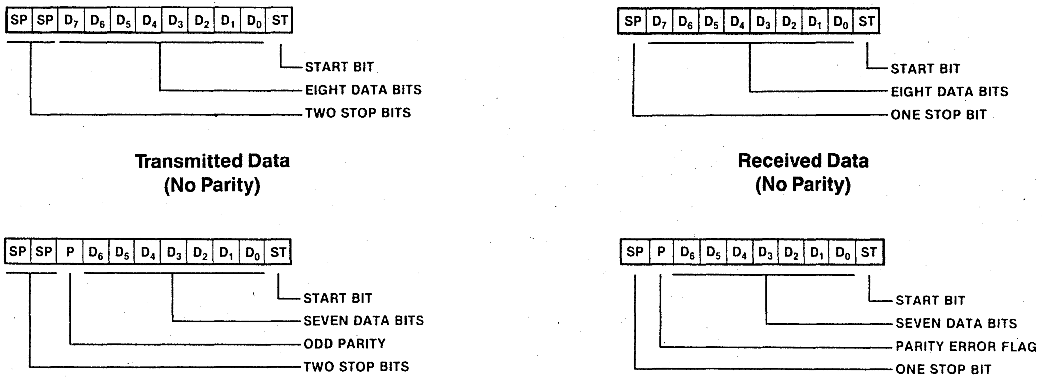
Figure 6. The Register Pointer

SERIAL INPUT/OUTPUT

Port 3 lines $P3_0$ and $P3_7$ can be programmed as serial I/O lines for full-duplex serial asynchronous receiver/transmitter operation. The bit rate is controlled by Counter/Timer 0, with a maximum rate of 62.5K bits/second at 8 MHz or 93.75K bits/second at 12 MHz on the Z8691.

The Z8691 automatically adds a start bit and two stop bits to transmitted data (Figure 7). Odd parity is also available as an option. Eight data bits are always transmitted, regardless of parity selection. If parity is enabled, the eighth data bit is used as the odd parity bit. An interrupt request (IRQ4) is generated on all transmitted characters.

Received data must have a start bit, eight data bits, and at least one stop bit. If parity is on, bit 7 of the received data is replaced by a parity error flag. Received characters generate the IRQ3 interrupt request.


SP SP D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 ST
START BIT
EIGHT DATA BITS
TWO STOP BITS
SP D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 ST
START BIT
EIGHT DATA BITS
ONE STOP BIT
SP SP P D_6 D_5 D_4 D_3 D_2 D_1 D_0 ST
START BIT
SEVEN DATA BITS
ODD PARITY
TWO STOP BITS
SP P D_6 D_5 D_4 D_3 D_2 D_1 D_0 ST
START BIT
SEVEN DATA BITS
PARITY ERROR FLAG
ONE STOP BIT

Transmitted Data (No Parity)

Received Data (No Parity)

Transmitted Data (With Parity)

Received Data (With Parity)

Figure 7. Serial Data Formats

COUNTER/TIMERS

The Z8691 contains two 8-bit programmable counter/timers (T_0 and T_1), each driven by its own 6-bit programmable prescaler. The T_1 prescaler can be driven by internal or external clock sources; however, the T_0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrements the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request—IRQ4 (T_0) or IRQ5 (T_1)—is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single-pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read any time without disturbing their value or count mode.

The clock source for T_1 is user-definable; it can be either the internal microprocessor clock divided by four, or an external signal input via Port 3. The Timer Mode register configures the external timer input as an external clock, a trigger input that can be retriggerable or nonretriggerable, or as a gate input for the internal clock. The counter/timers can be programmably cascaded by connecting the T_0 output to the input of T_1. Port 3 line $P3_6$ also serves as a timer output (T_{OUT}) through which T_0, T_1 or the internal clock can be output.

I/O PORTS

The Z8691 has 24 lines available for input and output. These lines are grouped into three ports of eight lines each and are configurable as input, output or address. Under software control, the ports can be programmed to provide address outputs, timing, status signals, serial I/O, and parallel I/O with or without handshake. All ports have active pull-ups and pull-downs compatible with TTL loads.

Port 1 is a dedicated Z-BUS compatible memory interface. The operations of Port 1 are supported by the Address Strobe ($\overline{AS}$) and Data Strobe ($\overline{DS}$) lines, and by the Read/Write (R/$\overline{W}$) and Data Memory ($\overline{DM}$) control lines. The low-order program and data memory addresses (A_0-A_7) are output through Port 1 (Figure 8) and are multiplexed with data in/out (D_0-D_7). Instruction fetch and data memory read/write operations are done through this port.

Port 1 cannot be used as a register nor can a handshake mode be used with this port.

The Z8691 wakes up with the 8 bits of Port 1 configured as address outputs for external memory. If more than eight address lines are required, additional lines can be obtained by programming Port 0 bits as address bits. The least-significant four bits of Port 0 can be configured to supply address bits A_8-A_{11} for 4K byte addressing or both nibbles of Port 0 can be configured to supply address bits A_8-A_{15} for 64K byte addressing.

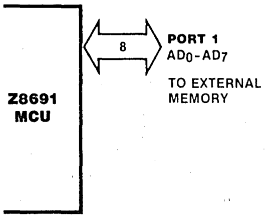


Figure 8. Port 1

Port 0 can be programmed as a nibble I/O port, or as an address port for interfacing external memory (Figure 9). When used as an I/O port, Port 0 can be placed under handshake control. In this configuration, Port 3 lines $P3_2$ and $P3_5$ are used as the handshake controls DAV_0 and RDY_0. Handshake signal assignment is dictated by the I/O direction of the upper nibble $P0_4$-$P0_7$.

For external memory references, Port 0 can provide address bits A_8-A_{11} (lower nibble) or A_8-A_{15} (lower and upper nibbles) depending on the required address space. If the address range requires 12 bits or less, the upper nibble of Port 0 can be programmed independently as I/O while the lower nibble is used for addressing.

Port 0 lines are configured as address lines A_8-A_{15} after a reset. If one or both nibbles are needed for I/O operation, they must be configured by writing to the Port 0 Mode register.

To permit the use of slow memory, an automatic wait mode of two oscillator clock cycles is configured for the bus timing of the Z8691 after each reset. The initialization routine could include reconfiguration to eliminate this extended timing mode.

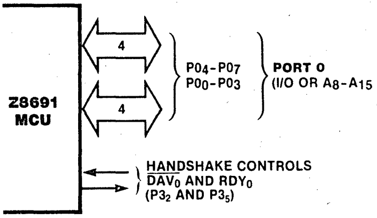


Figure 9. Port 0

Port 2 bits can be programmed independently as input or output (Figure 10). This port is always available for I/O operations. In addition, Port 2 can be configured to provide open-drain outputs.

Port 2 may also be placed under handshake control. In this configuration, Port 3 lines $P3_1$ and $P3_6$ are used as the handshake controls lines $\overline{DAV}_2$ and RDY_2. The handshake signal assignment for Port 3 lines $P3_1$ and $P3_6$ is dictated by the direction (input or output) assigned to bit 7 of Port 2.

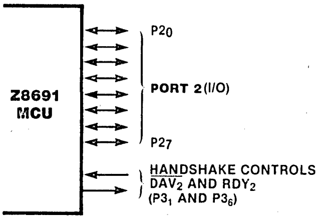


Figure 10. Port 2

Port 3 lines can be configured as I/O or control lines (Figure 11). In either case, the direction of the eight lines is fixed as four input ($P3_0$-$P3_3$) and four output ($P3_4$-$P3_7$). For serial I/O, lines $P3_0$ and $P3_7$ are programmed as serial in and serial out, respectively.

Port 3 can also provide the following control functions: handshake for Ports 0 and 2 ($\overline{DAV}$ and RDY); four external interrupt request signals (IRQ0-IRQ3); timer input and output signals (T_{IN} and T_{OUT}) and Data Memory Select ($\overline{DM}$).

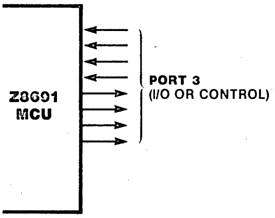


Figure 11. Port 3

INTERRUPTS

The Z8691 allows six different interrupts from eight sources: the four Port 3 lines $P3_0$-$P3_3$, Serial In, Serial Out, and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask register globally or individually enables or disables the six interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All interrupts are vectored through locations in program memory. When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the Program Counter and status flags, and accesses the program memory vector location reserved for that interrupt. This memory location and the next byte contain the 16-bit address of the interrupt service routine for that particular interrupt request. The Z8691 takes 26 system clock cycles to enter an interrupt subroutine.

Polled interrupt systems are also supported. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the Interrupt Request register polled to determine which of the interrupt requests needs service.

CLOCK

The on-chip oscillator has a high-gain, parallel-resonant amplifier for connection to a crystal or to any suitable external clock source (XTAL1 = Input, XTAL2 = Output).

The crystal source is connected across XTAL1 and XTAL2, using the recommended capacitance (C_L = 15 pf maximum) from each pin to ground. The specifications for the crystal are as follows:

□ AT cut, parallel-resonant

□ Fundamental type

□ Series resistance, $R_S \leqslant 100$ Q

□ 8 or 12 MHz maximum

INSTRUCTION SET NOTATION

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "←". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*****	Set or cleared according to operation
—	Unaffected
X	Undefined

CONDITION CODES

Value	Mnemonic	Meaning	Flags Set
1000		Always true	—
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	—

INSTRUCTION FORMATS

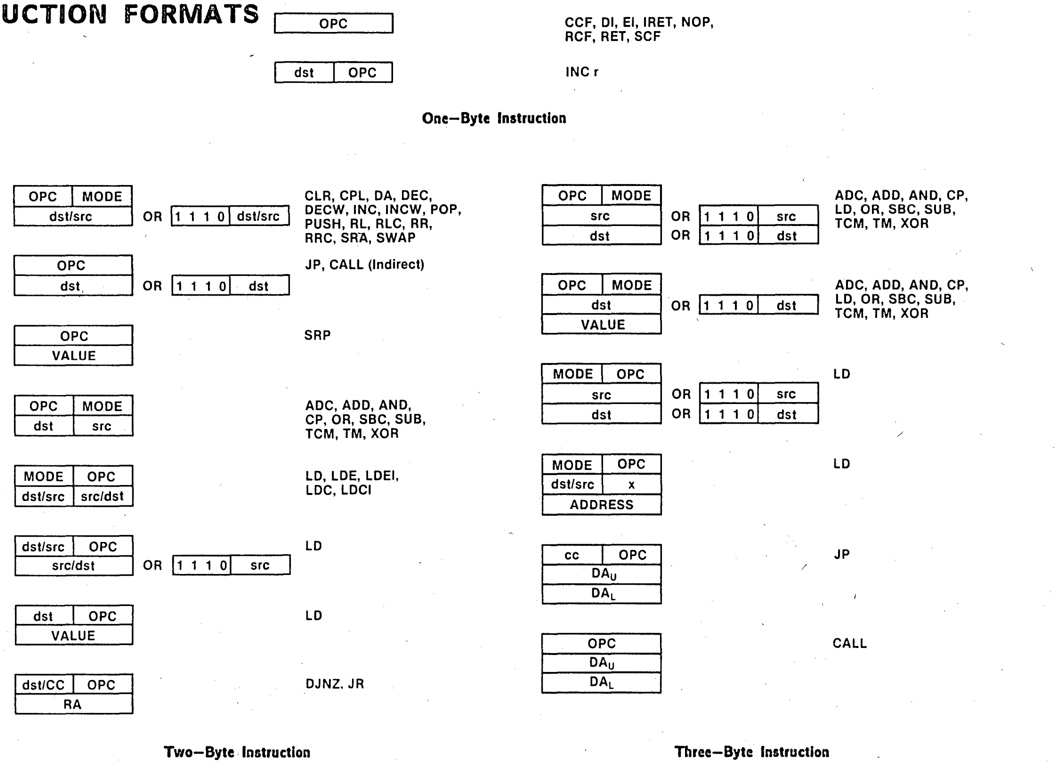


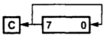
Figure 12. Instruction Formats

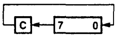
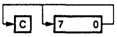
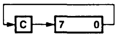
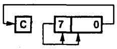
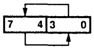
INSTRUCTION SUMMARY

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
ADC dst,src dst ← dst + src + C	(Note 1)		1□	*	*	*	*	0	*
ADD dst,src dst ← dst + src	(Note 1)		0□	*	*	*	*	0	*
AND dst,src dst ← dst AND src	(Note 1)		5□	—	*	*	0	—	—
CALL dst SP ← SP – 2 @SP ← PC; PC ← dst	DA IRR		D6 D4	—	—	—	—	—	—
CCF C ← NOT C			EF	*	—	—	—	—	—
CLR dst dst ← 0	R IR		B0 B1	—	—	—	—	—	—
COM dst dst ← NOT dst	R IR		60 61	—	*	*	0	—	—
CP dst,src dst – src	(Note 1)		A□	*	*	*	*	—	—
DA dst dst ← DA dst	R IR		40 41	*	*	*	X	—	—

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
DEC dst dst ← dst – 1	R IR		00 01	—	*	*	*	—	—
DECW dst dst ← dst – 1	RR IR		80 81	—	*	*	*	—	—
DI IMR (7) ← 0			8F	—	—	—	—	—	—
DJNZ r,dst r ← r – 1 if r ≠ 0 PC ← PC + dst Range: +127, –128	RA		rA r = 0 – F	—	—	—	—	—	—
EI IMR (7) ← 1			9F	—	—	—	—	—	—
INC dst dst ← dst + 1	r R IR		rE r = 0 – F 20 21	—	*	*	*	—	—
INCW dst dst ← dst + 1	RR IR		A0 A1	—	*	*	*	—	—

INSTRUCTION SUMMARY (Continued)

Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
IRET			BF	*	*	*	*	*	*
FLAGS ← @SP; SP ← SP + 1									
PC ← @SP; SP ← SP + 2; IMR (7) ← 1									
JP cc,dst	DA		cD	—	—	—	—	—	—
if cc is true			c = 0 – F						
PC ← dst	IRR		30						
JR cc,dst	RA		cB	—	—	—	—	—	—
if cc is true,			c = 0 – F						
PC ← PC + dst									
Range: +127, –128									
LD dst,src	r	Im	rC	—	—	—	—	—	—
dst ← src	r	R	r8						
	R	r	r9						
			r = 0 – F						
	r	X	C7						
	X	r	D7						
	r	Ir	E3						
	Ir	r	F3						
	R	R	E4						
	R	IR	E5						
	R	IM	E6						
	IR	IM	E7						
	IR	R	F5						
LDC dst,src	r	Irr	C2	—	—	—	—	—	—
dst ← src	Irr	r	D2						
LDCI dst,src	Ir	Irr	C3	—	—	—	—	—	—
dst ← src	Irr	Ir	D3						
r ← r + 1; rr ← rr + 1									
LDE dst,src	r	Irr	82	—	—	—	—	—	—
dst ← src	Irr	r	92						
LDEI dst,src	Ir	Irr	83	—	—	—	—	—	—
dst ← src	Irr	Ir	93						
r ← r + 1; rr ← rr + 1									
NOP			FF	—	—	—	—	—	—
OR dst,src	(Note 1)		4□	—	*	*	0	—	—
dst ← dst OR src									
POP dst	R		50	—	—	—	—	—	—
dst ← @SP;	IR		51						
SP ← SP + 1									
PUSH src		R	70	—	—	—	—	—	—
SP ← SP – 1; @SP ← src		IR	71						
RCF			CF	0	—	—	—	—	—
C ← 0									
RET			AF	—	—	—	—	—	—
PC ← @SP; SP ← SP + 2									
RL dst 	R		90	*	*	*	*	—	—
	IR		91						

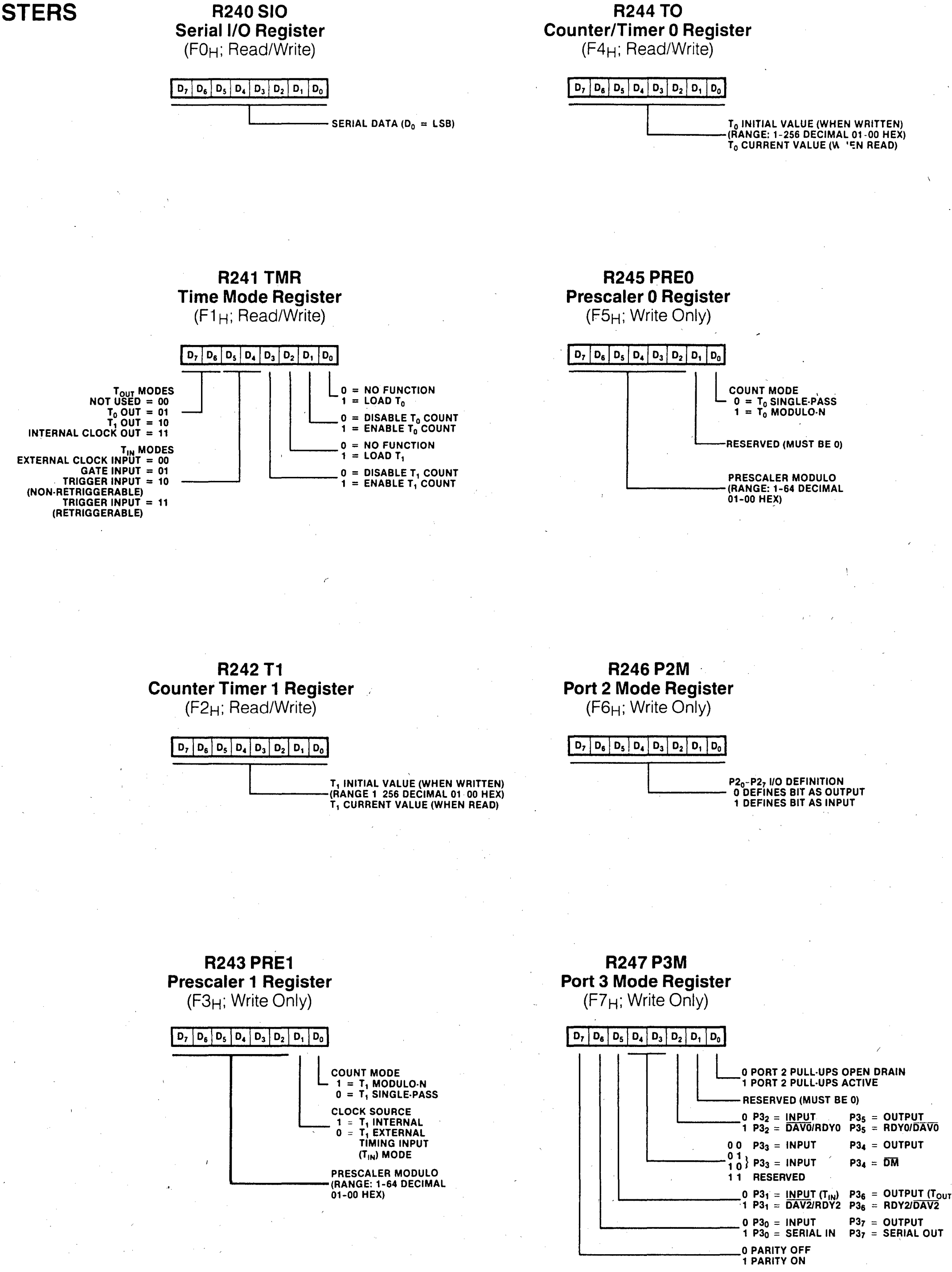
Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
RLC dst 	R		10	*	*	*	*	—	—
	IR		11						
RR dst 	R		E0	*	*	*	*	—	—
	IR		E1						
RRC dst 	R		C0	*	*	*	*	—	—
	IR		C1						
SBC dst,src	(Note 1)		3□	*	*	*	*	1	*
dst ← dst ← src ← C									
SCF			DF	1	—	—	—	—	—
C ← 1									
SRA dst 	R		D0	*	*	*	0	—	—
	IR		D1						
SRP src		Im	31	—	—	—	—	—	—
RP ← src									
SUB dst,src	(Note 1)		2□	*	*	*	*	1	*
dst ← dst ← src									
SWAP dst 	R		F0	X	*	*	X	—	—
	IR		F1						
TCM dst,src	(Note 1)		6□	—	*	*	0	—	—
(NOT dst) AND src									
TM dst,src	(Note 1)		7□	—	*	*	0	—	—
dst AND src									
XOR dst,src	(Note 1)		B□	—	*	*	0	—	—
dst ← dst XOR src									

NOTE: These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the left of the applicable addressing mode pair.

For example, the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

Addr Mode		Lower
dst	src	Opcode Nibble
r	r	$\boxed{2}$
r	Ir	$\boxed{3}$
R	R	$\boxed{4}$
R	IR	$\boxed{5}$
R	IM	$\boxed{6}$
IR	IM	$\boxed{7}$

REGISTERS



R240 SIO
Serial I/O Register
($F0_H$; Read/Write)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
SERIAL DATA (D_0 = LSB)

R241 TMR
Time Mode Register
($F1_H$; Read/Write)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_{OUT} MODES
NOT USED = 00
T_0 OUT = 01
T_1 OUT = 10
INTERNAL CLOCK OUT = 11
T_{IN} MODES
EXTERNAL CLOCK INPUT = 00
GATE INPUT = 01
TRIGGER INPUT = 10
(NON-RETRIGGERABLE)
TRIGGER INPUT = 11
(RETRIGGERABLE)
0 = NO FUNCTION
1 = LOAD T_0
0 = DISABLE T_0 COUNT
1 = ENABLE T_0 COUNT
0 = NO FUNCTION
1 = LOAD T_1
0 = DISABLE T_1 COUNT
1 = ENABLE T_1 COUNT

R242 T1
Counter Timer 1 Register
($F2_H$; Read/Write)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_1 INITIAL VALUE (WHEN WRITTEN)
(RANGE 1 256 DECIMAL 01 00 HEX)
T_1 CURRENT VALUE (WHEN READ)

R243 PRE1
Prescaler 1 Register
($F3_H$; Write Only)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
COUNT MODE
1 = T_1 MODULO-N
0 = T_1 SINGLE-PASS
CLOCK SOURCE
1 = T_1 INTERNAL
0 = T_1 EXTERNAL TIMING INPUT (T_{IN}) MODE
PRESCALER MODULO
(RANGE: 1-64 DECIMAL 01-00 HEX)

R244 TO
Counter/Timer 0 Register
($F4_H$; Read/Write)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_0 INITIAL VALUE (WHEN WRITTEN)
(RANGE: 1-256 DECIMAL 01-00 HEX)
T_0 CURRENT VALUE (WHEN READ)

R245 PRE0
Prescaler 0 Register
($F5_H$; Write Only)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
COUNT MODE
0 = T_0 SINGLE-PASS
1 = T_0 MODULO-N
RESERVED (MUST BE 0)
PRESCALER MODULO
(RANGE: 1-64 DECIMAL 01-00 HEX)

R246 P2M
Port 2 Mode Register
($F6_H$; Write Only)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
$P2_0$-$P2_7$ I/O DEFINITION
0 DEFINES BIT AS OUTPUT
1 DEFINES BIT AS INPUT

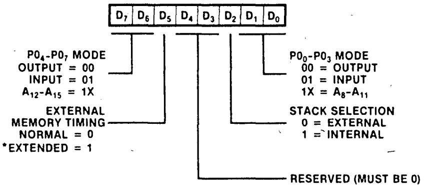
R247 P3M
Port 3 Mode Register
($F7_H$; Write Only)

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
RESERVED (MUST BE 0)
0 $P3_2$ = INPUT $P3_5$ = OUTPUT
1 $P3_2$ = $\overline{DAV0}$/RDY0 $P3_5$ = RDY0/$\overline{DAV0}$
0 0 $P3_3$ = INPUT $P3_4$ = OUTPUT
0 1 } $P3_3$ = INPUT $P3_4$ = $\overline{DM}$
1 0
1 1 RESERVED
0 $P3_1$ = INPUT (T_{IN}) $P3_6$ = OUTPUT (T_{OUT})
1 $P3_1$ = $\overline{DAV2}$/RDY2 $P3_6$ = RDY2/$\overline{DAV2}$
0 $P3_0$ = INPUT $P3_7$ = OUTPUT
1 $P3_0$ = SERIAL IN $P3_7$ = SERIAL OUT
0 PARITY OFF
1 PARITY ON

Figure 13. Control Registers

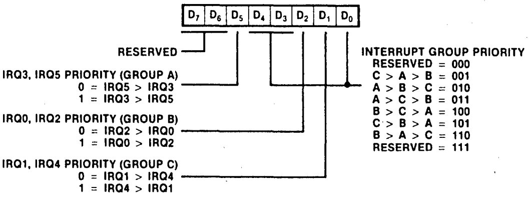
REGISTERS
(Continued)

R248 P01M
Port 0 Mode Register
($F8_H$; Write Only)

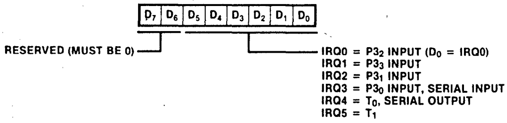


*ALWAYS EXTENDED TIMING AFTER RESET

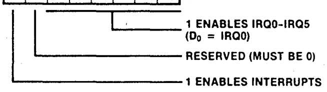
R249 IPR
Interrupt Priority Register
($F9_H$; Write Only)



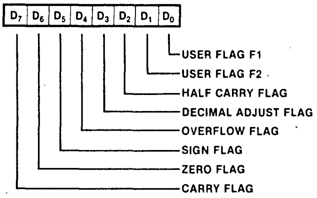
R250 IRQ
Interrupt Request Register
(FA_H; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
RESERVED (MUST BE 0)
IRQ0 = $P3_2$ INPUT (D_0 = IRQ0)
IRQ1 = $P3_3$ INPUT
IRQ2 = $P3_1$ INPUT
IRQ3 = $P3_0$ INPUT, SERIAL INPUT
IRQ4 = T_0, SERIAL OUTPUT
IRQ5 = T_1

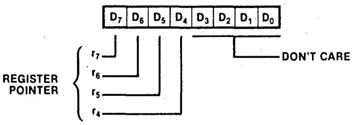
R251 IMR
Interrupt Mask Register
(FB_H; Read/Write)



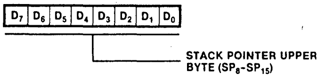
R252 FLAGS
Flag Register
(FC_H; Read/Write)



R253 RP
Register Pointer
(FD_H; Read/Write)



R254 SPH
Stack Pointer
(FE_H; Read/Write)



R255 SPL
Stack Pointer
(FF_H; Read/Write)

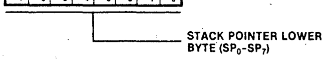
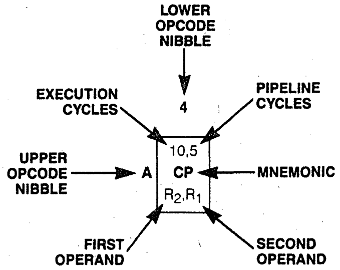


Figure 13. Control Registers (Continued)

OPCODE MAP

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6,5 **DEC** R_1	6,5 **DEC** IR_1	6,5 **ADD** r_1,r_2	6,5 **ADD** r_1,Ir_2	10,5 **ADD** R_2,R_1	10,5 **ADD** IR_2,R_1	10,5 **ADD** R_1,IM	10,5 **ADD** IR_1,IM	6,5 **LD** r_1,R_2	6,5 **LD** r_2,R_1	12/10,5 **DJNZ** r_1,RA	12/10,0 **JR** cc,RA	6,5 **LD** r_1,IM	12/10,0 **JP** cc,DA	6,5 **INC** r1	
1	6,5 **RLC** R_1	6,5 **RLC** IR_1	6,5 **ADC** r_1,r_2	6,5 **ADC** r_1,Ir_2	10,5 **ADC** R_2,R_1	10,5 **ADC** IR_2,R_1	10,5 **ADC** R_1,IM	10,5 **ADC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
2	6,5 **INC** R_1	6,5 **INC** IR_1	6,5 **SUB** r_1,r_2	6,5 **SUB** r_1,Ir_2	10,5 **SUB** R_2,R_1	10,5 **SUB** IR_2,R_1	10,5 **SUB** R_1,IM	10,5 **SUB** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
3	8,0 **JP** IRR_1	6,1 **SRP** IM	6,5 **SBC** r_1,r_2	6,5 **SBC** r_1,Ir_2	10,5 **SBC** R_2,R_1	10,5 **SBC** IR_2,R_1	10,5 **SBC** R_1,IM	10,5 **SBC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
4	8,5 **DA** R_1	8,5 **DA** IR_1	6,5 **OR** r_1,r_2	6,5 **OR** r_1,Ir_2	10,5 **OR** R_2,R_1	10,5 **OR** IR_2,R_1	10,5 **OR** R_1,IM	10,5 **OR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
5	10,5 **POP** R_1	10,5 **POP** IR_1	6,5 **AND** r_1,r_2	6,5 **AND** r_1,Ir_2	10,5 **AND** R_2,R_1	10,5 **AND** IR_2,R_1	10,5 **AND** R_1,IM	10,5 **AND** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
6	6,5 **COM** R_1	6,5 **COM** IR_1	6,5 **TCM** r_1,r_2	6,5 **TCM** r_1,Ir_2	10,5 **TCM** R_2,R_1	10,5 **TCM** IR_2,R_1	10,5 **TCM** R_1,IM	10,5 **TCM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
7	10/12,1 **PUSH** R_2	12/14,1 **PUSH** IR_2	6,5 **TM** r_1,r_2	6,5 **TM** r_1,Ir_2	10,5 **TM** R_2,R_1	10,5 **TM** IR_2,R_1	10,5 **TM** R_1,IM	10,5 **TM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
8	10,5 **DECW** RR_1	10,5 **DECW** IR_1	12,0 **LDE** r_1,Irr_2	18,0 **LDEI** Ir_1,Irr_2					↓	↓	↓	↓	↓	↓	↓	6,1 **DI**
9	6,5 **RL** R_1	6,5 **RL** IR_1	12,0 **LDE** r_2,Irr_1	18,0 **LDEI** Ir_2,Irr_1					↓	↓	↓	↓	↓	↓	↓	6,1 **EI**
A	10,5 **INCW** RR_1	10,5 **INCW** IR_1	6,5 **CP** r_1,r_2	6,5 **CP** r_1,Ir_2	10,5 **CP** R_2,R_1	10,5 **CP** IR_2,R_1	10,5 **CP** R_1,IM	10,5 **CP** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	14,0 **RET**
B	6,5 **CLR** R_1	6,5 **CLR** IR_1	6,5 **XOR** r_1,r_2	6,5 **XOR** r_1,Ir_2	10,5 **XOR** R_2,R_1	10,5 **XOR** IR_2,R_1	10,5 **XOR** R_1,IM	10,5 **XOR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	16,0 **IRET**
C	6,5 **RRC** R_1	6,5 **RRC** IR_1	12,0 **LDC** r_1,Irr_2	18,0 **LDCI** Ir_1,Irr_2				10,5 **LD** r_1,x,R_2	↓	↓	↓	↓	↓	↓	↓	6,5 **RCF**
D	6,5 **SRA** R_1	6,5 **SRA** IR_1	12,0 **LDC** r_2,Irr_1	18,0 **LDCI** Ir_2,Irr_1	20,0 **CALL*** IRR_1		20,0 **CALL** DA	10,5 **LD** r_2,x,R_1	↓	↓	↓	↓	↓	↓	↓	6,5 **SCF**
E	6,5 **RR** R_1	6,5 **RR** IR_1		6,5 **LD** r_1,IR_2	10,5 **LD** R_2,R_1	10,5 **LD** IR_2,R_1	10,5 **LD** R_1,IM	10,5 **LD** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	6,5 **CCF**
F	8,5 **SWAP** R_1	8,5 **SWAP** IR_1		6,5 **LD** Ir_1,r_2		10,5 **LD** R_2,IR_1			↓	↓	↓	↓	↓	↓	↓	6,0 **NOP**
Bytes per Instruction	2				3				2					3	1	



Legend:
R = 8-bit address
r = 4-bit address
R_1 or r_1 = Dst address
R_2 or r_2 = Src address

Sequence:
Opcode, First Operand, Second Operand

NOTE: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

ABSOLUTE MAXIMUM RATINGS

Voltages on all pins except RESET
with respect to GND . . . . −0.3V to +7.0V
Operating Ambient
Temperature. . . . . .See Ordering Information
Storage Temperature . . . . −65 °C to +150 °C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin.

Standard conditions are as follows:

- $+4.75V \leqslant V_{CC} \leqslant +5.25V$
- GND = 0V
- $0\,°C \leqslant T_A \leqslant +70\,°C$ for S (Standard temperature)
- $-40\,°C \leqslant T_A \leqslant +100\,°C$ for E (Extended temperature)

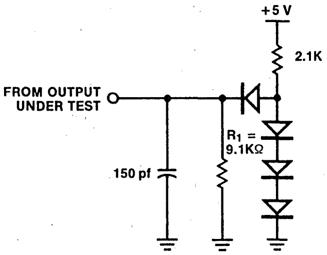


Figure 14. Test Load 1

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8	V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	−0.3	0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{IL}	Input Low Voltage	−0.3	0.8	V	
V_{RH}	Reset Input High Voltage	3.8	V_{CC}	V	
V_{RL}	Reset Input Low Voltage	−0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -250\,\mu A$
V_{OL}	Output Low Voltage		0.4	V	$I_{OL} = +2.0\,mA$
I_{IL}	Input Leakage	−10	10	μA	$V_{IN} = 0V, 5.25V$
I_{OL}	Output Leakage	−10	10	μA	$V_{IN} = 0V, 5.25V$
I_{IR}	Reset Input Current		−50	μA	$V_{CC} = +5.25V, V_{RL} = 0V$
I_{CC}	V_{CC} Supply Current		180	mA	All outputs and I/O pins floating

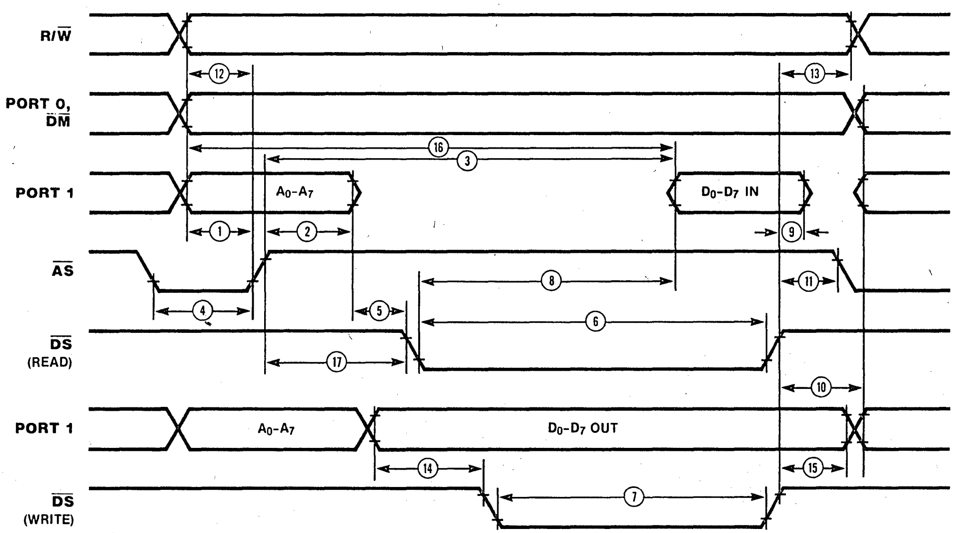


Figure 15. External I/O or Memory Read/Write Timing

AC CHARACTERISTICS

External I/O or Memory Read and Write Timing

			8 MHz		12 MHz		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes*†°
1	TdA(AS)	Address Valid to $\overline{AS}$ ↑ Delay	50		35		2,3
2	TdAS(A)	$\overline{AS}$ ↑ to Address Float Delay	70		45		2,3
3	TdAS(DR)	$\overline{AS}$ ↑ to Read Data Required Valid		360		220	1,2,3
4	TwAS	$\overline{AS}$ Low Width	80		55		2,3
5	TdAz(DS)	Address Float to $\overline{DS}$ ↓	0		0		
6	TwDSR	$\overline{DS}$ (Read) Low Width	250		185		1,2,3
7	TwDSW	$\overline{DS}$ (Write) Low Width	160		110		1,2,3
8	TdDSR(DR)	$\overline{DS}$ ↓ to Read Data Required Valid		200		130	1,2,3
9	ThDR(DS)	Read Data to $\overline{DS}$ ↑ Hold Time	0		0		
10	TdDS(A)	$\overline{DS}$ ↑ to Address Active Delay	70		45		2,3
11	TdDS(AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	70		55		2,3
12	TdR/W(AS)	R/$\overline{W}$ Valid to $\overline{AS}$ ↑ Delay	50		30		2,3
13	TdDS(R/W)	$\overline{DS}$ ↑ to R/W Not Valid	60		35		2,3
14	TdDW(DSW)	Write Data Valid to $\overline{DS}$ (Write) ↓ Delay	50		35		2,3
15	TdDS(DW)	$\overline{DS}$ ↑ to Write Data Not Valid Delay	60		35		2,3
16	TdA(DR)	Address Valid to Read Data Required Valid		410		255	1,2,3
17	TdAS(DS)	$\overline{AS}$ ↑ to $\overline{DS}$ ↓ Delay	80		55		2,3

NOTES:

1. When using extended memory timing add 2 TpC.
2. Timing numbers given are for minimum TpC.
3. See clock cycle time dependent characteristics table.

* All units in nanoseconds (ns).

† Test Load 1

° All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".

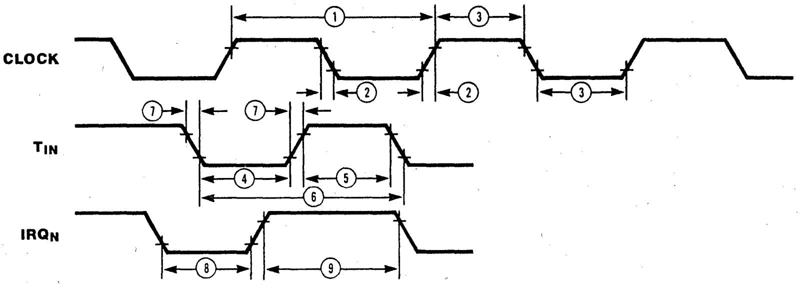


Figure 16. Additional Timing

AC CHARACTERISTICS

Additional Timing Table

			8 MHz		12 MHz		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes*
1	TpC	Input Clock Period	125	1000	83	1000	1
2	TrC,TfC	Clock Input Rise and Fall Times		25		15	1
3	TwC	Input Clock Width	37		70		1
4	TwTinL	Timer Input Low Width	100		70		2
5	TwTinH	Timer Input High Width	3TpC		3TpC		2
6	TpTin	Timer Input Period	8TpC		8TpC		2
7	TrTin,TfTin	Timer Input Rise and Fall Times		100		100	2
8A	TwIL	Interrupt Request Input Low Time	100		70		2,4
8B	TwIL	Interrupt Request Input Low Time	3TpC		3TpC		2,5
9	TwIH	Interrupt Request Input High Time	3TpC		3TpC		2,3

NOTES:

1. Clock timing references use 3.8V for a logic "1" and 0.8V for a logic "0".
2. Timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
3. Interrupt request via Port 3.
4. Interrupt request via Port 3 ($P3_1$-$P3_3$)
5. Interrupt request via Port 3 ($P3_0$)

* Units in nanoseconds (ns).

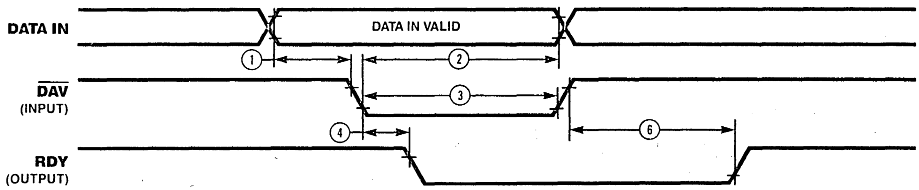


Figure 17a. Input Handshake Timing

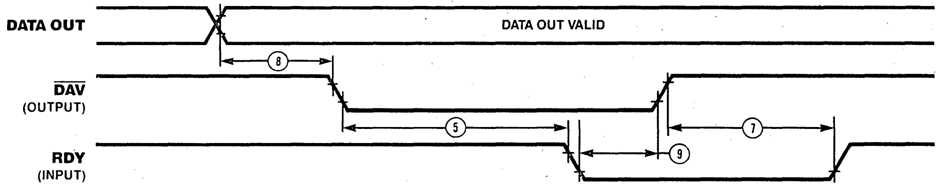


Figure 17b. Output Handshake Timing

AC CHARACTERISTICS

Handshake Timing

Number	Symbol	Parameter	8 MHz		12 MHz		Notes†*
			Min	Max	Min	Max	
1	TsDI(DAV)	Data In Setup Time	0		0		
2	ThDI(DAV)	Data In Hold Time	230		160		
3	TwDAV	Data Available Width	175		120		
4	TdDAVIf(RDY)	$\overline{\text{DAV}}$ ↓ Input to RDY ↓ Delay		175		120	1,2
5	TdDAVOf(RDY)	$\overline{\text{DAV}}$ ↓ Output to RDY ↓ Delay	0		0		1,3
6	TdDAVIr(RDY)	$\overline{\text{DAV}}$ ↑ Input to RDY ↑ Delay		175		120	1,2
7	TdDAVOr(RDY)	$\overline{\text{DAV}}$ ↑ Output to RDY ↑ Delay	0		0		1,3
8	TdDO(DAV)	Data Out to $\overline{\text{DAV}}$ ↓ Delay	50		30		1
9	TdRDY(DAV)	Rdy ↓ Input to $\overline{\text{DAV}}$ ↑ Delay	0	200	0	140	1

NOTES:
1. Test load 1
2. Input handshake
3. Output handshake
† All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
* Units in nanoseconds (ns).

CLOCK CYCLE TIME-DEPENDENT CHARACTERISTICS

Number	Symbol	8 MHz Equation	12 MHz Equation
1	TdA(AS)	TpC-75	TpC-50
2	TdAS(A)	TpC-55	TpC-40
3	TdAS(DR)	4TpC-140*	4TpC-110*
4	TwAS	TpC-45	TpC-30
6	TwDSR	3TpC-125*	3TpC-65*
7	TwDSW	2TpC-90*	2TpC-55*
8	TdDSR(DR)	3TpC-175*	3TpC-120*
10	Td(DS)A	TpC-55	TpC-40
11	TdDS(AS)	TpC-55	TpC-30
12	TdR/W(AS)	TpC-75	TpC-55
13	TdDS(R/W)	TpC-65	TpC-50
14	TdDW(DSW)	TpC-75	TpC-50
15	TdDS(DW)	TpC-55	TpC-40
16	TdA(DR)	5TpC-215*	5TpC-160*
17	TdAS(DS)	TpC-45	TpC-30

*Add 2TpC when using extended memory timing

Zilog

PRELIMINARY
Product Specification

April 1988

Z86C08 CMOS Z8 MICROCONTROLLER

FEATURES:

- Complete microcomputer with 18-pin package, 14 I/O lines, and 2K bytes of on-chip ROM.
- 142-byte register file, including 124 general purpose 8-bit registers, 3 I/O port registers, and 15 status and control registers.
- Two programmable 8-bit counter/timers, each with a 6-bit programmable prescaler.
- On-chip oscillator that accepts a crystal or external clock drive.
- 2 Volt "BROWN OUT" protection.
- Two analog comparators.
- Register pointer so that short fast instructions access any one of the eight working register groups
- Internal power on reset.
- Standby modes - HALT and STOP.
- 12 MHz
- CMOS process.

GENERAL DESCRIPTION:

The Z86C08 is a 2K ROM version of the Z8 single-chip microcomputer housed in an 18-pin DIP. It offers all the outstanding features of the Z8 family architecture in a low cost plastic DIP for price and size sensitive designs.

Flexible I/O with low power (15mA max, 2mA HALT, 10uA STOP) operation make this an ideal microcomputer for hand-held and consumer applications. It has Instruction compatibility with the entire Z8 family for easy software migration.

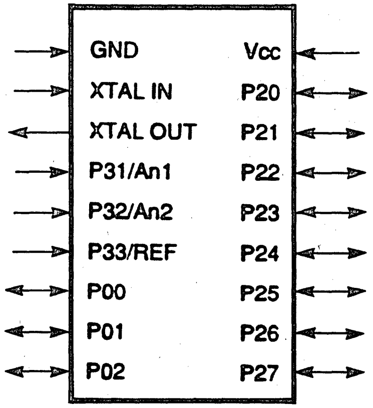


Figure 1. Pin Functions

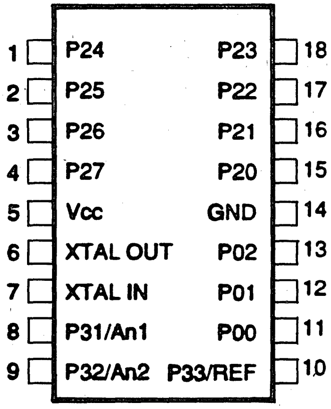


Figure 2. Pin Assignments

PIN DESCRIPTION:

$P0_0$-$P0_2$. I/O Port Lines (inputs/outputs, CMOS compatible). The three lines of Port 0 are programmable as inputs or outputs on a group basis (Figure 3).

$P2_0$-$P2_7$. I/O Port Lines (inputs/outputs, CMOS compatible). The eight lines of Port 2 are programmable as inputs or outputs on a line by line basis (Figure 3).

$P3_1$-$P3_3$. Input Port Lines (inputs, CMOS compatible). The three lines of Port 3 are programmable as digital or analog comparator inputs on a group basis (Figure 3).

XTAL IN, XTAL OUT. Crystal In, Crystal Out (time-base input and output). These pins connect a parallel-resonant crystal (12 MHz maximum) or an external single-phase clock (12 MHz maximum) to the on-chip clock oscillator and buffer.

ARCHITECTURE:

Z86C08 architecture is characterized by a flexible I/O scheme, an efficient register and address space structure and a number of ancillary features that are helpful in many applications (Figure 3).

Microcomputer applications demand powerful I/O capabilities. The Z86C08 fulfills this with 14 pins dedicated to input and output. These lines are grouped into three I/O ports which are configurable under software control.

Two basic address spaces are available: program memory and the internal register file. The register file is composed of 124 general purpose 8-bit registers, three I/O port registers, and 15 control and status registers.

To unburden the program from coping with real-time problems two counter/timers with a large number of user-selectable modes are offered on-chip.

ADDRESS SPACES:

Program Memory. The program counter addresses 2K bytes of program memory space as shown in Figure 4. The first 12 bytes of program memory are reserved for the interrupt vectors. These locations contain six 16-bit vectors that correspond to the six available interrupts.

Register File. The register file includes three I/O port registers, 124 general purpose registers (R4 - R127), and 15 control registers (R240 - R255). These registers are assigned the address locations shown in Figure 5.

Instructions can access registers directly or indirectly with an 8-bit address field. The Z86C08 also allows short 4-bit register addressing using the Register Pointer (one of the control registers). In the 4-bit mode, the register file is divided into eight working register groups, each occupying 16 contiguous locations. The Register Pointer addresses the starting location of the active working-register group (Figure 6).

STACKS. An 8-bit Stack Pointer (R255) is used for the internal stack that resides within the 124 general purpose registers (R4 - R127).

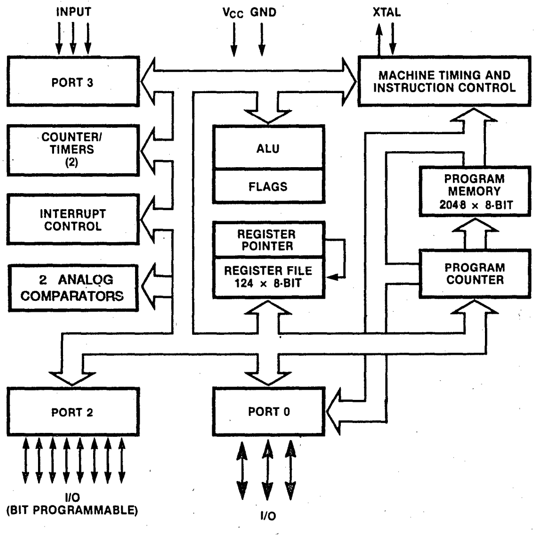


Figure 3. Functional Block Diagram

COUNTER/TIMERS:

The Z86C08 contains two 8-bit programmable counter/ timers (T0 and T1), each driven by its own 6-bit programmable prescaler. The T1 prescaler can be driven by internal or external clock sources; however, the T0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrement the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request - IRQ4 (T0) or IRQ5 (T1) - is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read at any time without disturbing their value or count mode.

The clock source for T1 is user-definable and can be retriggerable or non-retriggerable, or a gate input for the internal clock.

I/O PORTS:

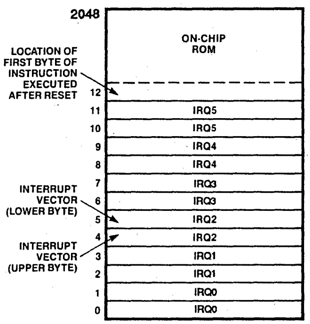
The Z86C08 has 14 lines dedicated to input and output. These lines are grouped into three ports and are configurable as input or output. All ports have active pull-ups and pull-downs compatible with CMOS loads.

Port 0 can be programmed on either inputs or outputs. The configuration is shown in Figure 7.

Port 2 bits can be programmed independently as input or output. In addition, Port 2 can be configured to provide open-drain outputs. The configuration is shown in Figure 8.

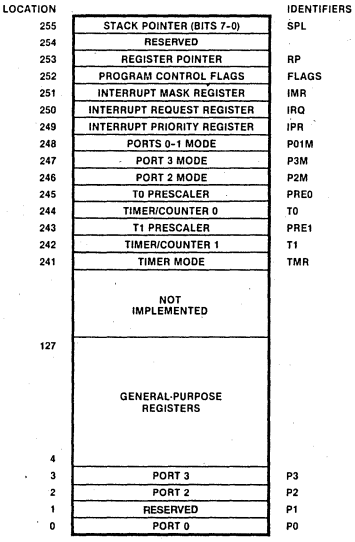
Port 3 lines can be configured as digital inputs, analog inputs, or control lines. In all cases, the direction of these three lines is fixed as inputs.

Port 3 can also provide the following control functions: four external interrupt request signals (IRQ0, IRQ1, IRQ2 and IRQ3) or timer input signal (TIN). The configuration of Port 3 is shown in Figure 9.



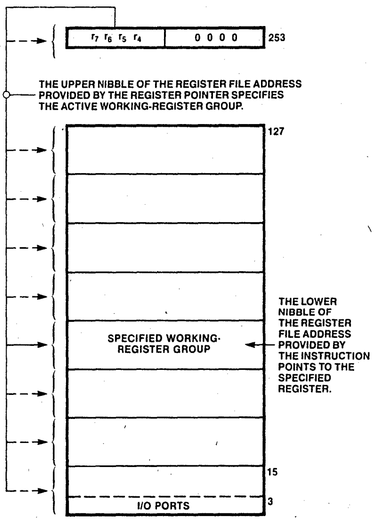
2048
ON-CHIP ROM
LOCATION OF FIRST BYTE OF INSTRUCTION EXECUTED AFTER RESET
12
11 IRQ5
10 IRQ5
9 IRQ4
8 IRQ4
7 IRQ3
6 IRQ3
5 IRQ2
4 IRQ2
3 IRQ1
2 IRQ1
1 IRQ0
0 IRQ0
INTERRUPT VECTOR (LOWER BYTE)
INTERRUPT VECTOR (UPPER BYTE)

Figure 4. Program Memory Map



LOCATION		IDENTIFIERS
255	STACK POINTER (BITS 7-0)	SPL
254	RESERVED	
253	REGISTER POINTER	RP
252	PROGRAM CONTROL FLAGS	FLAGS
251	INTERRUPT MASK REGISTER	IMR
250	INTERRUPT REQUEST REGISTER	IRQ
249	INTERRUPT PRIORITY REGISTER	IPR
248	PORTS 0-1 MODE	P01M
247	PORT 3 MODE	P3M
246	PORT 2 MODE	P2M
245	T0 PRESCALER	PRE0
244	TIMER/COUNTER 0	T0
243	T1 PRESCALER	PRE1
242	TIMER/COUNTER 1	T1
241	TIMER MODE	TMR
	NOT IMPLEMENTED	
127	GENERAL-PURPOSE REGISTERS	
4		
3	PORT 3	P3
2	PORT 2	P2
1	RESERVED	P1
0	PORT 0	P0

Figure 5. Register File



r7 r6 r5 r4
0 0 0 0
253
THE UPPER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE REGISTER POINTER SPECIFIES THE ACTIVE WORKING-REGISTER GROUP.
127
SPECIFIED WORKING-REGISTER GROUP
THE LOWER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE INSTRUCTION POINTS TO THE SPECIFIED REGISTER.
15
3
I/O PORTS

Figure 6. Register Pointer

Figure 4. Program Memory Map Figure 5. Register File Figure 6. Register Pointer

INTERRUPTS:

The Z86C08 allows six different interrupts from five sources: the three Port 3 lines P31 - P33, both the rising and falling edge of P32 (AN2), the falling edge of P31 (AN1) and P32 (REF - Figure 9), and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask Register globally or individually enables or disables the six interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All Z86C08 interrupts are vectored through locations in program memory. When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the Program Counter and status flags, and branches to the program memory vector location reserved for that interrupt. This memory location and the next byte contain the 16-bit address of the interrupt service routine for that particular interrupt request.

Polled interrupt systems are also supported. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the interrupt request register polled to determine which of the interrupt requests needs service. Interrupt sources and corresponding interrupts are shown in Table 2.

STANDBY MODE:

The Z86C08 has two standby modes which are entered by executing either:

- **STOP**
- **HALT**

The STOP instruction stops the internal clock and external crystal oscillation; the HALT instruction stops the internal clock but not crystal oscillation.

The STOP mode can be released by two methods. The first method is a RESET of the device by removing Vcc. The second method is if P27 is configured as an input line when the device executes the STOP instruction. A low input condition on P27 releases the STOP mode. Program execution under both conditions begins at location %000C(HEX). However, when P27 is used to release the STOP mode the I/O port mode registers are not reconfigured to their default power-on conditions. This prevents any I/O, configured as output when the STOP instruction was executed, from glitching to an unknown state.

The HALT mode is released by an interrupt on Port 3 input, a time-out in Timer 0 or Timer 1, or by a RESET of the device. To complete an instruction prior to entering standby mode,use the instructions:

NOP
HALT or STOP

To use the P27 release approach with STOP mode, use the following instructions:

OR P2, #% 80
NOP
STOP

RESET:

Power-On Reset is in the Z86C08.The Z86C08 waits for 50 to 150 ms + 18 crystal clocks (Figure 10) while power is on, and then jumps to the starting address %000C(HEX). The control register Reset value is listed in Table 1.

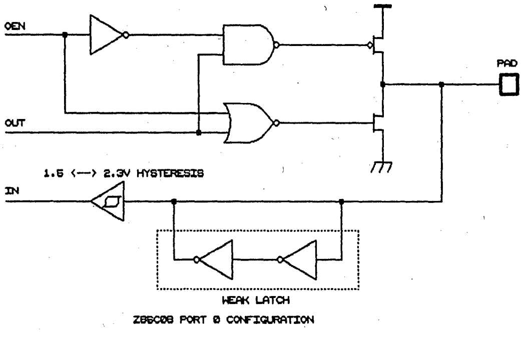


Figure 7. Z86C08 Port 0 Configuration

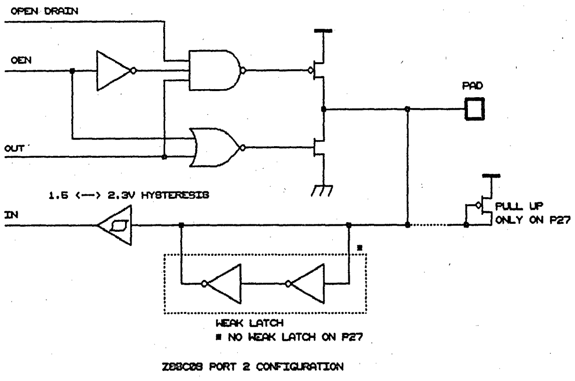


Figure 8. Z86C08 Port 2 Configuration

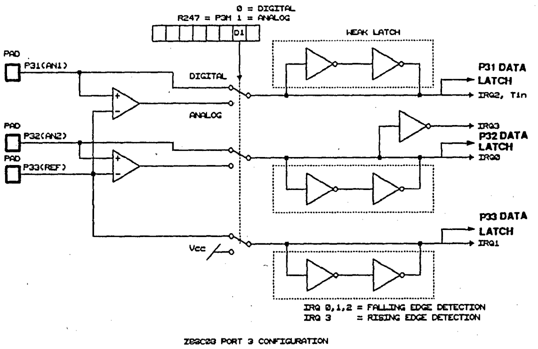


Figure 9. Z86C08 Port 3 Configuration

Table 1. Z86C08 Control Registers

86C08 control registers :

Addr.	reg.	Reset condition	Commments
F1	TMR	0 0 0 0 0 0 0 0	
F2	T1	U U U U U U U U	
F3	PRE1	U U U U U U 0 0	
F4	T0	U U U U U U U U	
F5	PRE0	U U U U U U U 0	
F6 *	P2M	1 1 1 1 1 1 1 1	Inputs after Reset
F7 *	P3M	U U U U U U 0 0	
F8 *	P01M	U U U 0 U U 0 1	
F9	IPR	U U U U U U U U	
FA	IRQ	U U 0 0 0 0 0 0	IRQ3 is used for pos. edge detection
FB	IMR	0 U U U U U U U	
FC	FLAGS	U U U U U U U U	
FD	RP	0 0 0 0 0 0 0 0	
FE	SPH	U U U U U U U U	Not used, stack always internal
FF	SPL	U U U U U U U U	

* Not reset after a low on P27 to get out of stop mode

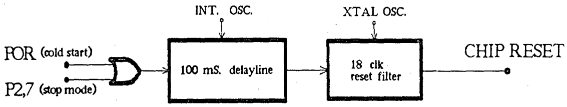


Figure 10. Internal Reset Configuration

Table 2. Interrupt Types, Sources, and Vectors

Source	Name	Vector Location	Comments
AN2 ($P3_{2l}$)	IRQ_0	0,1	External ↓ Edge Trig.
REF ($P3_3$)	IRQ_1	2,3	External ↓ Edge Trig.
AN1 ($P3_1$)	IRQ_2	4,5	External ↓ Edge Trig.
AN2 ($P3_2$)	IRQ_3	6,7	External ↑ Edge Trig.
T0	IRQ_4	8,9	Internal
T1	IRQ_5	10,11	Internal

Figure 10. Internal Reset Configuration

WATCH DOG TIMER (WDT):

The Watch Dog Timer (WDT) should be refreshed within 15 ms. If not refreshed, then the Z86C08 resets itself.

WDT: 5F(HEX).

CLOCK:

The on-chip oscillator has a high-gain, parallel-resonant amplifier for connection to a crystal, ceramic resonator, or to any suitable external clock source (XTAL IN = Input, XTAL OUT = Output).

The crystal source is connected across XTAL IN and XTAL OUT, using the recommended capacitors (C_L = 15 pF) from each pin to ground. The specifications for the crystal are as follows:

- AT cut, parallel resonant
- Fundamental type, 12 MHz max
- Series resistance, RS $\leq$ 100 ohm

The oscillator configuration is shown in Figure 11.

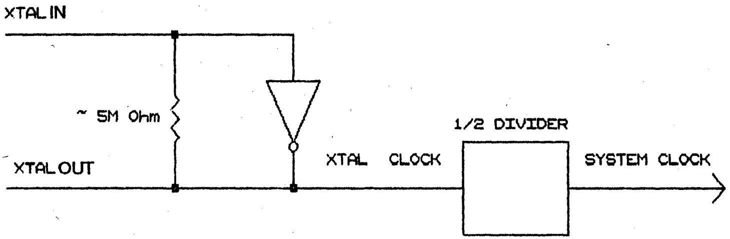


Figure 11 . Z86C08 Crystal Input Config.

PORT 3 COMPARATORS:

The 86C08's port 3 inputs include two analog comparators for added interface flexibility. Interrupts are generated on either edge of comparator 2's output, or on the falling edge of comparator 1's output. The block diagram is shown in Figure 9. , Comparator outputs may be used for interrupt generation, Port 3 data inputs, or Tin in the case of AN1 (P31). Alternatively, the comparators may be disabled, freeing the reference input (P33) for use as IRQ1 and/or P33 input.

The dual comparator (common inverting terminal) features a single power supply which discontinues power in stop mode. The common voltage range is 0-4V; the power supply and common mode rejection ratios are 90db and 60db, respectively. See comparator specifications for details (Page 16).

Typical applications for the on-board comparators include: zero crossing detection, analog-to-digital conversion, voltage scaling, and threshold detection.

INSTRUCTION SET NOTATION

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or, indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "←". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

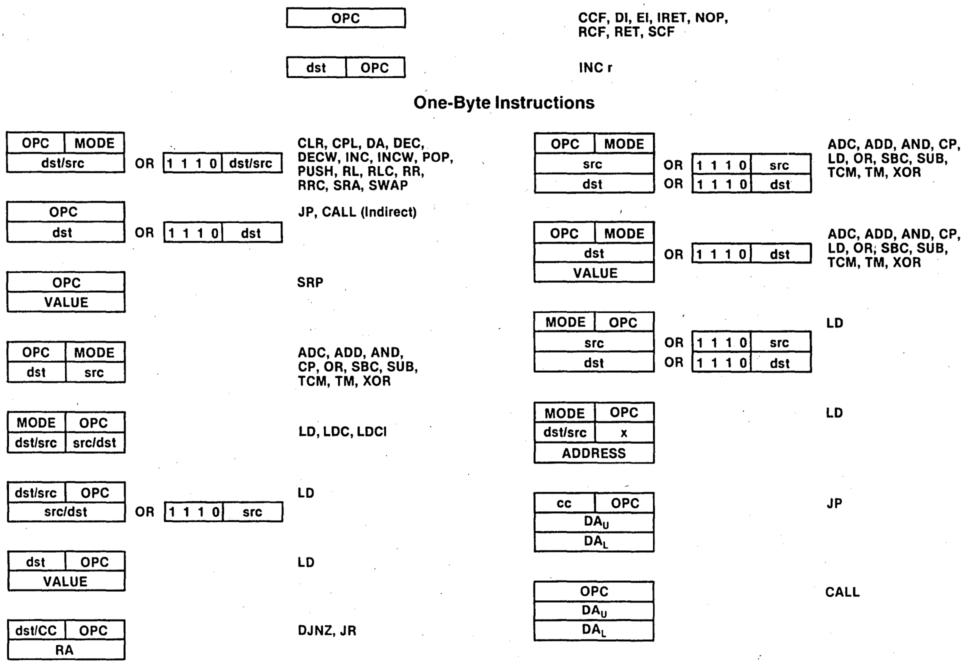
Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*****	Set or cleared according to operation
—	Unaffected
X	Undefined

CONDITION CODES

Value	Mnemonic	Meaning	Flags Set
1000		Always true	—
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	—

INSTRUCTION FORMATS



One-Byte Instructions

Two-Byte Instructions

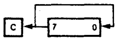
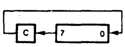
Three-Byte Instructions

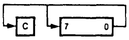
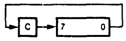
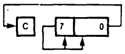
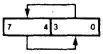
Figure 12. Instruction Formats

INSTRUCTION SUMMARY

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
ADC dst,src dst ← dst + src + C	(Note 1)		1□	*	*	*	*	0	*
ADD dst,src dst ← dst + src	(Note 1)		0□	*	*	*	*	0	*
AND dst,src dst ← dst AND src	(Note 1)		5□	—	*	*	0	—	—
CALL dst SP ← SP – 2 @SP ← PC; PC ← dst	DA IRR		D6 D4	—	—	—	—	—	—
CCF C ← NOT C			EF	*	—	—	—	—	—
CLR dst dst ← 0	R IR		B0 B1	—	—	—	—	—	—
COM dst dst ← NOT dst	R IR		60 61	—	*	*	0	—	—
CP dst,src dst – src	(Note 1)		A□	*	*	*	*	—	—
DA dst dst ← DA dst	R IR		40 41	*	*	*	X	—	—
DEC dst dst ← dst – 1	R IR		00 01	—	*	*	*	—	—
DECW dst dst ← dst – 1	RR IR		80 81	—	*	*	*	—	—
DI IMR (7) ← 0			8F	—	—	—	—	—	—
DJNZ r,dst r ← r – 1 if r ≠ 0 PC ← PC + dst Range: +127, –128	RA		rA r = 0 – F	—	—	—	—	—	—
EI IMR (7) ← 1			9F	—	—	—	—	—	—
HALT			7F						
INC dst dst ← dst + 1	r R IR		rE r = 0 – F 20 21	—	*	*	*	—	—
INCW dst dst ← dst + 1	RR IR		A0 A1	—	*	*	*	—	—

INSTRUCTION SUMMARY (Continued)

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
IRET			BF	*	*	*	*	*	*
FLAGS ← @SP; SP ← SP + 1									
PC ← @SP; SP ← SP + 2; IMR (7) ← 1									
JP cc,dst	DA		cD	—	—	—	—	—	—
if cc is true			c = 0 – F						
PC ← dst	IRR		30						
JR cc,dst	RA		cB	—	—	—	—	—	—
if cc is true,			c = 0 – F						
PC ← PC + dst									
Range: + 127, – 128									
LD dst,src	r	Im	rC	—	—	—	—	—	—
dst ← src	r	R	r8						
	R	r	r9						
			r = 0 – F						
	r	X	C7						
	X	r	D7						
	r	Ir	E3						
	Ir	r	F3						
	R	R	E4						
	R	IR	E5						
	R	IM	E6						
	IR	IM	E7						
	IR	R	F5						
LDC dst,src	r	Irr	C2	—	—	—	—	—	—
dst ← src	Irr	r	D2						
LDCI dst,src	Ir	Irr	C3	—	—	—	—	—	—
dst ← src	Irr	Ir	D3						
r ← r + 1; rr ← rr + 1									
LDE dst,src	r	Irr	82	—	—	—	—	—	—
dst ← src	Irr	r	92						
LDEI dst,src	Ir	Irr	83	—	—	—	—	—	—
dst ← src	Irr	Ir	93						
r ← r + 1; rr ← rr + 1									
NOP			FF	—	—	—	—	—	—
OR dst,src	(Note 1)		4□	—	*	*	0	—	—
dst ← dst OR src									
POP dst	R		50	—	—	—	—	—	—
dst ← @SP;	IR		51						
SP ← SP + 1									
PUSH src		R	70	—	—	—	—	—	—
SP ← SP – 1; @SP ← src		IR	71						
RCF			CF	0	—	—	—	—	—
C ← 0									
RET			AF	—	—	—	—	—	—
PC ← @SP; SP ← SP + 2									
RL dst 	R		90	*	*	*	*	—	—
	IR		91						
RLC dst 	R		10	*	*	*	*	—	—
	IR		11						

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
RR dst 	R		E0	*	*	*	*	—	—
	IR		E1						
RRC dst 	R		C0	*	*	*	*	—	—
	IR		C1						
SBC dst,src	(Note 1)		3□	*	*	*	*	1	*
dst ← dst ← src ← C									
SCF			DF	1	—	—	—	—	—
C ← 1									
SRA dst 	R		D0	*	*	*	0	—	—
	IR		D1						
SRP src		Im	31	—	—	—	—	—	—
RP ← src									
STOP			6F						
SUB dst,src	(Note 1)		2□	*	*	*	*	1	*
dst ← dst ← src									
SWAP dst 	R		F0	X	*	*	X	—	—
	IR		F1						
TCM dst,src	(Note 1)		6□	—	*	*	0	—	—
(NOT dst) AND src									
TM dst,src	(Note 1)		7□	—	*	*	0	—	—
dst AND src									
WDT			**5F**	--	--	--	--	--	--
XOR dst,src	(Note 1)		B□	—	*	*	0	—	—
dst ← dst XOR src									

NOTE: These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the left of the applicable addressing mode pair.

For example, the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

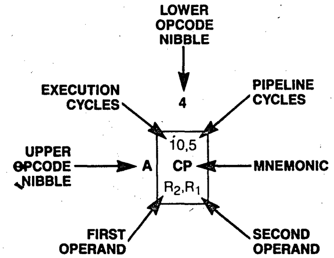
Addr Mode dst	Addr Mode src	Lower Opcode Nibble
r	r	2
r	Ir	3
R	R	4
R	IR	5
R	IM	6
IR	IM	7

OPCODE MAP

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6.5 **DEC** R_1	6.5 **DEC** IR_1	6.5 **ADD** $r_1.r_2$	6.5 **ADD** $r_1.Ir_2$	10.5 **ADD** $R_2.R_1$	10.5 **ADD** $IR_2.R_1$	10.5 **ADD** $R_1.IM$	10.5 **ADD** $IR_1.IM$	6.5 **LD** $r_1.R_2$	6.5 **LD** $r_2.R_1$	12/10.5 **DJNZ** $r_1.RA$	12/10.0 **JR** cc.RA	6.5 **LD** $r_1.IM$	12/10.0 **JP** cc.DA	6.5 **INC** r1	
1	6.5 **RLC** R_1	6.5 **RLC** IR_1	6.5 **ADC** $r_1.r_2$	6.5 **ADC** $r_1.Ir_2$	10.5 **ADC** $R_2.R_1$	10.5 **ADC** $IR_2.R_1$	10.5 **ADC** $R_1.IM$	10.5 **ADC** $IR_1.IM$	↓	↓	↓	↓	↓	↓	↓	
2	6.5 **INC** R_1	6.5 **INC** IR_1	6.5 **SUB** $r_1.r_2$	6.5 **SUB** $r_1.Ir_2$	10.5 **SUB** $R_2.R_1$	10.5 **SUB** $IR_2.R_1$	10.5 **SUB** $R_1.IM$	10.5 **SUB** $IR_1.IM$	↓	↓	↓	↓	↓	↓	↓	
3	8.0 **JP** IRR_1	6.1 **SRP** IM	6.5 **SBC** $r_1.r_2$	6.5 **SBC** $r_1.Ir_2$	10.5 **SBC** $R_2.R_1$	10.5 **SBC** $IR_2.R_1$	10.5 **SBC** $R_1.IM$	10.5 **SBC** $IR_1.IM$	↓	↓	↓	↓	↓	↓	↓	
4	8.5 **DA** R_1	8.5 **DA** IR_1	6.5 **OR** $r_1.r_2$	6.5 **OR** $r_1.Ir_2$	10.5 **OR** $R_2.R_1$	10.5 **OR** $IR_2.R_1$	10.5 **OR** $R_1.IM$	10.5 **OR** $IR_1.IM$	↓	↓	↓	↓	↓	↓	↓	
5	10.5 **POP** R_1	10.5 **POP** IR_1	6.5 **AND** $r_1.r_2$	6.5 **AND** $r_1.Ir_2$	10.5 **AND** $R_2.R_1$	10.5 **AND** $IR_2.R_1$	10.5 **AND** $R_1.IM$	10.5 **AND** $IR_1.IM$	↓	↓	↓	↓	↓	↓	↓	**6.0** **WDT**
6	6.5 **COM** R_1	6.5 **COM** IR_1	6.5 **TCM** $r_1.r_2$	6.5 **TCM** $r_1.Ir_2$	10.5 **TCM** $R_2.R_1$	10.5 **TCM** $IR_2.R_1$	10.5 **TCM** $R_1.IM$	10.5 **TCM** $IR_1.IM$	↓	↓	↓	↓	↓	↓	↓	**6.0** **STOP**
7	10/12.1 **PUSH** R_2	12/14.1 **PUSH** IR_2	6.5 **TM** $r_1.r_2$	6.5 **TM** $r_1.Ir_2$	10.5 **TM** $R_2.R_1$	10.5 **TM** $IR_2.R_1$	10.5 **TM** $R_1.IM$	10.5 **TM** $IR_1.IM$	↓	↓	↓	↓	↓	↓	↓	**7.0** **HALT**
8	10.5 **DECW** RR_1	10.5 **DECW** IR_1							↓	↓	↓	↓	↓	↓	↓	6.1 **DI**
9	6.5 **RL** R_1	6.5 **RL** IR_1							↓	↓	↓	↓	↓	↓	↓	6.1 **EI**
A	10.5 **INCW** RR_1	10.5 **INCW** IR_1	6.5 **CP** $r_1.r_2$	6.5 **CP** $r_1.Ir_2$	10.5 **CP** $R_2.R_1$	10.5 **CP** $IR_2.R_1$	10.5 **CP** $R_1.IM$	10.5 **CP** $IR_1.IM$	↓	↓	↓	↓	↓	↓	↓	14.0 **RET**
B	6.5 **CLR** R_1	6.5 **CLR** IR_1	6.5 **XOR** $r_1.r_2$	6.5 **XOR** $r_1.Ir_2$	10.5 **XOR** $R_2.R_1$	10.5 **XOR** $IR_2.R_1$	10.5 **XOR** $R_1.IM$	10.5 **XOR** $IR_1.IM$	↓	↓	↓	↓	↓	↓	↓	16.0 **IRET**
C	6.5 **RRC** R_1	6.5 **RRC** IR_1	12.0 **LDC** $r_1.Irr_2$	18.0 **LDCI** $Ir_1.Irr_2$				10.5 **LD** $r_1.x.R_2$	↓	↓	↓	↓	↓	↓	↓	6.5 **RCF**
D	6.5 **SRA** R_1	6.5 **SRA** IR_1	12.0 **LDC** $r_2.Irr_1$	18.0 **LDCI** $Ir_2.Irr_1$	20.0 **CALL*** IRR_1		20.0 **CALL** DA	10.5 **LD** $r_2.x.R_1$	↓	↓	↓	↓	↓	↓	↓	6.5 **SCF**
E	6.5 **RR** R_1	6.5 **RR** IR_1		6.5 **LD** $r_1.IR_2$	10.5 **LD** $R_2.R_1$	10.5 **LD** $IR_2.R_1$	10.5 **LD** $R_1.IM$	10.5 **LD** $IR_1.IM$	↓	↓	↓	↓	↓	↓	↓	6.5 **CCF**
F	8.5 **SWAP** R_1	8.5 **SWAP** IR_1		6.5 **LD** $Ir_1.r_2$		10.5 **LD** $R_2.IR_1$			↓	↓	↓	↓	↓	↓	↓	6.0 **NOP**
	2				3				2					3	1	

Bytes per Instruction



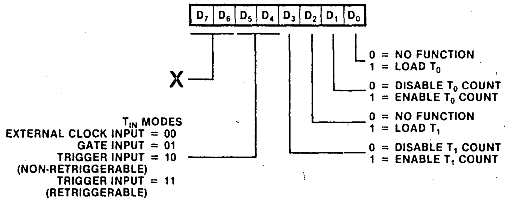
Legend:
R = 8-bit address
r = 4-bit address
R_1 or r_1 = Dst address
R_2 or r_2 = Src address

Sequence:
Opcode, First Operand, Second Operand

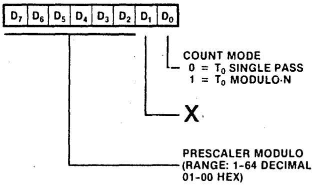
NOTE: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

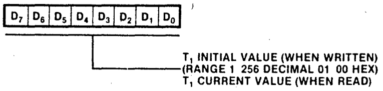
R241 TMR
TIMER MODE REGISTER
($F1_H$; Read/Write)



R245 PRE0
PRESCALER 0 REGISTER
($F5_H$; Write Only)

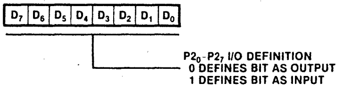


R242 T1
COUNTER TIMER 1 REGISTER
($F2_H$; Read/Write)



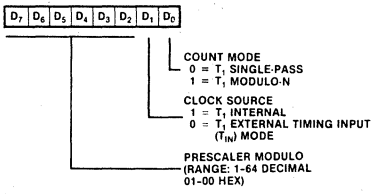
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_1 INITIAL VALUE (WHEN WRITTEN)
(RANGE 1 256 DECIMAL 01 00 HEX)
T_1 CURRENT VALUE (WHEN READ)

R246 P2M
PORT 2 MODE REGISTER
($F6_H$; Write Only)

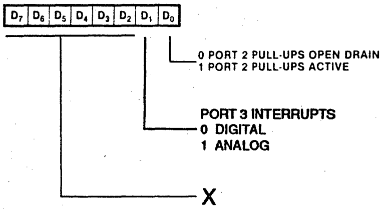


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
$P2_0$-$P2_7$ I/O DEFINITION
0 DEFINES BIT AS OUTPUT
1 DEFINES BIT AS INPUT

R243 PRE1
PRESCALER 1 REGISTER
($F3_H$; Write Only)

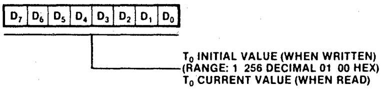


R247 P3M
PORT 3 MODE REGISTER
($F7_H$; Write Only)



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
PORT 3 INTERRUPTS
0 DIGITAL
1 ANALOG
X

R244 T0
COUNTER/TIMER 0 REGISTER
($F4_H$; Read/Write)



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_0 INITIAL VALUE (WHEN WRITTEN)
(RANGE: 1 256 DECIMAL 01 00 HEX)
T_0 CURRENT VALUE (WHEN READ)

NOTE: **All "don't care" bits return a "1" when read.**

Figure 16 Control Registers

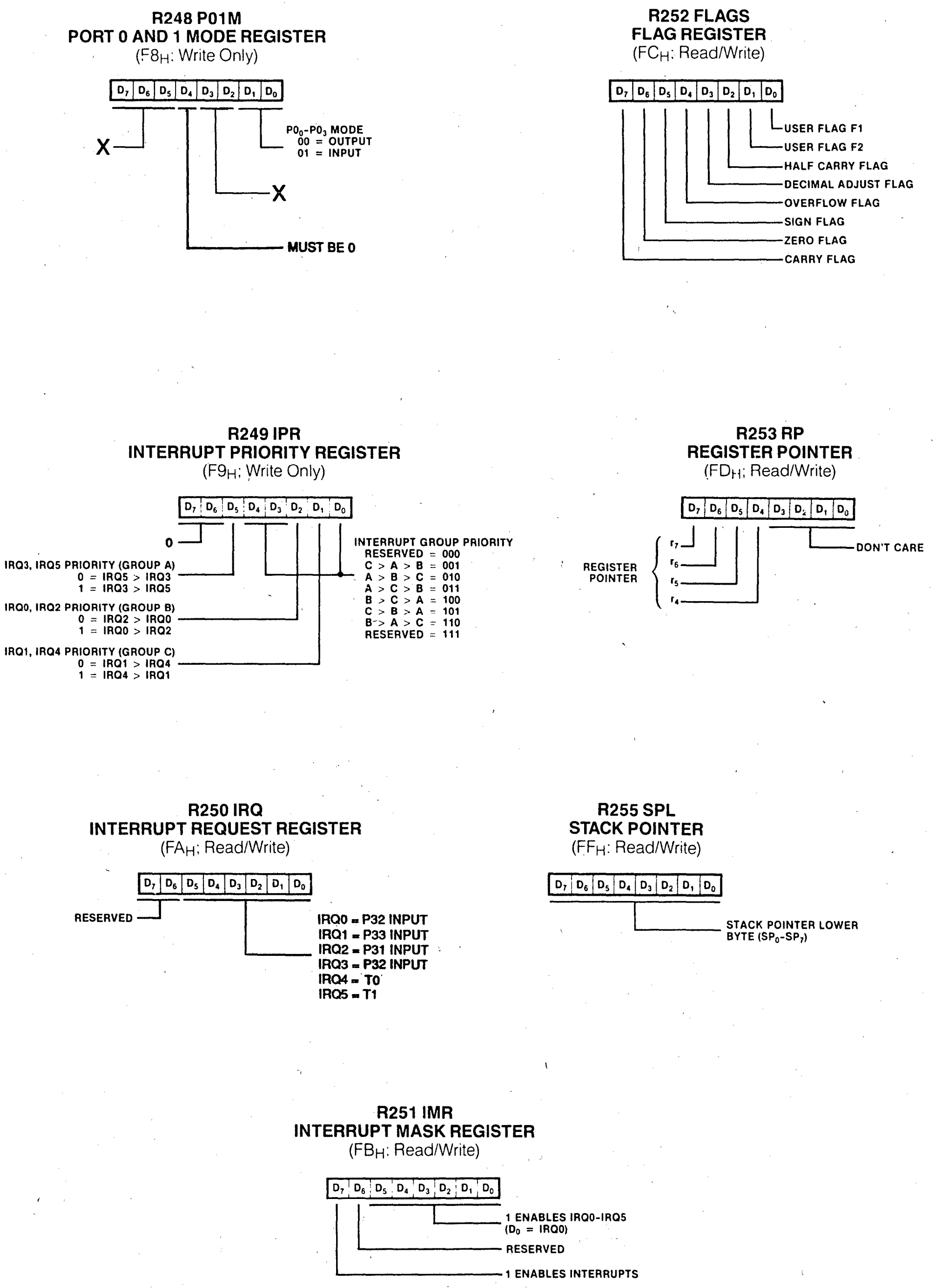


Figure 16 Control Registers (Continued)

ABSOLUTE MAXIMUM RATINGS

Voltages on all pins with respect
to GND . . . . – 0.3V to + 7.0V
Operating Ambient
Temperature . . . . .See Ordering Information
Storage Temperature . . . . – 65 °C to + 150 °C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin **(Figure 13)**.

Standard conditions are as follows:

- **+4.5 V <_ Vcc <_ +5.5 V**
- GND = 0V
- 0 °C ≤ T_A ≤ + 70 °C

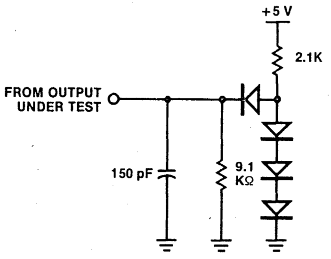


Figure 13 Test Load 1

DC CHARACTERISTICS V_{cc} = 5.0 V ± 10% 0° to 70° C

Symbol	Parameter	Min	Typ	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	V_{CC}-0.2		V_{CC}	V	Driven by external CG
V_{CL}	Clock Input Low Voltage	-0.3		V_{SS}+0.2	V	Driven by External CG
V_{IH}	Input High Voltage	V_{CC}-0.2		V_{CC}	V	
V_{IL}	Input Low Voltage	-0.3		V_{SS}+0.2	V	
V_{RH}	RESET Input High Voltage	V_{CC}-0.2		V_{CC}	V	
V_{RL}	RESET Input Low Voltage	-0.3		V_{SS}+0.2	V	
V_{OH}	Output High Voltage	V_{CC}-0.4			V	I_{OH} = -2.0mA
V_{OL1}	Output Low Voltage			0.4	V	I_{OL} = +4.0mA
V_{OL2}	Output Low Voltage			0.8	V	I_{OL} = +12mA, 3 pins max.
I_{IL}	Input Leakage	-10		10	uA	V_{IN} = 0V, V_{CC}
I_{OL}	Output Leakage	-10		10	uA	V_{IN} = 0V, V_{CC}
I_{IR}	RESET Input Current		-10	-50	uA	V_{CC} = 4.5 to 5.5V, V_{RL} = 0V, P27
I_{CC}	Supply Current			15	mA	All Output & I/O pins float
I_{CC1}	Standby Current			2	mA	HALT Mode V_{in} = 0V, V_{CC}
I_{CC2}	Standby Current			10	uA	STOP Mode V_{in} = 0V, V_{CC}

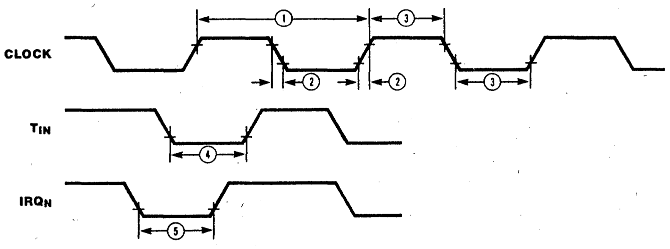


Figure 14. Additional Timing

AC CHARACTERISTICS

Number	Symbol	Parameter	Min	Max	Notes
1	TpC	Input Clock Period	125	100,000	1
2	TrC, TfC	Clock Input Rise and Fall Times		25	1
3	TwC	Input Clock Width	37		1
4	TwTinL	Timer Input Low Width	100		2
5	TwTinH	Timer Input High Width	3TpC		2
6	TpTin	Timer Input Period	8TpC		2
7	TrTin,TfTin	Timer Input Rise and Fall Times		100	2
8A	TwIL	Int. Resquest Input Low Time	100		2,4
9	TwIH	Int. Request Input High Time	3TpC		2,3

NOTES:

1. Clock timing references use V_{CC} for a logic "1" and V_{SS} for logic "0".
2. Timing references use V_{CC} for a logic "1" and V_{SS} for a logic "0".
3. Interup request via P31- P33
4. Interrupt request via P31-P33

*Units in nanoseconds (ns)

PRELIMINARY Z86C08 COMPARATOR SPECIFICATIONS

	CASE 1	CASE 2	CASE 3	CASE 4	CASE 5
Conditions Parameters	VDD=2.5V Temp=40C°	VDD=2.5V Temp=85C°	VDD=5.5V Temp=40C°	VDD=5.5V Temp=85C°	VDD=5.0V Temp=27C°
Offset Voltage (mv)	±50 (est)	±50 (est)	±50 (est)	±50 (est)	±25 (typ)
Open Loop Gain (db)	60 (min)	60 (min)	60 (min)	60 (min)	75 (typ)
CMRR (db)	60 (est)	60 (est)	60 (est)	60 (est)	70 (typ)
PSRR (db)	70 (est)	70 (est)	70 (est)	70 (est)	80 (typ)
Internal Delay Time (us)	15 (max)	15 (max)	1. (max)	1.0 (max)	0.1 (typ)
Overdrive (mv)	±300	±300	±300	±300	±300
CMR (+)	2.0 (max)	2.0 (max)	4.5 (max)	4.5 (max)	4.0 (max)
CMR (-)	0 (min)	0 (min)	0 (min)	0 (min)	0 (min)
I_{Bias} (ma)	0.1 (max)	0.1 (max)	1.0 (max)	1.0 (max)	0.2 (typ)
Power (mw)	0.25	0.25	5.5	4.125	1.25
Power Down	Yes	Yes	Yes	Yes	Yes

ORDERING INFORMATION

Z86C08 CMOS Microcontroller
Z86C0808PSC 8MHz
Z86C0812PSC 12MHz

Codes

First letter is for package; second letter is for temperature.

C	=	Ceramic DIP
P	=	Plastic DIP
L	=	Ceramic LCC
V	=	Plastic PCC

R	=	Protopack
T	=	Low Profile Protopack
DIP	=	Dual-In-Line Package
LCC	=	Leadless Chip Carrier
PCC	=	Plastic Chip Carrier (Leaded)

TEMPERATURE

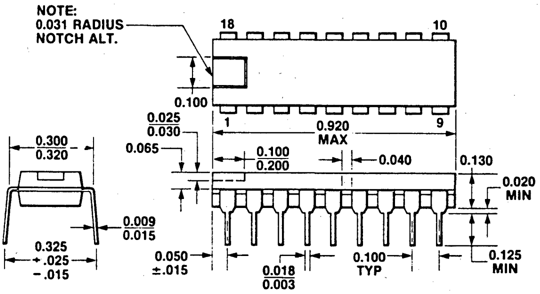
S	=	0°C to +70°C
E	=	-40°C to +85°C
M*	=	-55°C to +125°C

FLOW

B	=	883 Class B
J	=	JAN 38510 Class B

Example: PS is a plastic DIP, 0°C to +70°C.

PACKAGE DIMENSIONS



18-Pin Plastic Package

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4:

Zilog

ADVANCED INFORMATION
Product Specification

Z86C00/C10/C20 CMOS Z8® MCU

June 1987

FEATURES

- Complete microcomputer, 2K (86C00), 4K (86C10), or 8K (86C20) bytes of ROM, 124 bytes of RAM, and 22 I/O lines.
- 144—byte register file, including 124 general—purpose registers, four I/O port registers, and 14 status and control registers.
- Average instruction execution time of 1.5 us, maximum of 2.8 us.
- Vectored, priority interrupts for I/O and counter/timers.
- Two programmable 8—bit counter/timers, each with a 6—bit programmable prescaler.
- Register Pointer so that short, fast instructions can access any of nine working—register groups in 1.0 us.
- On—chip oscillator which accepts crystal, external clock drive, LC, ceramic resonator.
- Standby modes —— Halt and Stop.
- Single +5V power supply —— all pins TTL—compatible.
- 12 MHz.
- CMOS process.

GENERAL DESCRIPTION

Z86C10/C20 microcomputer (Figures 1 and 2) introduces a new level of sophistication to single-chip architecture. Compared to earlier single-chip microcomputers, the **Z86C10/C20** offers faster execution; more efficient use of memory; more sophisticated interrupt, input/output and bit-manipulation capabilities; and easier system expansion.

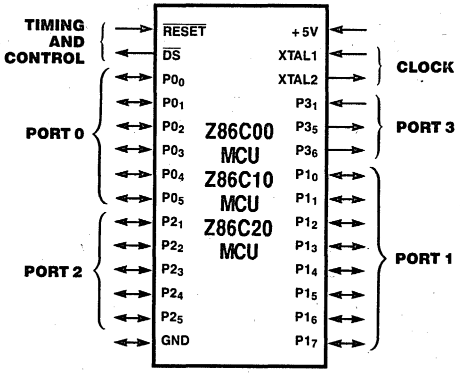


Figure 1. Pin Functions

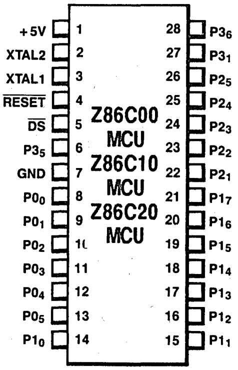


Figure 2. Pin Assignments

PIN DESCRIPTIONS

$\overline{DS}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each memory transfer.

$P0_0$-$P0_5$, $P1_0$-$P1_7$, $P2_1$-$P2_5$, $P3_1$, $P3_5$, $P3_6$. *I/O Port lines* (bidirectional, TTL-compatible). These 22 I/O lines are grouped in four ports that can be configured under program control for I/O.

$\overline{RESET}$. *Reset* (input, active Low). $\overline{RESET}$ initializes the MCU. When $\overline{RESET}$ is deactivated, program execution begins from internal program location $000C_H$.

XTAL1, XTAL2. *Crystal 1, Crystal 2* (time-base input and output). These pins connect a parallel-resonant crystal to the on-chip clock oscillator and buffer.

ARCHITECTURE

The MCU's architecture is characterized by a flexible I/O scheme, an efficient register and address space structure, and a number of ancillary features that are helpful in many applications. (Figure 3).

Microcomputer applications demand powerful I/O capabilities. The MCU fulfills this with 22 pins dedicated to input and output. These lines are grouped in four ports and are configurable under software control to provide timing, status signals, and parallel I/O.

Two basic internal address spaces are available to support this wide range of configurations: program memory and the register file. The 144-byte random-access register file is composed of 124 general-purpose registers, four I/O port registers, and 14 control and status registers.

To unburden the program from coping with real-time problems such as counting/timing, two counter/timers with a large number of user-selectable modes are offered on-chip.

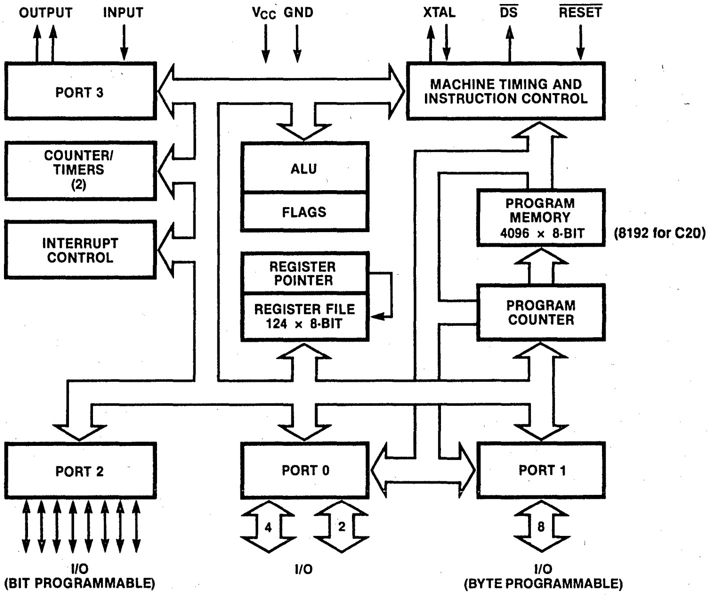


Figure 3. Functional Block Diagram

STANDBY MODE

The Z86C00/C10/C20's standby modes are:

- Stop
- Halt

The Stop instruction stops the internal clock and clock oscillation; the Halt instruction stops the internal clock but not clock oscillation.

A reset input releases the standby mode.

To complete an instruction prior to entering standby mode, use the instructions:

LD TMR, #00
NOP
STOP or HALT

ADDRESS SPACES

Program Memory. The 16-bit program counter addresses 4K **or 8K** bytes of program memory space as shown in Figure 4.

The first 12 bytes of program memory are reserved for the interrupt vectors. These locations contain three 16-bit vectors that correspond to the three available interrupts.

Register File. The 144-byte register file includes four I/O port registers (R_0-R_3), 124 general-purpose registers (R_4-R_{127}) and 15 control and status registers (R_{241}-R_{255}). These registers are assigned the address locations shown in Figure 5.

Instructions can access registers directly or indirectly with an 8-bit address field. The MCU also allows short 4-bit register addressing using the Register Pointer (one of the control registers). In the 4-bit mode, the register file is divided into nine working-register groups, each occupying 16 contiguous locations (Figure 6). The Register Pointer addresses the starting location of the active working-register group.

Stacks. An 8-bit Stack Pointer (R_{255}) is used for the internal stack that resides within the 124 general-purpose registers (R_4-R_{127}).

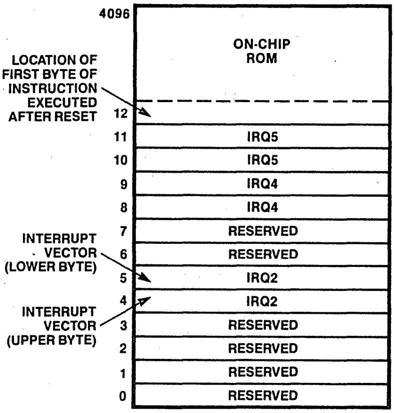

4096
ON-CHIP ROM
LOCATION OF FIRST BYTE OF INSTRUCTION EXECUTED AFTER RESET
12
11 IRQ5
10 IRQ5
9 IRQ4
8 IRQ4
7 RESERVED
6 RESERVED
5 IRQ2
4 IRQ2
3 RESERVED
2 RESERVED
1 RESERVED
0 RESERVED
INTERRUPT VECTOR (LOWER BYTE)
INTERRUPT VECTOR (UPPER BYTE)

Figure 4. Program Memory Map

LOCATION		IDENTIFIERS
255	STACK POINTER (BITS 7-0)	SPL
254	RESERVED	
253	REGISTER POINTER	RP
252	PROGRAM CONTROL FLAGS	FLAGS
251	INTERRUPT MASK REGISTER	IMR
250	INTERRUPT REQUEST REGISTER	IRQ
249	INTERRUPT PRIORITY REGISTER	IPR
248	PORTS 0-1 MODE	P01M
247	PORT 3 MODE	P3M
246	PORT 2 MODE	P2M
245	T0 PRESCALER	PRE0
244	TIMER/COUNTER 0	T0
243	T1 PRESCALER	PRE1
242	TIMER/COUNTER 1	T1
241	TIMER MODE	TMR
	NOT IMPLEMENTED	
127	GENERAL-PURPOSE REGISTERS	
4		
3	PORT 3	P3
2	PORT 2	P2
1	PORT 1	P1
0	PORT 0	P0

Figure 5. Register File

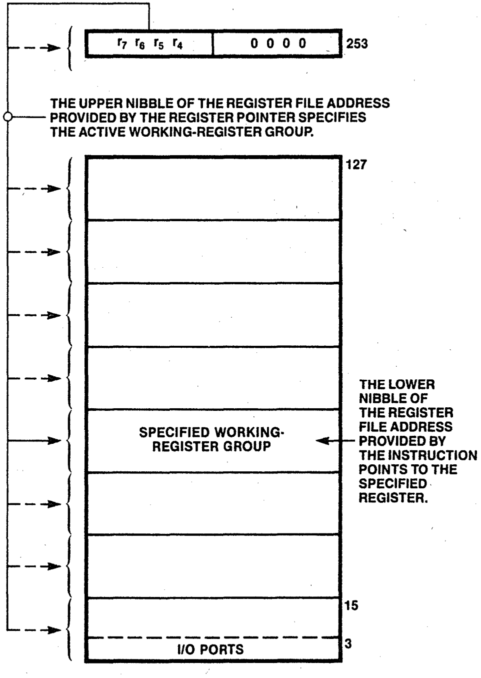

r7 r6 r5 r4
0 0 0 0
253
THE UPPER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE REGISTER POINTER SPECIFIES THE ACTIVE WORKING-REGISTER GROUP.
127
SPECIFIED WORKING-REGISTER GROUP
THE LOWER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE INSTRUCTION POINTS TO THE SPECIFIED REGISTER.
15
3
I/O PORTS

Figure 6. Register Pointer

COUNTER/TIMERS

The MCU contains two 8-bit programmable counter/timers (T_0 and T_1), each driven by its own 6-bit programmable prescaler. The T_1 prescaler can be driven by internal or external clock sources; however, the T_0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrements the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request—IRQ_4 (T_0) or IRQ_5 (T_1)—is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single-pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read any time without disturbing their value or count mode.

The clock source for T_1 is user-definable and can be the internal microprocessor clock divided by four, or an external signal input via Port 3. The Timer Mode register configures the external timer input as an external clock , a trigger input that can be retriggerable or non-retriggerable, or as a gate input for the internal clock. The counter/timers can be programmably cascaded by connecting the T_0 output to the input of T_1. Port 3 line $P3_6$ also serves as a timer output (T_{OUT}) through which T_0, T_1 or the internal clock can be output.

I/O PORTS

The MCU has 22 lines dedicated to input and output grouped in four ports. Under software control, the ports can be programmed to provide address outputs, timing, status signals, and parallel I/O. All ports have active pull-ups and pull-downs compatible with TTL loads.

Port 0 can be programmed as an I/O port.

Port 1 can be programmed as a byte I/O port.

Port 2 can be programmed independently as input or output and is always available for I/O operations. In addition, Port 2 can be configured to provide open-drain outputs.

Port 3 can be configured as I/O or control lines. $P3_1$ is a general purpose input or can be used for an external interrupt request signal (IRQ_2). $P3_5$ and $P3_6$ are general purpose outputs. $P3_6$ is also used for timer input (T_{IN}) and output (T_{OUT}) signals.

INTERRUPTS

The MCU allows three different interrupts from three sources, the Port 3 line $P3_1$ and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask register globally or individually enables or disables the three interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All interrupts are vectored. When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the Program Counter and status flags, and branches to the program memory vector locations reserved for that interrupt. This memory location and the next byte contain the 16-bit address of the interrupt service routine for that particular interrupt request.

Polled interrupt systems are also supported. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the Interrupt Request register polled to determine which of the interrupt requests needs service.

CLOCK

The on-chip oscillator has a high-gain parallel-resonant amplifier for connection to a crystal or to any suitable external clock source (XTAL1 = Input, XTAL2 = Output).

Crystal source is connected across XTAL1 and XTAL2 using the recommended capacitors (C1 ≤ 15 pf) from each pin to ground. The specifications are as follows:

- AT cut, parallel resonant
- **Fundamental type, 16 MHz maximum.**
- Series resistance, Rs ≤ 100 n

INSTRUCTION SET NOTATION

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "←". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

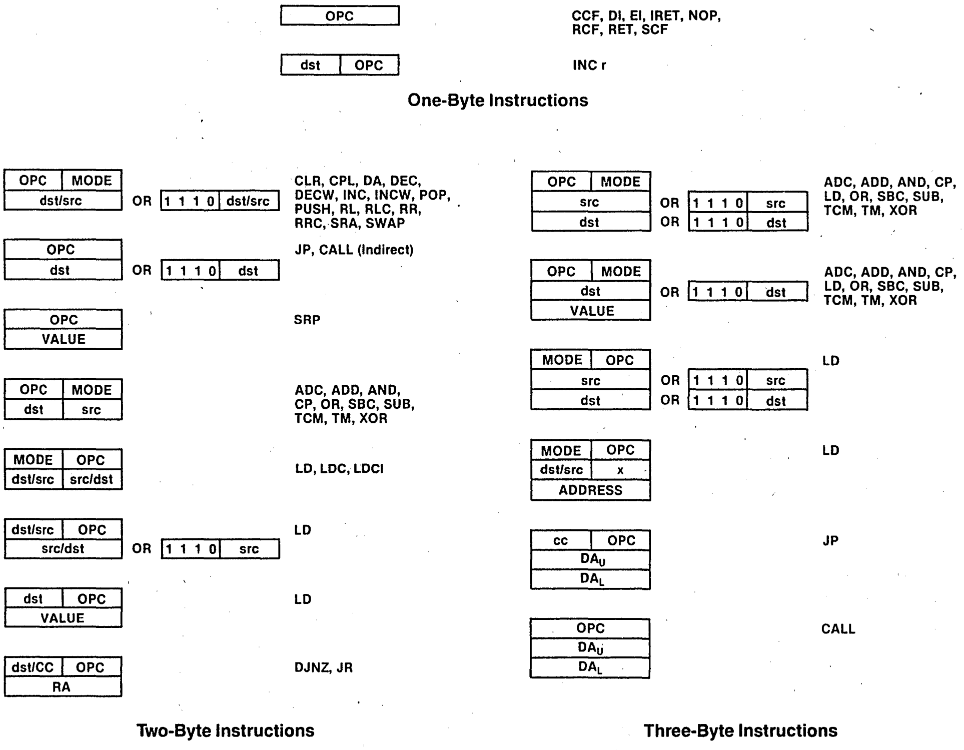
Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*****	Set or cleared according to operation
—	Unaffected
X	Undefined

CONDITION CODES

Value	Mnemonic	Meaning	Flags Set
1000		Always true	—
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	—

INSTRUCTION FORMATS



One-Byte Instructions

Two-Byte Instructions

Three-Byte Instructions

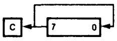
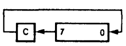
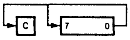
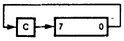
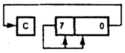
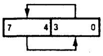
Figure 7. Instruction Formats

INSTRUCTION SUMMARY

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
ADC dst,src dst ← dst + src + C	(Note 1)		1□	*	*	*	*	0	*
ADD dst,src dst ← dst + src	(Note 1)		0□	*	*	*	*	0	*
AND dst,src dst ← dst AND src	(Note 1)		5□	—	*	*	0	—	—
CALL dst SP ← SP − 2 @SP ← PC; PC ← dst	DA IRR		D6 D4	—	—	—	—	—	—
CCF C ← NOT C			EF	*	—	—	—	—	—
CLR dst dst ← 0	R IR		B0 B1	—	—	—	—	—	—
COM dst dst ← NOT dst	R IR		60 61	—	*	*	0	—	—
CP dst,src dst − src	(Note 1)		A□	*	*	*	*	—	—
DA dst dst ← DA dst	R IR		40 41	*	*	*	X	—	—
DEC dst dst ← dst − 1	R IR		00 01	—	*	*	*	—	—
DECW dst dst ← dst − 1	RR IR		80 81	—	*	*	*	—	—
DI IMR (7) ← 0			8F	—	—	—	—	—	—
DJNZ r,dst r ← r − 1 if r ≠ 0 PC ← PC + dst Range: +127, −128	RA		rA r = 0 − F	—	—	—	—	—	—

INSTRUCTION SUMMARY (Continued)

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
EI			9F	—	—	—	—	—	—
IMR (7) ← 1									
HALT			7F						
INC dst	r		rE	—	*	*	*	—	—
dst ← dst + 1			r = 0 – F						
	R		20						
	IR		21						
INCW dst	RR		A0	—	*	*	*	—	—
dst ← dst + 1	IR		A1						
IRET			BF	*	*	*	*	*	*
FLAGS ← @SP; SP ← SP + 1									
PC ← @SP; SP ← SP + 2; IMR (7) ← 1									
JP cc,dst	DA		cD	—	—	—	—	—	—
if cc is true			c = 0 – F						
PC ← dst	IRR		30						
JR cc,dst	RA		cB	—	—	—	—	—	—
if cc is true,			c = 0 – F						
PC ← PC + dst									
Range: +127, –128									
LD dst,src	r	Im	rC	—	—	—	—	—	—
dst ← src	r	R	r8						
	R	r	r9						
			r = 0 – F						
	r	X	C7						
	X	r	D7						
	r	Ir	E3						
	Ir	r	F3						
	R	R	E4						
	R	IR	E5						
	R	IM	E6						
	IR	IM	E7						
	IR	R	F5						
LDC dst,src	r	Irr	C2	—	—	—	—	—	—
dst ← src	Irr	r	D2						
LDCI dst,src	Ir	Irr	C3	—	—	—	—	—	—
dst ← src	Irr	Ir	D3						
r ← r + 1; rr ← rr + 1									
LDE dst,src	r	Irr	82	—	—	—	—	—	—
dst ← src	Irr	r	92						
LDEI dst,src	Ir	Irr	83	—	—	—	—	—	—
dst ← src	Irr	Ir	93						
r ← r + 1; rr ← rr + 1									
NOP			FF	—	—	—	—	—	—
OR dst,src	(Note 1)		4□	—	*	*	0	—	—
dst ← dst OR src									
POP dst	R		50	—	—	—	—	—	—
dst ← @SP;	IR		51						
SP ← SP + 1									
PUSH src		R	70	—	—	—	—	—	—
SP ← SP – 1; @SP ← src		IR	71						

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
RCF			CF	0	—	—	—	—	—
C ← 0									
RET			AF	—	—	—	—	—	—
PC ← @SP; SP ← SP + 2									
RL dst 	R		90	*	*	*	*	—	—
	IR		91						
RLC dst 	R		10	*	*	*	*	—	—
	IR		11						
RR dst 	R		E0	*	*	*	*	—	—
	IR		E1						
RRC dst 	R		C0	*	*	*	*	—	—
	IR		C1						
SBC dst,src	(Note 1)		3□	*	*	*	*	1	*
dst ← dst ← src ← C									
SCF			DF	1	—	—	—	—	—
C ← 1									
SRA dst 	R		D0	*	*	*	0	—	—
	IR		D1						
SRP src		Im	31	—	—	—	—	—	—
RP ← src									
STOP			6F						
SUB dst,src	(Note 1)		2□	*	*	*	*	1	*
dst ← dst ← src									
SWAP dst 	R		F0	X	*	*	X	—	—
	IR		F1						
TCM dst,src	(Note 1)		6□	—	*	*	0	—	—
(NOT dst) AND src									
TM dst,src	(Note 1)		7□	—	*	*	0	—	—
dst AND src									
XOR dst,src	(Note 1)		B□	—	*	*	0	—	—
dst ← dst XOR src									

NOTE: These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the left of the applicable addressing mode pair.

For example, the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

Addr Mode		Lower Opcode Nibble
dst	src	
r	r	2
r	Ir	3
R	R	4
R	IR	5
R	IM	6
IR	IM	7

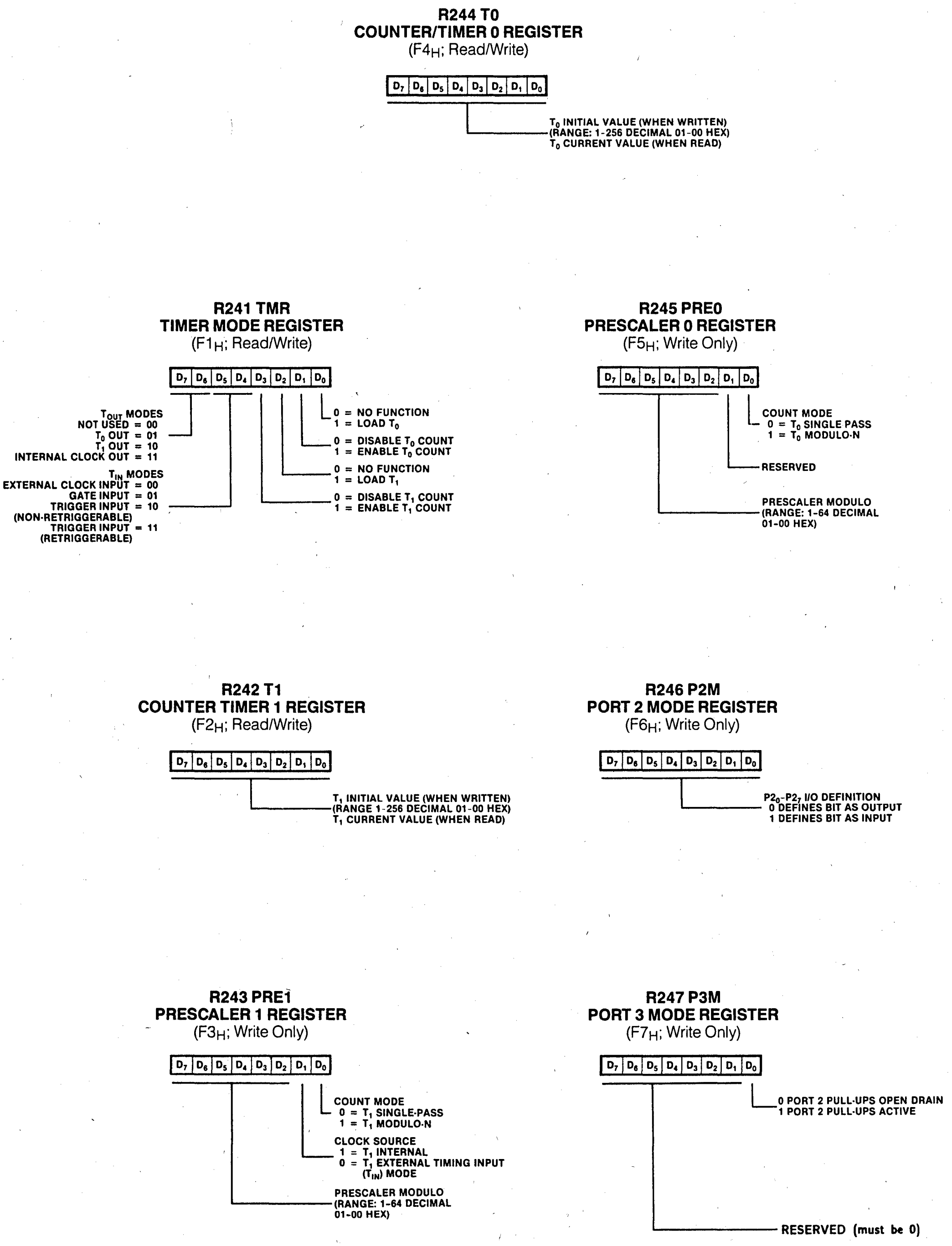
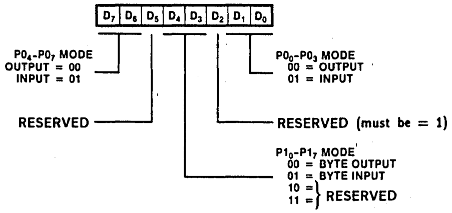
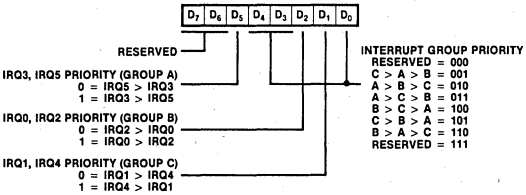


Figure 11. Control Registers

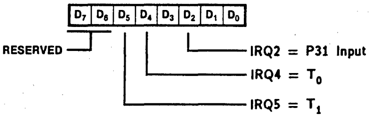
R248 P01M
PORT 0 AND 1 MODE REGISTER
($F8_H$; Write Only)



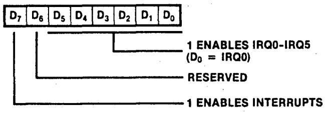
R249 IPR
INTERRUPT PRIORITY REGISTER
($F9_H$; Write Only)



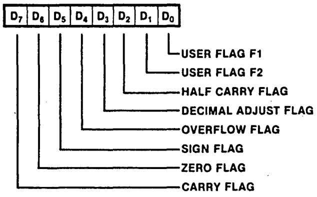
R250 IRQ
INTERRUPT REQUEST REGISTER
(FA_H; Read/Write)



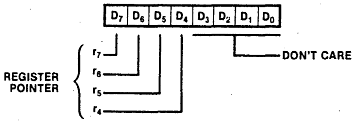
R251 IMR
INTERRUPT MASK REGISTER
(FB_H; Read/Write)



R252 FLAGS
FLAG REGISTER
(FC_H; Read/Write)



R253 RP
REGISTER POINTER
(FD_H; Read/Write)



R255 SPL
STACK POINTER
(FF_H; Read/Write)

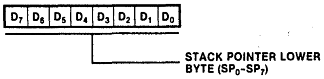
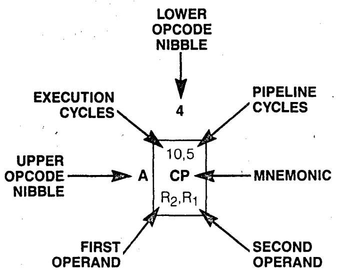


Figure 11. Control Registers (Continued)

OPCODE MAP

Upper Nibble (Hex)	Lower Nibble (Hex)															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6.5 **DEC** R_1	6.5 **DEC** IR_1	6.5 **ADD** r_1,r_2	6.5 **ADD** r_1,Ir_2	10.5 **ADD** R_2,R_1	10.5 **ADD** IR_2,R_1	10.5 **ADD** R_1,IM	10.5 **ADD** IR_1,IM	6.5 **LD** r_1,R_2	6.5 **LD** r_2,R_1	12/10.5 **DJNZ** r_1,RA	12/10.0 **JR** cc,RA	6.5 **LD** r_1,IM	12/10.0 **JP** cc,DA	6.5 **INC** r1	
1	6.5 **RLC** R_1	6.5 **RLC** IR_1	6.5 **ADC** r_1,r_2	6.5 **ADC** r_1,Ir_2	10.5 **ADC** R_2,R_1	10.5 **ADC** IR_2,R_1	10.5 **ADC** R_1,IM	10.5 **ADC** IR_1,IM								
2	6.5 **INC** R_1	6.5 **INC** IR_1	6.5 **SUB** r_1,r_2	6.5 **SUB** r_1,Ir_2	10.5 **SUB** R_2,R_1	10.5 **SUB** IR_2,R_1	10.5 **SUB** R_1,IM	10.5 **SUB** IR_1,IM								
3	8.0 **JP** IRR_1	6.1 **SRP** IM	6.5 **SBC** r_1,r_2	6.5 **SBC** r_1,Ir_2	10.5 **SBC** R_2,R_1	10.5 **SBC** IR_2,R_1	10.5 **SBC** R_1,IM	10.5 **SBC** IR_1,IM								
4	8.5 **DA** R_1	8.5 **DA** IR_1	6.5 **OR** r_1,r_2	6.5 **OR** r_1,Ir_2	10.5 **OR** R_2,R_1	10.5 **OR** IR_2,R_1	10.5 **OR** R_1,IM	10.5 **OR** IR_1,IM								
5	10.5 **POP** R_1	10.5 **POP** IR_1	6.5 **AND** r_1,r_2	6.5 **AND** r_1,Ir_2	10.5 **AND** R_2,R_1	10.5 **AND** IR_2,R_1	10.5 **AND** R_1,IM	10.5 **AND** IR_1,IM								
6	6.5 **COM** R_1	6.5 **COM** IR_1	6.5 **TCM** r_1,r_2	6.5 **TCM** r_1,Ir_2	10.5 **TCM** R_2,R_1	10.5 **TCM** IR_2,R_1	10.5 **TCM** R_1,IM	10.5 **TCM** IR_1,IM								6,0 **STOP**
7	10/12.1 **PUSH** R_2	12/14.1 **PUSH** IR_2	6.5 **TM** r_1,r_2	6.5 **TM** r_1,Ir_2	10.5 **TM** R_2,R_1	10.5 **TM** IR_2,R_1	10.5 **TM** R_1,IM	10.5 **TM** IR_1,IM								7,0 **HALT**
8	10.5 **DECW** RR_1	10.5 **DECW** IR_1														6.1 **DI**
9	6.5 **RL** R_1	6.5 **RL** IR_1														6.1 **EI**
A	10.5 **INCW** RR_1	10.5 **INCW** IR_1	6.5 **CP** r_1,r_2	6.5 **CP** r_1,Ir_2	10.5 **CP** R_2,R_1	10.5 **CP** IR_2,R_1	10.5 **CP** R_1,IM	10.5 **CP** IR_1,IM								14.0 **RET**
B	6.5 **CLR** R_1	6.5 **CLR** IR_1	6.5 **XOR** r_1,r_2	6.5 **XOR** r_1,Ir_2	10.5 **XOR** R_2,R_1	10.5 **XOR** IR_2,R_1	10.5 **XOR** R_1,IM	10.5 **XOR** IR_1,IM								16.0 **IRET**
C	6.5 **RRC** R_1	6.5 **RRC** IR_1	12.0 **LDC** r_1,Irr_2	18.0 **LDCI** Ir_1,Irr_2				10.5 **LD** r_1,x,R_2								6.5 **RCF**
D	6.5 **SRA** R_1	6.5 **SRA** IR_1	12.0 **LDC** r_2,Irr_1	18.0 **LDCI** Ir_2,Irr_1	20.0 **CALL*** IRR_1		20.0 **CALL** DA	10.5 **LD** r_2,x,R_1								6.5 **SCF**
E	6.5 **RR** R_1	6.5 **RR** IR_1		6.5 **LD** r_1,IR_2	10.5 **LD** R_2,R_1	10.5 **LD** IR_2,R_1	10.5 **LD** R_1,IM	10.5 **LD** IR_1,IM								6.5 **CCF**
F	8.5 **SWAP** R_1	8.5 **SWAP** IR_1		6.5 **LD** Ir_1,r_2		10.5 **LD** R_2,IR_1										6.0 **NOP**
Bytes per Instruction	2				3				2					3	1	



Legend:
R = 8-bit address
r = 4-bit address
R_1 or r_1 = Dst address
R_2 or r_2 = Src address

Sequence:
Opcode, First Operand, Second Operand

NOTE: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

ABSOLUTE MAXIMUM RATINGS

Voltages on all pins with respect
to GND . . . . −0.3V to +7.0V
Operating Ambient
Temperature . . . . .See Ordering Information
Storage Temperature . . . . −65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin.

Standard conditions are as follows:

- **+4.5 ≤ Vcc ≤ +5.5**
- GND = 0V
- 0°C ⩽ T_A ⩽ +70°C

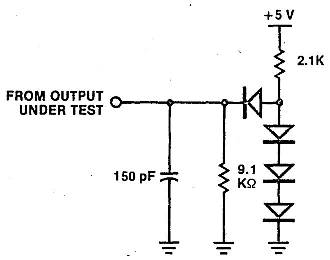


Figure 12. Test Load 1

DC CHARACTERISTICS

Symbol	Parameter	Min	Typ	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8		V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	−0.3		0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{IL}	Input Low Voltage	−0.3		0.8	V	
V_{RH}	Reset Input High Voltage	3.8		V_{CC}	V	
V_{RL}	Reset Input Low Voltage	−0.3		0.8	V	
V_{OH}	Output High Voltage	2.4			V	I_{OH} = −250 μA
V_{OH}	**Output High Voltage**	**V_{CC} -100 mV**			**V**	**I_{OH} = -100μA**
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = +2.0 mA
I_{IL}	Input Leakage	−10		10	μA	0V ⩽ V_{IN} ⩽ + 5.25V
I_{OL}	Output Leakage	−10		10	μA	0V ⩽ V_{IN} ⩽ + 5.25V
I_{IR}	Reset Input Current			−50	μA	V_{CC} = +5.25V, V_{RL} = 0V
I_{CC}	Supply Current			2	mA	All outputs and I/O pins floating
I_{CC1}	Standby Current		**5**		mA	Halt Mode
I_{CC2}	Standby Current			10	μA	Stop Mode

NOTE:

Icc2 low power requires loading TMR (%F1) with any value prior to stop execution.
Use sequence:

```
LD TMR, #%00.
NOP
STOP
```

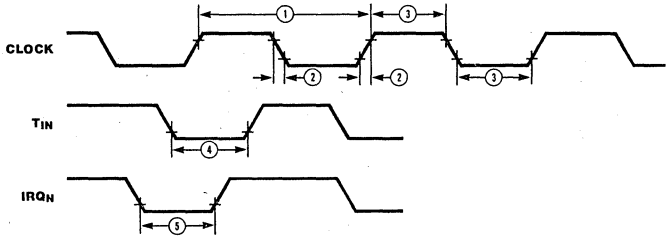


Figure 14. Additional Timing

AC CHARACTERISTICS

Additional Timing Table

			Z86C10		
Number	Symbol	Parameter	Min	Max	Notes*
1	TpC	Input Clock Period	**83**	**100,000**	1
2	TrC,TfC	Clock Input Rise and Fall Times		**15**	1
3	TwC	Input Clock Width	**70**		1
4	TwTinL	Timer Input Low Width	**70**		2
5	TwIL	Interrupt Request Input Low Time	**70**		2,3

NOTES:
1. Clock timing references use 3.8V for a logic "1" and 0.8V for a logic "0".
2. Timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
3. Interrupt request via Port 3.
* Units in nanoseconds (ns).

Zilog **Product Specification**

Z86C11 CMOS
Z8® 4K ROM MCU

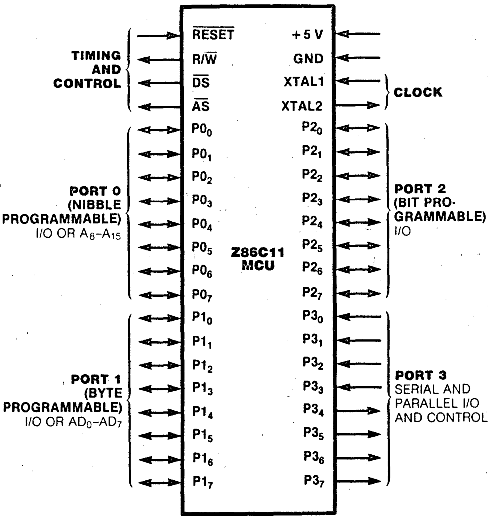
June 1987

FEATURES

- Complete microcomputer, 4K bytes of ROM, **256** bytes of RAM, 32 I/O lines, and up to 60K bytes addressable external space each for program and data memory.
- **256**-byte register file, including **236** general-purpose registers, four I/O port registers, and 16 status and control registers.
- Vectored, priority interrupts for I/O, counter/timers, and UART.
- Full-duplex UART and two programmable 8-bit counter/timers, each with a 6-bit programmable prescaler.
- Register Pointer so that short, fast instructions can access any of **16** working-register groups in 1.5 μs.
- On-chip oscillator which accepts crystal or external clock drive.
- Standby modes—Halt and Stop
- Single +5V power supply—all pins TTL-compatible.
- 12 MHz, 16 MHz
- CMOS process

GENERAL DESCRIPTION

The Z86C11 microcomputer (Figures 1 and 2) introduces a new level of sophistication to single-chip architecture. Compared to earlier single-chip microcomputers, the Z86C11 offers faster execution; more efficient use of memory; more sophisticated interrupt, input/output and bit-manipulation capabilities; and easier system expansion.


TIMING AND CONTROL
RESET
R/W
DS
AS
+5 V
GND
XTAL1
XTAL2
CLOCK
PORT 0 (NIBBLE PROGRAMMABLE) I/O OR A_8–A_{15}
$P0_0$ $P0_1$ $P0_2$ $P0_3$ $P0_4$ $P0_5$ $P0_6$ $P0_7$
PORT 1 (BYTE PROGRAMMABLE) I/O OR AD_0–AD_7
$P1_0$ $P1_1$ $P1_2$ $P1_3$ $P1_4$ $P1_5$ $P1_6$ $P1_7$
Z86C11 MCU
$P2_0$ $P2_1$ $P2_2$ $P2_3$ $P2_4$ $P2_5$ $P2_6$ $P2_7$
PORT 2 (BIT PROGRAMMABLE) I/O
$P3_0$ $P3_1$ $P3_2$ $P3_3$ $P3_4$ $P3_5$ $P3_6$ $P3_7$
PORT 3 SERIAL AND PARALLEL I/O AND CONTROL

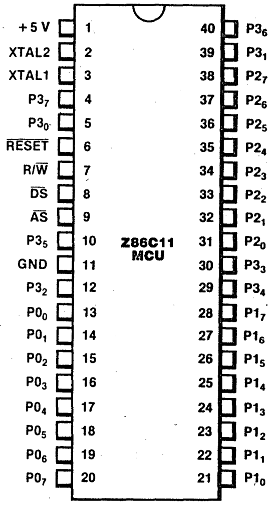


Figure 2. 40-pin Dual-In-Line Package (DIP), Pin Assignments

Under program control, the Z86C11 can be tailored to the needs of its user. It can be configured as a stand-alone microcomputer with 4K bytes of internal ROM, a traditional microprocessor that manages up to 120K bytes of external memory, or a parallel-processing element in a system with other processors and peripheral controllers linked by the Z-BUS® bus. In all configurations, a large number of pins remain available for I/O.

FIELD PROGRAMMABLE VERSION

The Z86E11 is a pin compatible "one time programmable" version of the Z86C11. The Z86C11 contains 4K bytes of EPROM memory in place of the 4K bytes of masked ROM in the Z86C11. The Z86E11 also contains a programmable memory protect feature to provide program security by disabling all external accesses to the internal EPROM array. This is preliminary information, and is subject to change.

ARCHITECTURE

Z86C11 architecture is characterized by a flexible I/O scheme, an efficient register and address space structure and a number of ancillary features that are helpful in many applications.

Microcomputer applications demand powerful I/O capabilities. The Z86C11 fulfills this with 32 pins dedicated to input and output. These lines are grouped into four ports of eight lines each and are configurable under software control to provide timing, status signals, serial or parallel I/O with or without handshake, and an address/data bus for interfacing external memory.

Because the multiplexed address/data bus is merged with the I/O-oriented ports, the Z86C11 can assume many different memory and I/O configurations. These configurations range from a self-contained microcomputer to a microprocessor that can address 120K bytes of external memory (Figure 3).

Three basic address spaces are available to support this wide range of configurations: program memory (internal and external), data memory (external) and the register file (internal). The 256-byte random-access register file is composed of 236 general-purpose registers, four I/O port registers, and 16 control and status registers.

To unburden the program from coping with real-time problems such as serial data communication and counting/timing, an asynchronous receiver/transmitter (UART) and two counter/timers with a large number of user-selectable modes are offered on-chip. Hardware support for the UART is minimized because one of the on-chip timers supplies the bit rate.

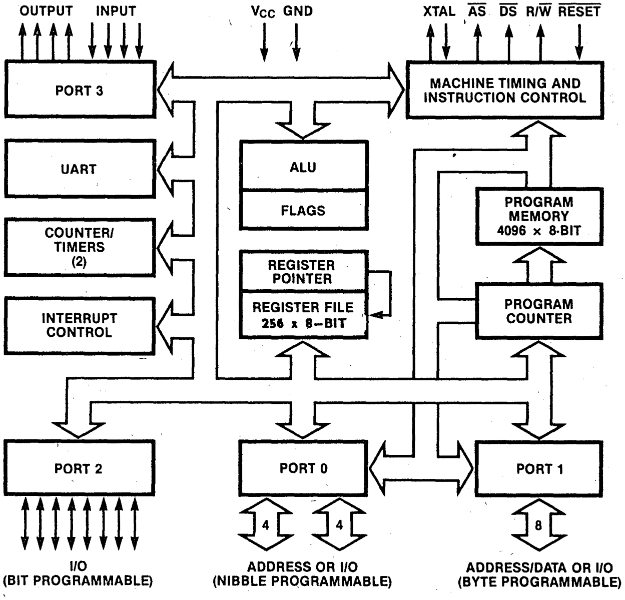

OUTPUT
INPUT
Vcc
GND
XTAL
AS
DS
R/W
RESET
PORT 3
MACHINE TIMING AND INSTRUCTION CONTROL
UART
ALU
FLAGS
PROGRAM MEMORY 4096 × 8-BIT
COUNTER/ TIMERS (2)
REGISTER POINTER
REGISTER FILE 256 x 8-BIT
PROGRAM COUNTER
INTERRUPT CONTROL
PORT 2
PORT 0
PORT 1
4
4
8
I/O (BIT PROGRAMMABLE)
ADDRESS OR I/O (NIBBLE PROGRAMMABLE)
ADDRESS/DATA OR I/O (BYTE PROGRAMMABLE)

Figure 3. Functional Block Diagram

STANDBY MODE

The Z86C11's standby modes are:

- □ Stop
- □ Halt

The Stop instruction stops the internal clock and clock oscillation; the Halt instruction stops the internal clock but not clock oscillation.

A reset input releases the standby mode.

POWER DOWN INSTRUCTIONS

The Z86C91 has two instructions to reduce power consumption during standby operation. HALT turns off the processor and UART while the counter/timers and external interrupts IRQ0, IRQ1, and IRQ2 remain active.

When an interrupt occurs the processor resumes execution after servicing the interrupt. STOP turns off the clock to the entire Z86C91 and reduces the standby current to 10 microamps. The stop mode is terminated by reset, which causes the processor to restart the application program at address 12.

To complete an instruction prior to entering standby mode, use the instructions:

```
LD TMR, #00
NOP
STOP or HALT
```

PIN DESCRIPTION

$\overline{\text{AS}}$. *Address Strobe* (output, active Low). Address Strobe is pulsed once at the beginning of each machine cycle. Addresses output via Port 1 for all external program or data memory transfers are valid at the trailing edge of $\overline{\text{AS}}$. Under program control, $\overline{\text{AS}}$ can be placed in the high-impedance state along with Ports 0 and 1, Data Strobe and Read/Write.

$\overline{\text{DS}}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each external memory transfer.

$P0_0-P0_7$, $P1_0-P1_7$, $P2_0-P2_7$, $P3_0-P3_7$. *I/O Port Lines* **(input/outputs, TTL–compatible). These 32 lines are divided into four 8–bit I/O ports that can be configured under program control for I/O or external memory interface (Figure 3).**

$\overline{\text{RESET}}$. *Reset* (input, active Low). $\overline{\text{RESET}}$ initializes the Z86C11. When $\overline{\text{RESET}}$ is deactivated, program execution begins from internal program location $000C_H$.

R/$\overline{\text{W}}$. *Read/Write* (output). R/$\overline{\text{W}}$ is Low when the Z86C11 is writing to external program or data memory.

XTAL1, XTAL2. Crystal 1, Crystal 2 (time–base input and output). These pins connect a parallel–resonant crystal (12 MHz maximum) or an external single–phase clock (12 MHz maximum) to the on–chip clock oscillator and buffer.

ADDRESS SPACE

Program Memory. The 16-bit program counter addresses 64K bytes of program memory space. Program memory can be located in two areas: one internal and the other external (Figure 4). The first 4096 bytes consist of on-chip mask-programmed ROM. At addresses 4096 and greater, the Z86C11 executes external program memory fetches.

The first 12 bytes of program memory are reserved for the interrupt vectors. These locations contain six 16-bit vectors that correspond to the six available interrupts.

Data Memory. The Z86C11 can address 60K bytes of external data memory beginning at location 4096 (Figure 5). External data memory may be included with or separated from the external program memory space. $\overline{\text{DM}}$, an optional I/O function that can be programmed to appear on pin $P3_4$, is used to distinguish between data and program memory space.

Register File. The **256**-byte register file includes four I/O port registers (R0-R3), **236** general-purpose registers (R4-R **239**) and 16 control and status registers (R240-R255). These registers are assigned the address locations shown in Figure 6.

Z86C11 instructions can access registers directly or indirectly with an 8–bit address field. The Z86C11 also allows short 4–bit register addressing using the Register Pointer (one of the control registers). In the 4–bit mode, the register file is divided into 16 working register groups, each occupying 16 contiguous locations (Figure 6). The Register Pointer addresses the starting location of the active working–register group (Figure 7).

Note: Register Bank E0-EF can only be accessed through working register and indirect addressing modes.

Stacks. Either the internal register file or the external data memory can be used for the stack. A 16-bit Stack Pointer (R254 and R255) is used for the external stack, which can reside anywhere in data memory between locations 4096 and 65535. An 8-bit Stack Pointer (R255) is used for the internal stack that resides within the 124 general-purpose registers (R4-R127).

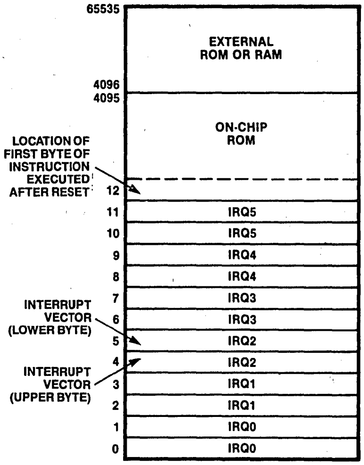


Figure 4. Program Memory Map

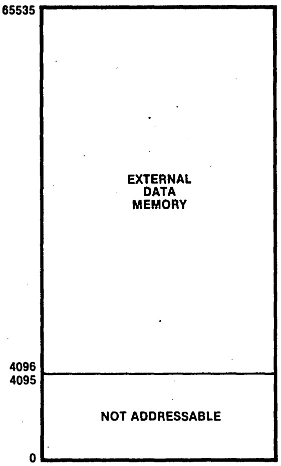
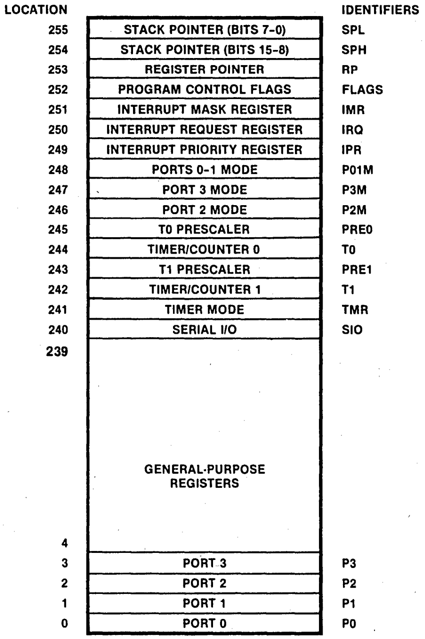


Figure 5. Data Memory Map



LOCATION		IDENTIFIERS
255	STACK POINTER (BITS 7-0)	SPL
254	STACK POINTER (BITS 15-8)	SPH
253	REGISTER POINTER	RP
252	PROGRAM CONTROL FLAGS	FLAGS
251	INTERRUPT MASK REGISTER	IMR
250	INTERRUPT REQUEST REGISTER	IRQ
249	INTERRUPT PRIORITY REGISTER	IPR
248	PORTS 0-1 MODE	P01M
247	PORT 3 MODE	P3M
246	PORT 2 MODE	P2M
245	T0 PRESCALER	PRE0
244	TIMER/COUNTER 0	T0
243	T1 PRESCALER	PRE1
242	TIMER/COUNTER 1	T1
241	TIMER MODE	TMR
240	SERIAL I/O	SIO
239 4	GENERAL-PURPOSE REGISTERS	
3	PORT 3	P3
2	PORT 2	P2
1	PORT 1	P1
0	PORT 0	P0

Figure 6. The Register File

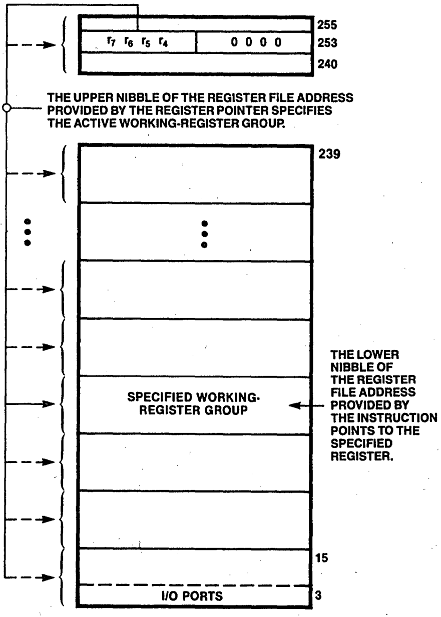

255
r_7 r_6 r_5 r_4
0 0 0 0
253
240
THE UPPER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE REGISTER POINTER SPECIFIES THE ACTIVE WORKING-REGISTER GROUP.
239
SPECIFIED WORKING-REGISTER GROUP
THE LOWER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE INSTRUCTION POINTS TO THE SPECIFIED REGISTER.
15
I/O PORTS
3

Figure 7. The Register Pointer

SERIAL INPUT/OUTPUT

Port 3 lines $P3_0$ and $P3_7$ can be programmed as serial I/O lines for full-duplex serial asynchronous receiver/transmitter operation. The bit rate is controlled by Counter/Timer 0, with a maximum rate of 62.5K bits/second for 8 MHz.

The Z86C11 automatically adds a start bit and two stop bits to transmitted data (Figure 8). Odd parity is also available as an option. Eight data bits are always transmitted, regardless of parity selection. If parity is enabled, the eighth bit is the odd parity bit. An interrupt request (IRQ_4) is generated on all transmitted characters.

Received data must have a start bit, eight data bits and at least one stop bit. If parity is on, bit 7 of the received data is replaced by a parity error flag. Received characters generate the IRQ_3 interrupt request.

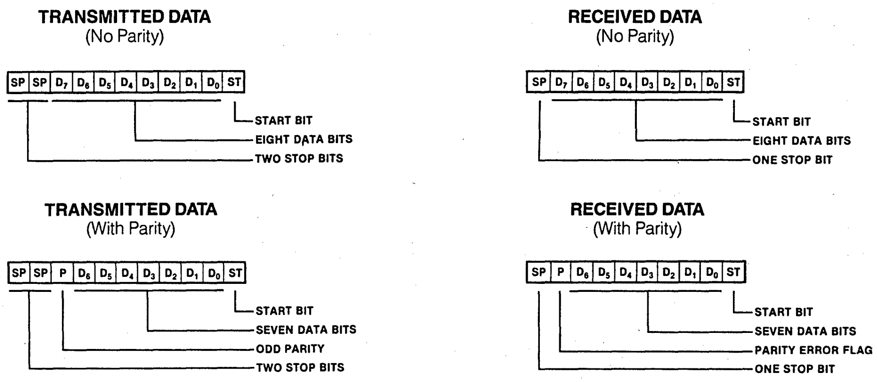


Figure 8. Serial Data Formats

COUNTER/TIMERS

The Z86C11 contains two 8-bit programmable counter/timers (T_0 and T_1), each driven by its own 6-bit programmable prescaler. The T_1 prescaler can be driven by internal or external clock sources; however, the T_0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrements the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request—IRQ_4 (T_0) or IRQ_5 (T_1)—is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single-pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read any time without disturbing their value or count mode.

The clock source for T_1 is user–definable and can be the internal microprocessor clock divided by four, or an external signal input via Port 3. The Timer Mode register configures the external timer input as an external clock (1 MHz maximum), a trigger input that can be retriggerable or non–retriggerable, or as a gate input for the internal clock. The counter/timers can be programmably cascaded by connecting the T_0 output to the input of T_1. Port 3 line $P3_6$ also serves as a timer output (T_{OUT}) through which T_0, T_1 or the internal clock can be output.

I/O PORTS

The Z86C11 has 32 lines dedicated to input and output. These lines are grouped into four ports of eight lines each and are configurable as input, output or address/data. Under software control, the ports can be programmed to provide address outputs, timing, status signals, serial I/O, and parallel I/O with or without handshake. All ports have active pull-ups and pull-downs compatible with TTL loads.

Port 1 can be programmed as a byte I/O port or as an address/data port for interfacing external memory. When used as an I/O port, Port 1 may be placed under handshake control. In this configuration, Port 3 lines $P3_3$ and $P3_4$ are used as the handshake controls RDY_1 and $\overline{DAV}_1$ (Ready and Data Available).

Memory locations greater than 4096 are referenced through Port 1. To interface external memory, Port 1 must be programmed for the multiplexed Address/Data mode. If more than 256 external locations are required, Port 0 must output the additional lines.

Port 1 can be placed in the high-impedance state along with Port 0, $\overline{AS}$, $\overline{DS}$ and $R/\overline{W}$, allowing the Z86C11 to share common resources in multiprocessor and DMA applications. Data transfers can be controlled by assigning $P3_3$ as a Bus Acknowledge input, and $P3_4$ as a Bus Request output.

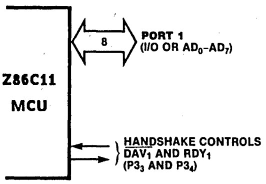


Figure 9a. Port 1

Port 0 can be programmed as a nibble I/O port, or as an address port for interfacing external memory. When used as an I/O port, Port 0 may be placed under handshake control. In this configuration, Port 3 lines $P3_2$ and $P3_5$ are used as the handshake controls $\overline{DAV}_0$ and RDY_0. Handshake signal assignment is dictated by the I/O direction of the upper nibble $P0_4$-$P0_7$.

For external memory references, Port 0 can provide address bits A_8-A_{11} (lower nibble) or A_8-A_{15} (lower and upper nibble) depending on the required address space. If the address range requires 12 bits or less, the upper nibble of Port 0 can be programmed independently as I/O while the lower nibble is used for addressing. When Port 0 nibbles are defined as address bits, they can be set to the high-impedance state along with Port 1 and the control signals $\overline{AS}$, $\overline{DS}$ and $R/\overline{W}$.

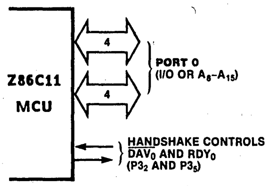


Figure 9b. Port 0

Port 2 bits can be programmed independently as input or output. This port is always available for I/O operations. In addition, Port 2 can be configured to provide open-drain outputs.

Like Ports 0 and 1, Port 2 may also be placed under handshake control. In this configuration, Port 3 lines $P3_1$ and $P3_6$ are used as the handshake controls lines $\overline{DAV}_2$ and RDY_2. The handshake signal assignment for Port 3 lines $P3_1$ and $P3_6$ is dictated by the direction (input or output) assigned to bit 7 of Port 2.

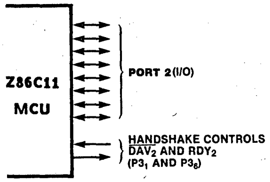


Figure 9c. Port 2

Port 3 lines can be configured as I/O or control lines. In either case, the direction of the eight lines is fixed as four input ($P3_0$-$P3_3$) and four output ($P3_4$-$P3_7$). For serial I/O, lines $P3_0$ and $P3_7$ are programmed as serial in and serial out respectively.

Port 3 can also provide the following control functions: handshake for Ports 0, 1 and 2 ($\overline{DAV}$ and RDY); four external interrupt request signals (IRQ_0-IRQ_3); timer input and output signals (T_{IN} and T_{OUT}) and Data Memory Select ($\overline{DM}$).

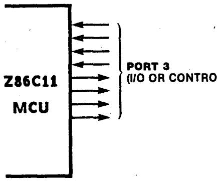


Figure 9d. Port 3

INTERRUPTS

The Z86C11 allows six different interrupts from eight sources: the four Port 3 lines $P3_0$-$P3_3$, Serial In, Serial Out, and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask register globally or individually enables or disables the six interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All Z86C11 interrupts are vectored. When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the Program Counter and status flags, and branches to the program memory vector location reserved for that interrupt. This memory location and the next byte contain the 16-bit address of the interrupt service routine for that particular interrupt request.

Polled interrupt systems are also supported. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the Interrupt Request register polled to determine which of the interrupt requests needs service.

CLOCK

The on-chip oscillator has a high-gain, parallel-resonant amplifier for connection to a crystal or to any suitable external clock source (XTAL1 = Input, XTAL2 = Output).

The crystal source is connected across XTAL1 and XTAL2, using the recommended capacitors ($C_1 \leqslant 15$ pf) from each pin to ground. The specifications for the crystal are as follows:

- AT cut, parallel resonant
- **Fundamental type, 12 MHz maximum**
- Series resistance, $R_S \leqslant 100\ \Omega$

INSTRUCTION SET NOTATION

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "←". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

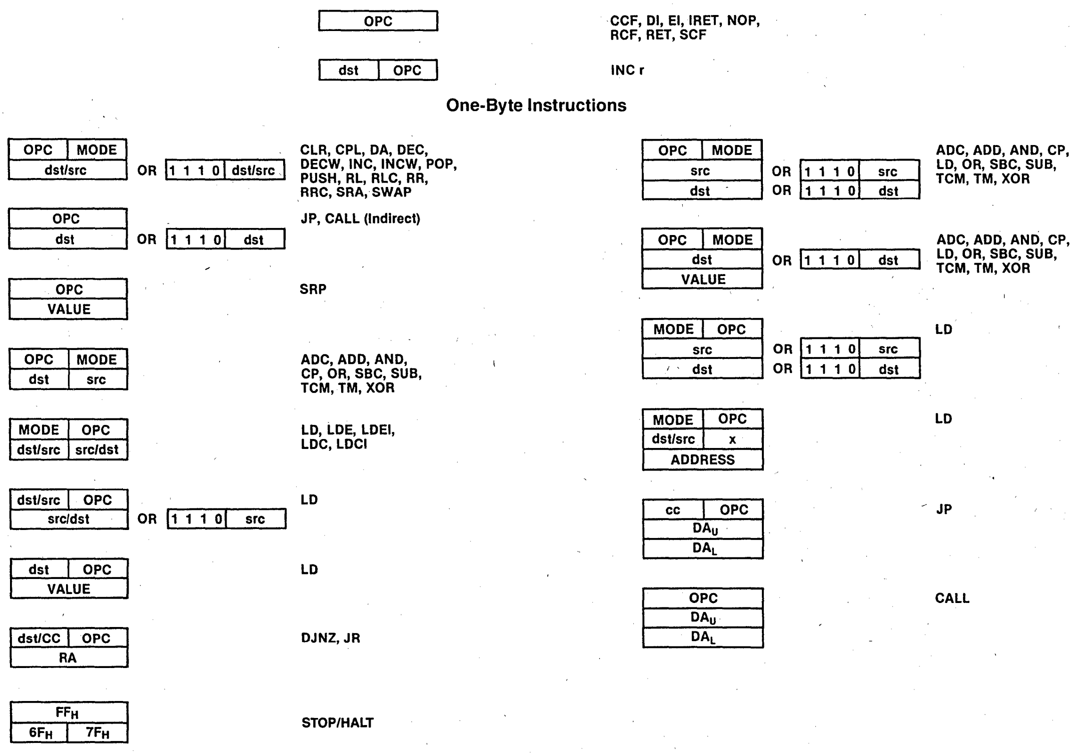
Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*	Set or cleared according to operation
—	Unaffected
X	Undefined

CONDITION CODES

Value	Mnemonic	Meaning	Flags Set
1000		Always true	—
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	—

INSTRUCTION FORMATS



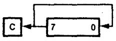
One-Byte Instructions

Two-Byte Instructions

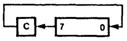
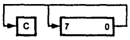
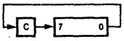
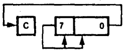
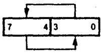
Three-Byte Instructions

INSTRUCTION SUMMARY

Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
ADC dst,src	(Note 1)		1□	*	*	*	*	0	*
dst ← dst + src + C									
ADD dst,src	(Note 1)		0□	*	*	*	*	0	*
dst ← dst + src									
AND dst,src	(Note 1)		5□	—	*	*	0	—	—
dst ← dst AND src									
CALL dst	DA		D6	—	—	—	—	—	—
SP ← SP − 2	IRR		D4						
@SP ← PC; PC ← dst									
CCF			EF	*	—	—	—	—	—
C ← NOT C									
CLR dst	R		B0	—	—	—	—	—	—
dst ← 0	IR		B1						
COM dst	R		60	—	*	*	0	—	—
dst ← NOT dst	IR		61						
CP dst,src	(Note 1)		A□	*	*	*	*	—	—
dst − src									
DA dst	R		40	*	*	*	X	—	—
dst ← DA dst	IR		41						
DEC dst	R		00	—	*	*	*	—	—
dst ← dst − 1	IR		01						
DECW dst	RR		80	—	*	*	*	—	—
dst ← dst − 1	IR		81						
DI									
IMR (7) ← 0			8F	—	—	—	—	—	—
DJNZ r,dst	RA		rA	—	—	—	—	—	—
r ← r − 1			r = 0 − F						
if r ≠ 0									
PC ← PC + dst									
Range: +127, −128									
EI			9F	—	—	—	—	—	—
IMR (7) ← 1									
HALT			7F						
INC dst	r		rE	—	*	*	*	—	—
dst ← dst + 1			r = 0 − F						
	R		20						
	IR		21						
INCW dst	RR		A0	—	*	*	*	—	—
dst ← dst + 1	IR		A1						
IRET			BF	*	*	*	*	*	*
FLAGS ← @SP; SP ← SP + 1									
PC ← @SP; SP ← SP + 2; IMR (7) ← 1									

Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
JP cc,dst	DA		cD	—	—	—	—	—	—
if cc is true			c = 0 − F						
PC ← dst	IRR		30						
JR cc,dst	RA		cB	—	—	—	—	—	—
if cc is true,			c = 0 − F						
PC ← PC + dst									
Range: +127, −128									
LD dst,src	r	Im	rC	—	—	—	—	—	—
dst ← src	r	R	r8						
	R	r	r9						
			r = 0 − F						
	r	X	C7						
	X	r	D7						
	r	Ir	E3						
	Ir	r	F3						
	R	R	E4						
	R	IR	E5						
	R	IM	E6						
	IR	IM	E7						
	IR	R	F5						
LDC dst,src	r	Irr	C2	—	—	—	—	—	—
dst ← src	Irr	r	D2						
LDCI dst,src	Ir	Irr	C3	—	—	—	—	—	—
dst ← src	Irr	Ir	D3						
r ← r + 1; rr ← rr + 1									
LDE dst,src	r	Irr	82	—	—	—	—	—	—
dst ← src	Irr	r	92						
LDEI dst,src	Ir	Irr	83	—	—	—	—	—	—
dst ← src	Irr	Ir	93						
r ← r + 1; rr ← rr + 1									
NOP			FF	—	—	—	—	—	—
OR dst,src	(Note 1)		4□	—	*	*	0	—	—
dst ← dst OR src									
POP dst	R		50	—	—	—	—	—	—
dst ← @SP;	IR		51						
SP ← SP + 1									
PUSH src		R	70	—	—	—	—	—	—
SP ← SP − 1; @SP ← src		IR	71						
RCF			CF	0	—	—	—	—	—
C ← 0									
RET			AF	—	—	—	—	—	—
PC ← @SP; SP ← SP + 2									
RL dst 	R		90	*	*	*	*	—	—
	IR		91						

INSTRUCTION SUMMARY (Continued)

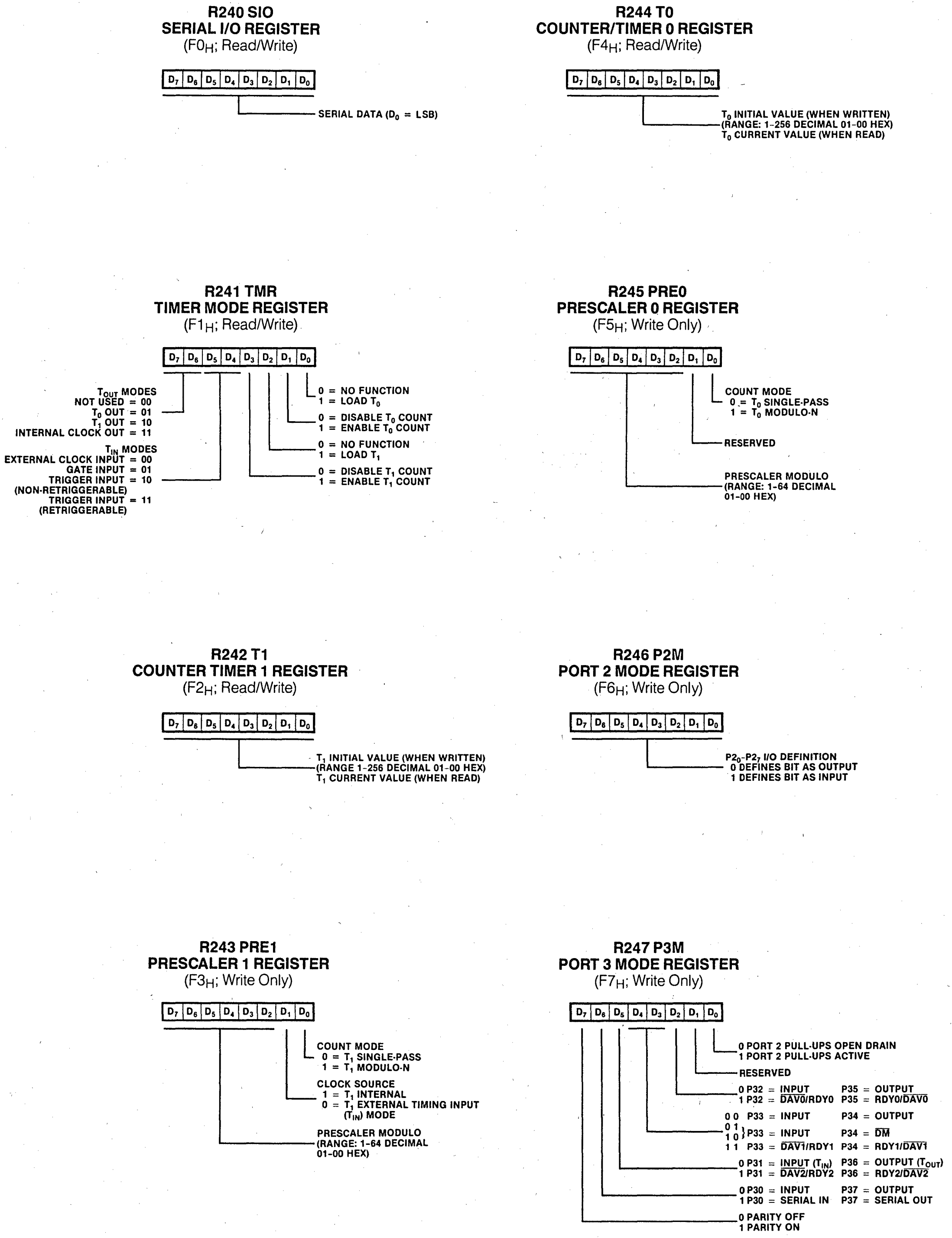
Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
RLC dst 	R IR		10 11	*	*	*	*	—	—
RR dst 	R IR		E0 E1	*	*	*	*	—	—
RRC dst 	R IR		C0 C1	*	*	*	*	—	—
SBC dst,src dst ← dst ← src ← C	(Note 1)		3□	*	*	*	*	1	*
SCF C ← 1			DF	1	—	—	—	—	—
SRA dst 	R IR		D0 D1	*	*	*	0	—	—
SRP src RP ← src		Im	31	—	—	—	—	—	—
STOP			6F						
SUB dst,src dst ← dst ← src	(Note 1)		2□	*	*	*	*	1	*
SWAP dst 	R IR		F0 F1	X	*	*	X	—	—
TCM dst,src (NOT dst) AND src	(Note 1)		6□	—	*	*	0	—	—

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
TM dst,src dst AND src	(Note 1)		7□	—	*	*	0	—	—
XOR dst,src dst ← dst XOR src	(Note 1)		B□	—	*	*	0	—	—

NOTE: These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the left of the applicable addressing mode pair.

For example, the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

Addr Mode dst	Addr Mode src	Lower Opcode Nibble
r	r	$\boxed{2}$
r	Ir	$\boxed{3}$
R	R	$\boxed{4}$
R	IR	$\boxed{5}$
R	IM	$\boxed{6}$
IR	IM	$\boxed{7}$



R240 SIO
SERIAL I/O REGISTER
($F0_H$; Read/Write)

R244 T0
COUNTER/TIMER 0 REGISTER
($F4_H$; Read/Write)

D7 D6 D5 D4 D3 D2 D1 D0
T0 INITIAL VALUE (WHEN WRITTEN)
(RANGE: 1-256 DECIMAL 01-00 HEX)
T0 CURRENT VALUE (WHEN READ)

R241 TMR
TIMER MODE REGISTER
($F1_H$; Read/Write)

R245 PRE0
PRESCALER 0 REGISTER
($F5_H$; Write Only)

R242 T1
COUNTER TIMER 1 REGISTER
($F2_H$; Read/Write)

D7 D6 D5 D4 D3 D2 D1 D0
T1 INITIAL VALUE (WHEN WRITTEN)
(RANGE 1-256 DECIMAL 01-00 HEX)
T1 CURRENT VALUE (WHEN READ)

R246 P2M
PORT 2 MODE REGISTER
($F6_H$; Write Only)

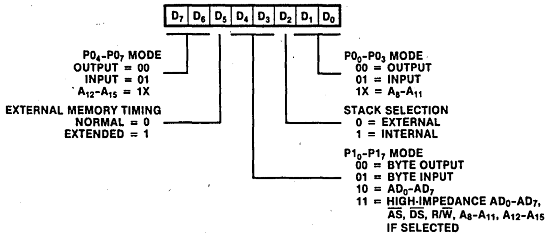
R243 PRE1
PRESCALER 1 REGISTER
($F3_H$; Write Only)

R247 P3M
PORT 3 MODE REGISTER
($F7_H$; Write Only)

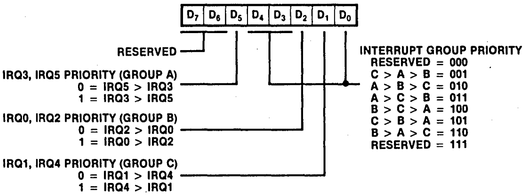
D7 D6 D5 D4 D3 D2 D1 D0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
RESERVED
0 P32 = INPUT P35 = OUTPUT
1 P32 = DAV0/RDY0 P35 = RDY0/DAV0
0 0 P33 = INPUT P34 = OUTPUT
0 1
1 0 } P33 = INPUT P34 = DM
1 1 P33 = DAV1/RDY1 P34 = RDY1/DAV1
0 P31 = INPUT (TIN) P36 = OUTPUT (TOUT)
1 P31 = DAV2/RDY2 P36 = RDY2/DAV2
0 P30 = INPUT P37 = OUTPUT
1 P30 = SERIAL IN P37 = SERIAL OUT
0 PARITY OFF
1 PARITY ON

Figure 11. Control Registers

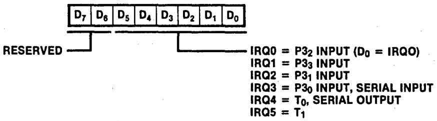
R248 P01M
PORT 0 AND 1 MODE REGISTER
($F8_H$; Write Only)



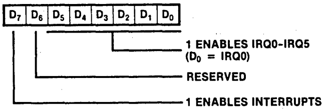
R249 IPR
INTERRUPT PRIORITY REGISTER
($F9_H$; Write Only)



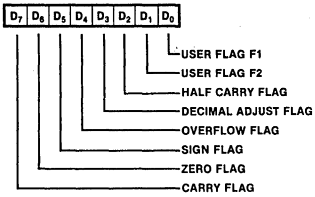
R250 IRQ
INTERRUPT REQUEST REGISTER
(FA_H; Read/Write)



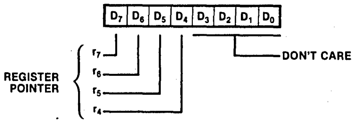
R251 IMR
INTERRUPT MASK REGISTER
(FB_H; Read/Write)



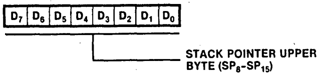
R252 FLAGS
FLAG REGISTER
(FC_H; Read/Write)



R253 RP
REGISTER POINTER
(FD_H; Read/Write)



R254 SPH
STACK POINTER
(FE_H; Read/Write)



R255 SPL
STACK POINTER
(FF_H; Read/Write)

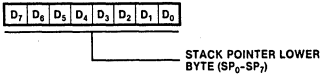
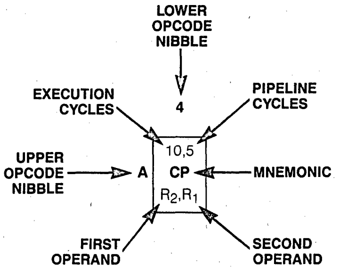


Figure 11. Control Registers (Continued)

OPCODE MAP

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6,5 **DEC** R_1	6,5 **DEC** IR_1	6,5 **ADD** $r_1.r_2$	6,5 **ADD** $r_1.Ir_2$	10.5 **ADD** $R_2.R_1$	10.5 **ADD** $IR_2.R_1$	10.5 **ADD** $R_1.IM$	10.5 **ADD** $IR_1.IM$	6,5 **LD** r_1,R_2	6,5 **LD** r_2,R_1	12/10,5 **DJNZ** r_1.RA	12/10.0 **JR** cc,RA	6,5 **LD** r_1,IM	12/10.0 **JP** cc,DA	6.5 **INC** r1	
1	6,5 **RLC** R_1	6,5 **RLC** IR_1	6,5 **ADC** r_1,r_2	6,5 **ADC** r_1,Ir_2	10,5 **ADC** R_2,R_1	10,5 **ADC** IR_2,R_1	10,5 **ADC** R_1,IM	10,5 **ADC** IR_1,IM								
2	6,5 **INC** R_1	6,5 **INC** IR_1	6,5 **SUB** r_1,r_2	6,5 **SUB** r_1,Ir_2	10,5 **SUB** R_2,R_1	10,5 **SUB** IR_2,R_1	10,5 **SUB** R_1,IM	10,5 **SUB** IR_1,IM								
3	8,0 **JP** IRR_1	6,1 **SRP** IM	6,5 **SBC** r_1,r_2	6,5 **SBC** r_1,Ir_2	10,5 **SBC** R_2,R_1	10,5 **SBC** IR_2,R_1	10,5 **SBC** R_1,IM	10,5 **SBC** IR_1,IM								
4	8,5 **DA** R_1	8,5 **DA** IR_1	6,5 **OR** r_1,r_2	6,5 **OR** r_1,Ir_2	10,5 **OR** R_2,R_1	10,5 **OR** IR_2,R_1	10,5 **OR** R_1,IM	10,5 **OR** IR_1,IM								
5	10,5 **POP** R_1	10,5 **POP** IR_1	6,5 **AND** r_1,r_2	6,5 **AND** r_1,Ir_2	10,5 **AND** R_2,R_1	10,5 **AND** IR_2,R_1	10,5 **AND** R_1,IM	10,5 **AND** IR_1,IM								
6	6,5 **COM** R_1	6,5 **COM** IR_1	6,5 **TCM** r_1,r_2	6,5 **TCM** r_1,Ir_2	10,5 **TCM** R_2,R_1	10,5 **TCM** IR_2,R_1	10,5 **TCM** R_1,IM	10,5 **TCM** IR_1,IM								6,0 **STOP**
7	10/12,1 **PUSH** R_2	12/14,1 **PUSH** IR_2	6,5 **TM** r_1,r_2	6,5 **TM** r_1,Ir_2	10,5 **TM** R_2,R_1	10,5 **TM** IR_2,R_1	10,5 **TM** R_1,IM	10,5 **TM** IR_1,IM								7,0 **HALT**
8	10,5 **DECW** RR_1	10,5 **DECW** IR_1	12,0 **LDE** r_1,Irr_2	18,0 **LDEI** Ir_1,Irr_2												6.1 **DI**
9	6,5 **RL** R_1	6,5 **RL** IR_1	12,0 **LDE** r_2,Irr_1	18,0 **LDEI** Ir_2,Irr_1												6.1 **EI**
A	10,5 **INCW** RR_1	10,5 **INCW** IR_1	6,5 **CP** r_1,r_2	6,5 **CP** r_1,Ir_2	10,5 **CP** R_2,R_1	10,5 **CP** IR_2,R_1	10,5 **CP** R_1,IM	10,5 **CP** IR_1,IM								14.0 **RET**
B	6,5 **CLR** R_1	6,5 **CLR** IR_1	6,5 **XOR** r_1,r_2	6,5 **XOR** r_1,Ir_2	10,5 **XOR** R_2,R_1	10,5 **XOR** IR_2,R_1	10.5 **XOR** R_1,IM	10,5 **XOR** IR_1,IM								16.0 **IRET**
C	6,5 **RRC** R_1	6,5 **RRC** IR_1	12,0 **LDC** r_1,Irr_2	18,0 **LDCI** Ir_1,Irr_2				10,5 **LD** r_1,x,R_2								6.5 **RCF**
D	6,5 **SRA** R_1	6,5 **SRA** IR_1	12,0 **LDC** r_2,Irr_1	18,0 **LDCI** Ir_2,Irr_1	20,0 **CALL*** IRR_1		20,0 **CALL** DA	10,5 **LD** r_2,x,R_1								6.5 **SCF**
E	6,5 **RR** R_1	6,5 **RR** IR_1		6,5 **LD** r_1,IR_2	10,5 **LD** R_2,R_1	10,5 **LD** IR_2,R_1	10,5 **LD** R_1,IM	10,5 **LD** IR_1,IM								6.5 **CCF**
F	8,5 **SWAP** R_1	8,5 **SWAP** IR_1		6,5 **LD** Ir_1,r_2		10,5 **LD** R_2,IR_1										6.0 **NOP**
Bytes per Instruction	2				3				2					3	1	



Legend:
R = 8-bit address
r = 4-bit address
R_1 or r_1 = Dst address
R_2 or r_2 = Src address

Sequence:
Opcode, First Operand, Second Operand

NOTE: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

ABSOLUTE MAXIMUM RATINGS

Voltages on all pins with respect
to GND . . . . –0.3V to +7.0V
Operating Ambient
Temperature . . . . .See Ordering Information
Storage Temperature . . . . –65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin.

Standard conditions are as follows:

- **+4.5 ≤ Vcc ≤ + 5.5V**
- GND = 0V
- **0 C ≤ T_A ≤ +70 C for S (Standard temperature)**
- **–40 C ≤ T_A ≤+100 C for E (Extended temperature)**

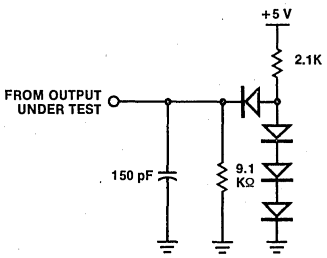


Figure 12. Test Load 1

DC CHARACTERISTICS

Symbol	Parameter	Min	Typ	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8		V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	–0.3		0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{IL}	Input Low Voltage	–0.3		0.8	V	
V_{RH}	Reset Input High Voltage	3.8		V_{CC}	V	
V_{RL}	Reset Input Low Voltage	–0.3		0.8	V	
V_{OH}	Output High Voltage	2.4			V	I_{OH} = –250 μA
V_{OH}	**Output High Voltage**	**V_{CC} -100mV**			**V**	**I_{OH} = -100μA**
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = +2.0 mA
I_{IL}	Input Leakage	–10		10	μA	0V ≤ V_{IN} ≤ + 5.25V
I_{OL}	Output Leakage	–10		10	μA	0V ≤ V_{IN} ≤ + 5.25V
I_{IR}	Reset Input Current			–50	μA	V_{CC} = +5.25V, V_{RL} = 0V
I_{CC}	Supply Current			30	mA	All outputs and I/O pins floating, 12 MHz
I_{CC1}	Standby Current		**5**		mA	Halt Mode
I_{CC2}	Standby Current			10	μA	Stop Mode

I_{CC}2 requires loading TMR (%F1) with any value prior to STOP execution.

Use the sequence:

```
LD TMR, #00
NOP
STOP
```

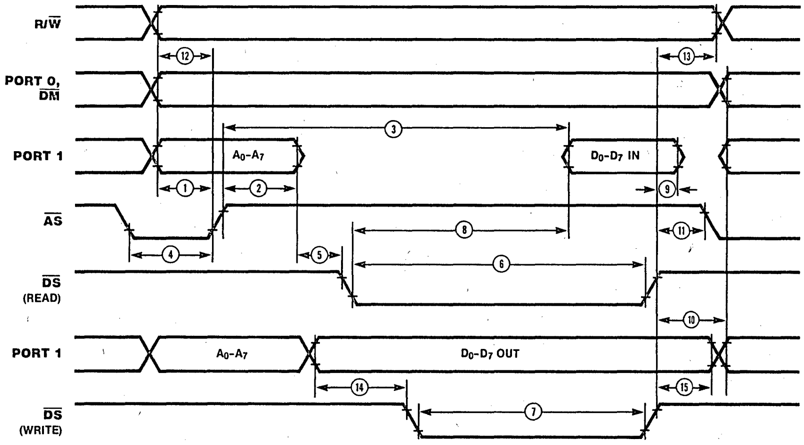


Figure 13. External I/O or Memory Read/Write

AC CHARACTERISTICS

External I/O or Memory Read and Write Timing

			12 MHz		16 MHz		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes*†°
1	TdA(AS)	Address Valid to $\overline{AS}$ ↑ Delay	35		20		2,3
2	TdAS(A)	$\overline{AS}$ ↑ to Address Float Delay	45		30		2,3
3	TdAS(DR)	$\overline{AS}$ ↑ to Read Data Required Valid		220		180	1,2,3
4	TwAS	$\overline{AS}$ Low Width	55		35		2,3
5	TdAz(DS)	Address Float to $\overline{DS}$ ↓	0		0		
6	TwDSR	$\overline{DS}$ (Read) Low Width	185		135		1,2,3
7	TwDSW	$\overline{DS}$ (Write) Low Width	110		80		1,2,3
8	TdDSR(DR)	$\overline{DS}$ ↓ to Read Data Required Valid		130		75	1,2,3
9	ThDR(DS)	Read Data to $\overline{DS}$ ↑ Hold Time	0		0		
10	TdDS(A)	$\overline{DS}$ ↑ to Address Active Delay	45		20		2,3
11	TdDS(AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	55		20		2,3
12	TdR/W(AS)	R/$\overline{W}$ Valid to $\overline{AS}$ ↑ Delay	30		20		2,3
13	TdDS(R/W)	$\overline{DS}$ ↑ to R/W Not Valid	35		20		2,3
14	TdDW(DSW)	Write Data Valid to $\overline{DS}$ (Write) ↓ Delay	35		25		2,3
15	TdDS(DW)	$\overline{DS}$ ↑ to Write Data Not Valid Delay	35		20		2,3
16	TdA(DR)	Address Valid to Read Data Required Valid		255		200	1,2,3
17	TdAS(DS)	$\overline{AS}$ ↑ to $\overline{DS}$ ↓ Delay	55		40		2,3

NOTES:
1. When using extended memory timing add 2 TpC.
2. Timing numbers given are for minimum TpC.
3. See clock cycle time dependent characteristics table.

* All units in nanoseconds (ns).
† Test Load 1
° All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".

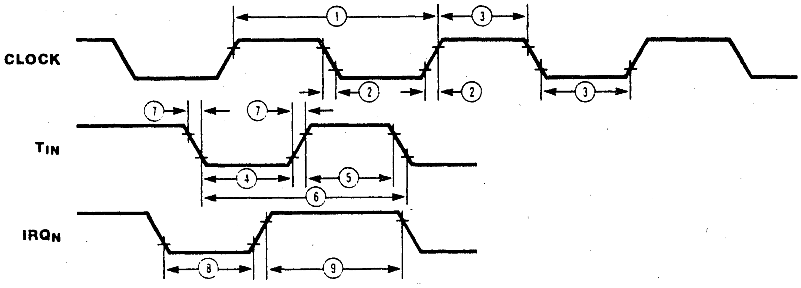


Figure 14. Additional Timing

AC CHARACTERISTICS

Additional Timing Table

			12 MHz		16 MHz		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes*
1	TpC	Input Clock Period	83	1000	62.5	1000	1
2	TrC,TfC	Clock Input Rise and Fall Times		15		10	1
3	TwC	Input Clock Width	70		21		1
4	TwTinL	Timer Input Low Width	70		50		2
5	TwTinH	Timer Input High Width	3TpC		3TpC		2
6	TpTin	Timer Input Period	8TpC		8TpC		2
7	TrTin,TfTin	Timer Input Rise and Fall Times		100		100	2
8A	TwIL	Interrupt Request Input Low Time	70		50		2,4
8B	TwIL	Interrupt Request Input Low Time	3TpC		3TpC		2,5
9	TwIH	Interrupt Request Input High Time	3TpC		3TpC		2,3

NOTES:
1. Clock timing references use 3.8V for a logic "1" and 0.8V for a logic "0".
2. Timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
3. Interrupt request via Port 3.
4. Interrupt request via Port 3 (P3$_1$-P3$_3$).
5. Interrupt request via Port 3 (P3$_0$).
* Units in nanoseconds (ns).

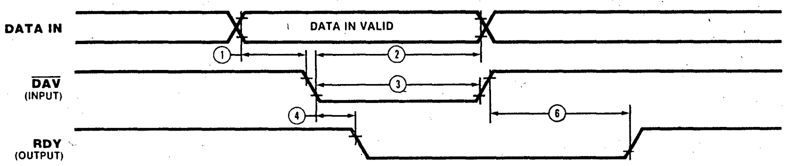


Figure 15a. Input Handshake

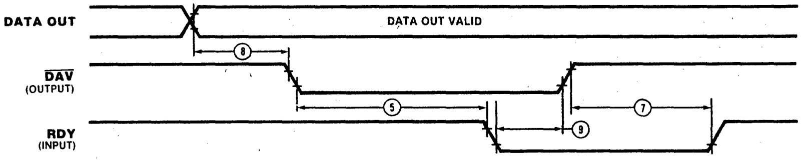


Figure 15b. Output Handshake

AC CHARACTERISTICS

Handshake Timing

			12MHz, 16MHz		
Number	**Symbol**	**Parameter**	**Min**	**Max**	**Notes†***
1	TsDI(DAV)	Data In Setup Time	0		
2	ThDI(DAV)	Data In Hold Time	145		
3	TwDAV	Data Available Width	110		
4	TdDAVIf(RDY)	$\overline{\text{DAV}}$ ↓ Input to RDY ↓ Delay	20	115	1,2
5	TdDAVOf(RDY)	$\overline{\text{DAV}}$ ↓ Output to RDY ↓ Delay	0		1,3
6	TdDAVIr(RDY)	$\overline{\text{DAV}}$ ↑ Input to RDY ↑ Delay		115	1,2
7	TdDAVOr(RDY)	$\overline{\text{DAV}}$ ↑ Output to RDY ↑ Delay	0		1,3
8	TdDO(DAV)	Data Out to $\overline{\text{DAV}}$ ↓ Delay	Tpc		1
9	TdRDY(DAV)	RDY ↓ Input to $\overline{\text{DAV}}$ ↑ Delay	0	130	1

NOTES:
1. Test load 1
2. Input handshake
3. Output handshake

† All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
* Units in nanoseconds (ns).

Zilog

ADVANCED INFORMATION
Product Specification

Z86C21/Z86E21 CMOS
CMOS Z8® 8K ROM MCU

June 1987

FEATURES

- Complete microcomputer, **8K** bytes of ROM, **256** bytes of RAM, 32 I/O lines, and up to **56K** bytes addressable external space each for program and data memory.
- **256**-byte register file, including **236** general-purpose registers, **4** I/O port registers, and 16 status and control registers.
- **Minimum instruction execution time of 0.6 μs, average of 1.0 μs.**
- Vectored, priority interrupts for I/O, counter/timers, and UART.
- Full-duplex UART and two programmable 8-bit counter/timers, each with a 6-bit programmable prescaler.
- **Register Pointer so that short, fast instructions can access any of 16 working–register groups in .6 μs.**
- On-chip oscillator which accepts crystal or external clock drive.
- Standby modes—Halt and Stop
- Single +5V power supply—all pins TTL-compatible.
- 12 and 16 MHz.
- CMOS process
- **Z86E21 compatible field–programmable version –– same feature set.**

GENERAL DESCRIPTION

The Z86C21 microcomputer (Figures 1 and 2) introduces a new level of sophistication to single–chip architecture. Compared to earlier single–chip microcomputers, the Z86C21 offers faster execution; more efficient use of memory; more sophisticated interrupt, input/output and bit–manipulation capabilities; and easier system expansion.

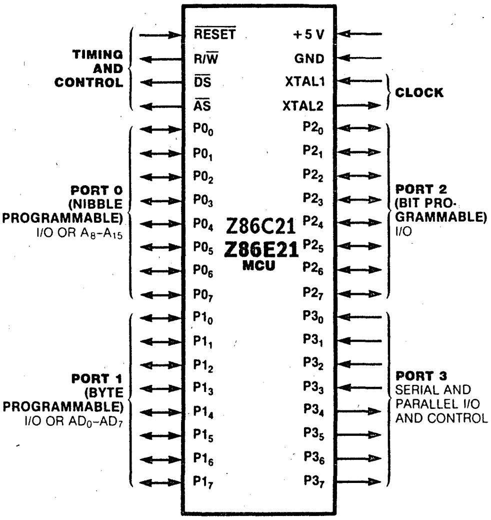

TIMING AND CONTROL
$\overline{RESET}$
R/$\overline{W}$
$\overline{DS}$
$\overline{AS}$
+5 V
GND
XTAL1
XTAL2
CLOCK
PORT 0 (NIBBLE PROGRAMMABLE) I/O OR A_8–A_{15}
$P0_0$ $P0_1$ $P0_2$ $P0_3$ $P0_4$ $P0_5$ $P0_6$ $P0_7$
PORT 1 (BYTE PROGRAMMABLE) I/O OR AD_0–AD_7
$P1_0$ $P1_1$ $P1_2$ $P1_3$ $P1_4$ $P1_5$ $P1_6$ $P1_7$
Z86C21
Z86E21
MCU
$P2_0$ $P2_1$ $P2_2$ $P2_3$ $P2_4$ $P2_5$ $P2_6$ $P2_7$
PORT 2 (BIT PROGRAMMABLE) I/O
$P3_0$ $P3_1$ $P3_2$ $P3_3$ $P3_4$ $P3_5$ $P3_6$ $P3_7$
PORT 3 SERIAL AND PARALLEL I/O AND CONTROL

Figure 1. Pin Functions

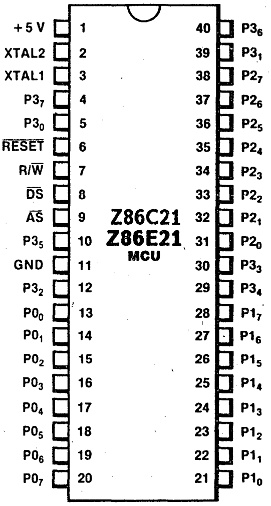


Figure 2. 40-pin Dual-In-Line Package (DIP), Pin Assignments

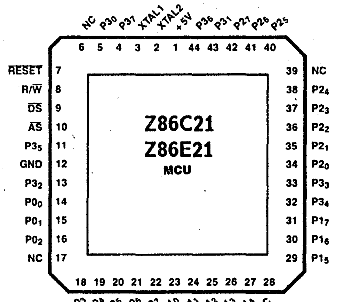


Figure 2b. 44-pin Chip Carrier, Pin Assignments

General Purpose Microcontroller

Under program control, the Z86C21 can be tailored to the needs of its user. It can be configured as a stand–alone microcomputer with 8K bytes of internal ROM, a traditional microprocessor that manages up to 112K bytes of external memory, or a parallel–processing element in a system with other processors and peripheral controllers linked by the Z–BUS bus. In all configurations, a large number of pins remain available for I/O.

Field Programmable Version

The Z86E21 is a pin compatible Onetime Programmable version of the Z86C21. The Z86E21 contains 8K bytes of EPROM memory in place of the 8K bytes of masked ROM on the Z86C21. The Z86E21 also contains a programmable memory protect feature to provide program security by disabling all external accesses to the internal EPROM array.

ARCHITECTURE

Z86C21 architecture is characterized by a flexible I/O scheme, an efficient register and address space structure and a number of ancillary features that are helpful in many applications.

Microcomputer applications demand powerful I/O capabilities. The **Z86C21** fulfills this with 32 pins dedicated to input and output. These lines are grouped into four ports of eight lines each and are configurable under software control to provide timing, status signals, serial or parallel I/O with or without handshake, and an address/data bus for interfacing external memory.

Because the multiplexed address/data bus is merged with the I/O-oriented ports, the **Z86C21** can assume many different memory and I/O configurations. These configurations range from a self-contained microcomputer to a microprocessor that can address 120K bytes of external memory (Figure 3).

Three basic address spaces are available to support this wide range of configurations: program memory (internal and external), data memory (external) and the register file (internal). The 256–byte random–access register file is composed of 236 general–purpose registers, 4 I/O port registers, and 16 control and status registers.

To unburden the program from coping with real-time problems such as serial data communication and counting/timing, an asynchronous receiver/transmitter (UART) and two counter/timers with a large number of user-selectable modes are offered on-chip. Hardware support for the UART is minimized because one of the on-chip timers supplies the bit rate.

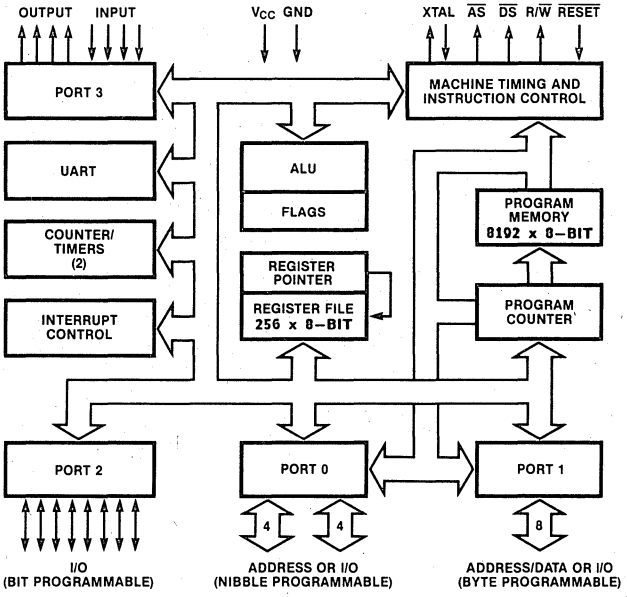


Figure 3. Functional Block Diagram

STANDBY MODE

The Z86C21's standby modes are:

- Stop
- Halt

The Stop instruction stops the internal clock and clock oscillation; the Halt instruction stops the internal clock but not clock oscillation.

A reset input releases the standby mode.

To complete an instruction prior to entering standby mode, use the instructions:

NOP(FF_H) + STOP($6F_H$)
NOP(FF_H) + HALT($7F_H$)

PIN DESCRIPTION

$\overline{AS}$. *Address Strobe* (output, active Low). Address Strobe is pulsed once at the beginning of each machine cycle. Addresses output via Port 1 for all external program or data memory transfers are valid at the trailing edge of $\overline{AS}$. Under program control, $\overline{AS}$ can be placed in the high-impedance state along with Ports 0 and 1, Data Strobe and Read/Write.

$\overline{DS}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each external memory transfer.

$P0_0$-$P0_7$, $P1_0$-$P1_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. *I/O Port Lines* (input/outputs, TTL-compatible). These 32 lines are divided into four 8-bit I/O ports that can be configured under program control for I/O or external memory interface **(Figure 3)**.

$\overline{RESET}$. *Reset* (input, active Low). $\overline{RESET}$ initializes the **Z86C21** . When $\overline{RESET}$ is deactivated, program execution begins from internal program location $000C_H$.

R/$\overline{W}$. *Read/Write* (output). R/$\overline{W}$ is Low when the **Z86C21** is writing to external program or data memory.

XTAL1, XTAL2. ***Crystal 1, Crystal 2*** **(time–base input and output). These pins connect a parallel–resonant crystal (12 or 20 MHz maximum) or an external single–phase clock (12 or 20 MHz maximum) to the on–chip clock oscillator and buffer.**

ADDRESS SPACE

Program Memory. The 16-bit program counter addresses 64K bytes of program memory space. Program memory can be located in two areas: one internal and the other external (Figure 4). The first **8192** bytes consist of on-chip mask-programmed ROM. At addresses **8192** and greater, the **Z86C21** executes external program memory fetches.

The first 12 bytes of program memory are reserved for the interrupt vectors. These locations contain six 16-bit vectors that correspond to the six available interrupts.

Data Memory. The **Z86C21** can address **56K** bytes of external data memory beginning at location 4096 (Figure 5). External data memory may be included with or separated from the external program memory space. $\overline{DM}$, an optional I/O function that can be programmed to appear on pin $P3_4$, is used to distinguish between data and program memory space.

Register File. The 256–byte register file includes 4 I/O port registers (R0–R3), 236 general–purpose registers (R4–R239) and 16 control and status registers (R240–R255). These registers are assigned the address locations shown in Figure 6.

Z86C21 instructions can access registers directly or indirectly with an 8–bit address field. The Z86C21 also allows short 4–bit register addressing using the Register Pointer (one of the control registers). In the 4–bit mode, the register file is divided into 16 working register groups, each occupying 16 contiguous locations (Figure 6). The Register Pointer addresses the starting location of the active working–register group (Figure 7). Note: Register Bank E0–EF can only be accessed through working register and indirect addressing mode.

Stacks. Either the internal register file or the external data memory can be used for the stack. A 16-bit Stack Pointer (R254 and R255) is used for the external stack, which can reside anywhere in data memory between locations 4096 and 65535. An 8-bit Stack Pointer (R255) is used for the internal stack that resides within the 124 general-purpose registers (R4-R127).

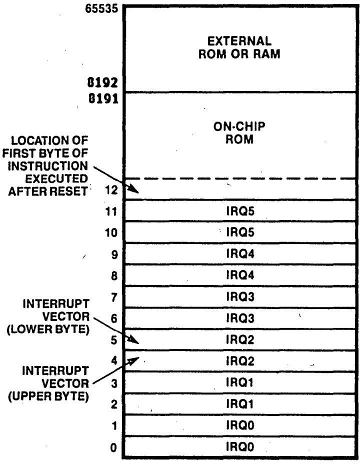


Figure 4. Program Memory Map

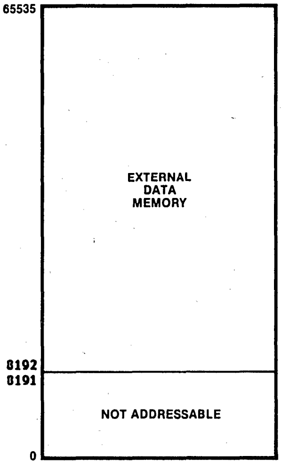
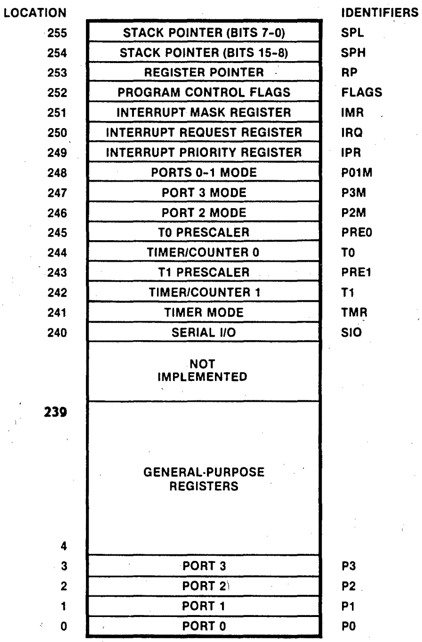


Figure 5. Data Memory Map



LOCATION		IDENTIFIERS
255	STACK POINTER (BITS 7-0)	SPL
254	STACK POINTER (BITS 15-8)	SPH
253	REGISTER POINTER	RP
252	PROGRAM CONTROL FLAGS	FLAGS
251	INTERRUPT MASK REGISTER	IMR
250	INTERRUPT REQUEST REGISTER	IRQ
249	INTERRUPT PRIORITY REGISTER	IPR
248	PORTS 0-1 MODE	P01M
247	PORT 3 MODE	P3M
246	PORT 2 MODE	P2M
245	T0 PRESCALER	PRE0
244	TIMER/COUNTER 0	T0
243	T1 PRESCALER	PRE1
242	TIMER/COUNTER 1	T1
241	TIMER MODE	TMR
240	SERIAL I/O	SIO
	NOT IMPLEMENTED	
239	GENERAL-PURPOSE REGISTERS	
4		
3	PORT 3	P3
2	PORT 2	P2
1	PORT 1	P1
0	PORT 0	P0

Figure 6. The Register File

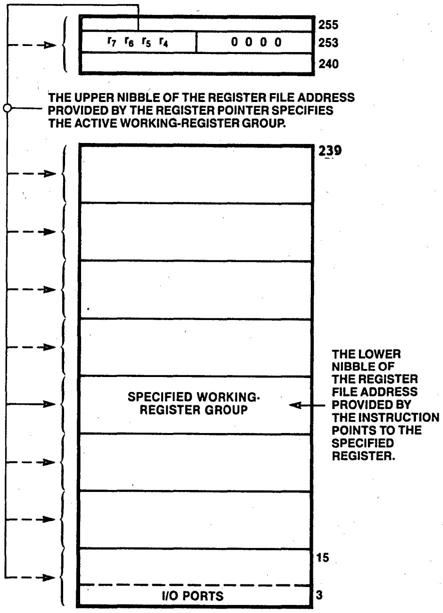

r_7 r_6 r_5 r_4
0 0 0 0
255
253
240
THE UPPER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE REGISTER POINTER SPECIFIES THE ACTIVE WORKING-REGISTER GROUP.
239
SPECIFIED WORKING-REGISTER GROUP
THE LOWER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE INSTRUCTION POINTS TO THE SPECIFIED REGISTER.
15
I/O PORTS
3

Figure 7. The Register Pointer

SERIAL INPUT/OUTPUT

Port 3 lines $P3_0$ and $P3_7$ can be programmed as serial I/O lines for full-duplex serial asynchronous receiver/transmitter operation. The bit rate is controlled by Counter/Timer 0.

The **Z86C21** automatically adds a start bit and two stop bits to transmitted data (Figure 8). Odd parity is also available as an option. Eight data bits are always transmitted, regardless of parity selection. If parity is enabled, the eighth bit is the odd parity bit. An interrupt request (IRQ_4) is generated on all transmitted characters.

Received data must have a start bit, eight data bits and at least one stop bit. If parity is on, bit 7 of the received data is replaced by a parity error flag. Received characters generate the IRQ_3 interrupt request.

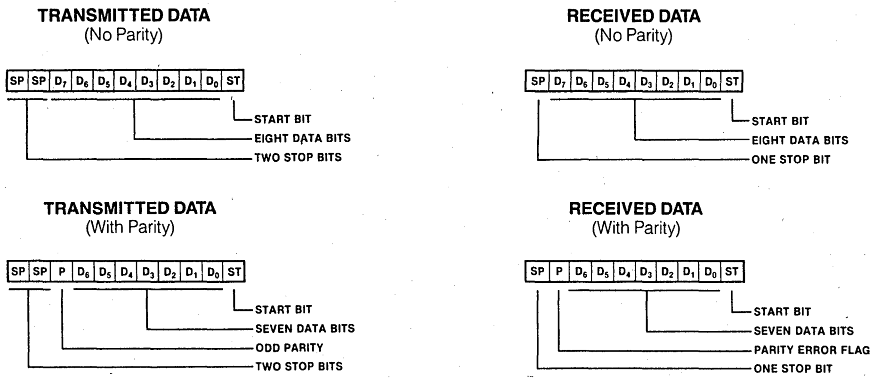


Figure 8. Serial Data Formats

COUNTER/TIMERS

The **Z86C21** contains two 8-bit programmable counter/timers (T_0 and T_1), each driven by its own 6-bit programmable prescaler. The T_1 prescaler can be driven by internal or external clock sources; however, the T_0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrements the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request—IRQ_4 (T_0) or IRQ_5 (T_1)—is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single-pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read any time without disturbing their value or count mode.

The clock source for T_1 is user—definable and can be the internal microprocessor clock divided by four, or an external signal input via Port 3. The Timer Mode register configures the external timer input as an external clock (1MHz maximum), a trigger input that can be retriggerable or non—retriggerable, or as a gate input for the internal clock. The counter/timers can be programmably cascaded by connecting the T_0 output to the input of T_1. Port 3 line $P3_6$ also serves as a timer output (T_{OUT}) through which T_0, T_1 or the internal clock can be output.

I/O PORTS

The **Z86C21** has 32 lines dedicated to input and output. These lines are grouped into four ports of eight lines each and are configurable as input, output or address/data. Under software control, the ports can be programmed to provide address outputs, timing, status signals, serial I/O, and parallel I/O with or without handshake. All ports have active pull-ups and pull-downs compatible with TTL loads.

Port 1 can be programmed as a byte I/O port or as an address/data port for interfacing external memory. When used as an I/O port, Port 1 may be placed under handshake control. In this configuration, Port 3 lines P3_3 and P3_4 are used as the handshake controls RDY$_1$ and $\overline{\text{DAV}}_1$ (Ready and Data Available).

Memory locations greater than **8192** are referenced through Port 1. To interface external memory, Port 1 must be programmed for the multiplexed Address/Data mode. If more than 256 external locations are required, Port 0 must output the additional lines.

Port 1 can be placed in the high-impedance state along with Port 0, $\overline{\text{AS}}$, $\overline{\text{DS}}$ and R/$\overline{\text{W}}$, allowing the **Z86C21** to share common resources in multiprocessor and DMA applications. Data transfers can be controlled by assigning P3_3 as a Bus Acknowledge input, and P3_4 as a Bus Request output.

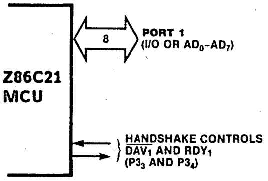


Figure 9a. Port 1

Port 0 can be programmed as a nibble I/O port, or as an address port for interfacing external memory. When used as an I/O port, Port 0 may be placed under handshake control. In this configuration, Port 3 lines P3_2 and P3_5 are used as the handshake controls $\overline{\text{DAV}}_0$ and RDY$_0$. Handshake signal assignment is dictated by the I/O direction of the upper nibble P0_4-P0_7.

For external memory references, Port 0 can provide address bits A$_8$-A$_{11}$ (lower nibble) or A$_8$-A$_{15}$ (lower and upper nibble) depending on the required address space. If the address range requires 12 bits or less, the upper nibble of Port 0 can be programmed independently as I/O while the lower nibble is used for addressing. When Port 0 nibbles are defined as address bits, they can be set to the high-impedance state along with Port 1 and the control signals $\overline{\text{AS}}$, $\overline{\text{DS}}$ and R/$\overline{\text{W}}$.

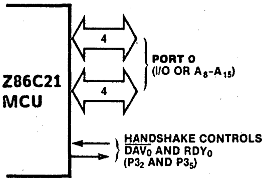


Figure 9b. Port 0

Port 2 bits can be programmed independently as input or output. This port is always available for I/O operations. In addition, Port 2 can be configured to provide open-drain outputs.

Like Ports 0 and 1, Port 2 may also be placed under handshake control. In this configuration, Port 3 lines P3_1 and P3_6 are used as the handshake controls lines $\overline{\text{DAV}}_2$ and RDY$_2$. The handshake signal assignment for Port 3 lines P3_1 and P3_6 is dictated by the direction (input or output) assigned to bit 7 of Port 2.

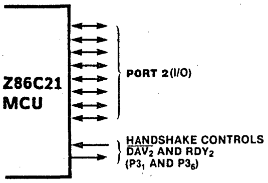


Figure 9c. Port 2

Port 3 lines can be configured as I/O or control lines. In either case, the direction of the eight lines is fixed as four input (P3_0-P3_3) and four output (P3_4-P3_7). For serial I/O, lines P3_0 and P3_7 are programmed as serial in and serial out respectively.

Port 3 can also provide the following control functions: handshake for Ports 0, 1 and 2 ($\overline{\text{DAV}}$ and RDY); four external interrupt request signals (IRQ$_0$-IRQ$_3$); timer input and output signals (T$_{\text{IN}}$ and T$_{\text{OUT}}$) and Data Memory Select ($\overline{\text{DM}}$).

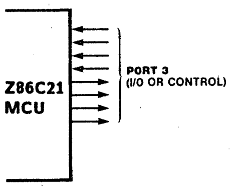


Figure 9d. Port 3

INTERRUPTS

The **Z86C21** allows six different interrupts from eight sources: the four Port 3 lines P3$_0$-P3$_3$, Serial In, Serial Out, and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask register globally or individually enables or disables the six interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All Z86C21 interrupts are vectored through locations in program memory. When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the Program Counter and status flags, and branches to the program memory vector location reserved for that interrupt. This memory location and the next byte contain the 16–bit address of the interrupt service routine for that particular interrupt request.

Polled interrupt systems are also supported. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the Interrupt Request register polled to determine which of the interrupt requests needs service.

CLOCK

The on-chip oscillator has a high-gain, parallel-resonant amplifier for connection to a crystal or to any suitable external clock source (XTAL1 = Input, XTAL2 = Output).

The crystal source is connected across XTAL1 and XTAL2, using the recommended capacitors ($C_1 \leqslant$ 15 pf) from each pin to ground. The specifications for the crystal are as follows:

- AT cut, parallel resonant
- Fundamental type, 16 MHz maximum
- Series resistance, $R_s \leqslant 100\ \Omega$

GENERAL DESCRIPTION

The Z86C12 development device allows users to prototype a system with an actual hardware device and to develop the code. This code is eventually mask-programmed into the on-chip ROM for any of the 86Cxx devices (except the 86C91). Development devices are also useful in emulator appli-cations where the final system configura-tion -- memory configuration, I/O, interrupt inputs, etc. -- are unknown.The Z86C12 development device is identical to its equivalent Z86C21 microcomputer with the following exceptions:

- No internal ROM is provided, so that code is developed in off-chip memory. Five "size" inputs configure the memory boundaries.

- The normally internal ROM address and data lines are buffered and brought out to external pins to interface with the external memory.

- Control lines (/MAS and /MDS) are added to interface with external program memory.

The Timing and Control, I/O ports, and clock pins on the Z86C12 are identical in function to those on the 86C21. This section covers those pins that do not appear on the Z86C21 8K ROM device. The pin functions and pin assignments are shown on figure 00.

Z86C12 PIN DESCRIPTION

D0 - D7 (Inputs, TTL compatible) Data bus. These 8 lines provide the input data bus to access external memory emulating on the on-chip ROM. During read cycles in the internal memory space the data on these lines is latched in just prior to the rise of the /MDS data strobe.

A0 - A15 (Outpus TTL compatible) Address bus. During T1 these lines output the current memory address. All addresses, whether internal or external, are output.

/MAS (Output, TTL compatible) Memory Address Strobe. This line is active during every T1 cycle. The rising edge of this signal may be used to latch the current memory address on the lines A0 - A15. This line is always valid; it is not tri-stated when /AS is tri-stated.

/MDS (Output, TTL compatible) Memory Data Strobe. This is a timing signal used to enable the external memory to emulate the on-chip ROM. It is active only during accesses to the on-chip ROM memory space, as selected by the configuration of the SIZEn pins.

/SCLK (Output, TTL compatible) System Clock. This line is teh internal system clock.

/SYNC (Output TTL, compatible) Sync signal. This signal indicates the last clock cycle of the currently executing instruction.

/IACK (Output TTL, compatible) Interrupt Acknow-ledge. This output, when low, indicates that the Z86C12 is an interrupt cycle.

/SIZE0, /SIZE1, /SIZE2, /SIZE3, SIZE4 (Inputs, TTL compatible). The /SIZEn lines control the emulation mode of the 86C12. Note that /SIZE0 - /SIZE3 are active low, while SIZE4 is active high. The functions are defined as shown in figure 00. The 86C12 should be in RESET when the state of these lines are changed.

NOTE:

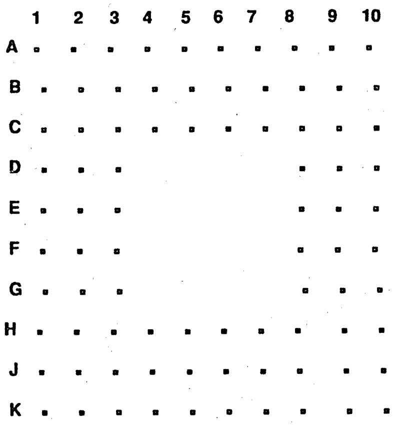
The SIZE pins may be configured to make the memory control signals (/MAS, /MDS, R/W, /AS, and /DS) look like the Z86C91 ROMless device, however on power-up or reset ports 0 and 1 are configured as inputs, rather than A15 - A8 and AD7 - AD0, respectively.

Table 1. Z86C12 Pin Assignments

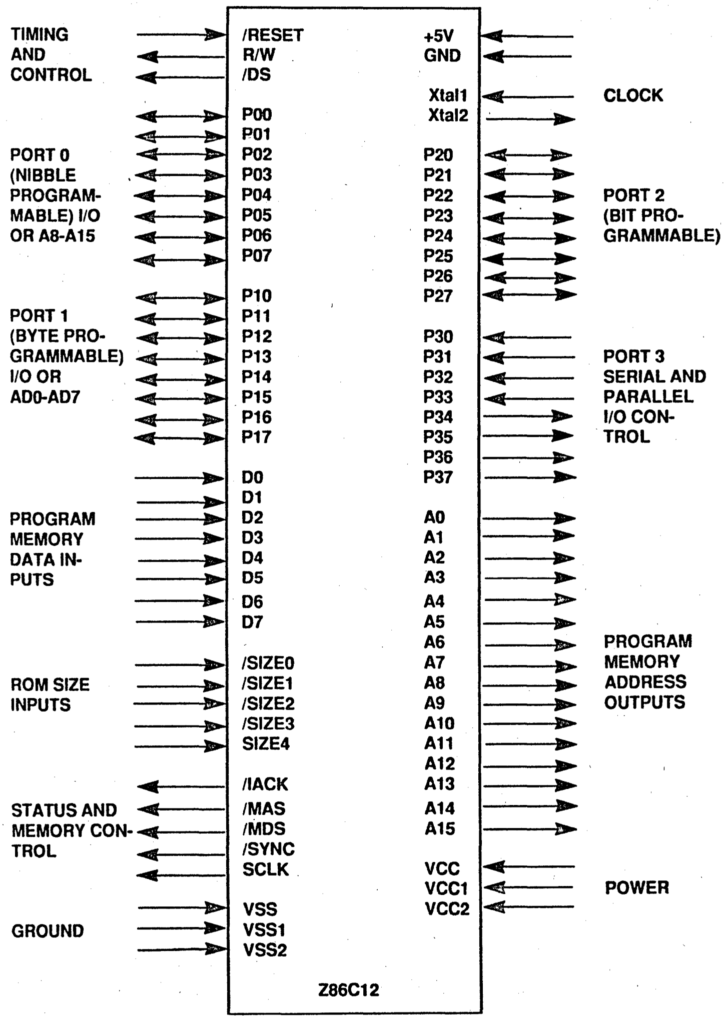
NAME		NAME	PIN	NAME	PIN	NAME	PIN
/AS	B2	A8	J5	P07	J1	P36	A7
/DS	C4	A9	K4	P10	G8	P37	A5
/MAS	E1	D0	H3	P11	G9	R/W	A1
/MDS	G3	D1	K2	P12	G10	SCLK	G2
/RESET	B3	D2	J3	P13	F8	SIZE4	F10
/SIZE0	A3	D3	K3	P14	D10	VCC	A4
/SIZE1	C5	D4	H8	P15	C10	VCC1	B6
/SIZE2	A6	D5	J10	P16	B10	VCC2	F9
/SIZE3	C6	D6	H9	P17	E9	VSS	F3
/SYNC	F1	D7	H10	P20	C9	VSS1	E2
A0	J9	IACK	F2	P21	A10	VSS2	H6
A1	H7	NC	J2	P22	B9	VSS3	E8
A10	J4	NC	C3	P23	C8	Xtal1	B5
A11	H4	NC	D8	P24	A9	Xtal2	A2
A12	K9	NC	H2	P25	B8		
A13	K7	NC	K1	P26	A8		
A14	K5	P00	C1	P27	C7		
A15	H5	P01	D3	P30	B4		
A2	K10	P02	D2	P31	B7		
A3	J8	P03	D1	P32	C2		
A4	J7	P04	E3	P33	D9		
A5	K6	P05	G1	P34	E10		
A6	J6	P06	H1	P35	B1		
A7	K8						

Table 2. Memory Size Configuration

SIZE4	/SIZE3	/SIZE2	/SIZE1	/SIZE0	MEMORY
0	1	1	1	1	ROMless
0	1	1	1	0	2K ROM
0	1	1	0	1	4K ROM
0	1	0	1	1	8K ROM
0	0	1	1	1	16K ROM
1	1	1	1	1	32K ROM



TOP VIEW



Z86C12 Pin Functions

INSTRUCTION SET NOTATION

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "←". For example,

$$dst \leftarrow dst + src$$

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

$$dst\,(7)$$

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

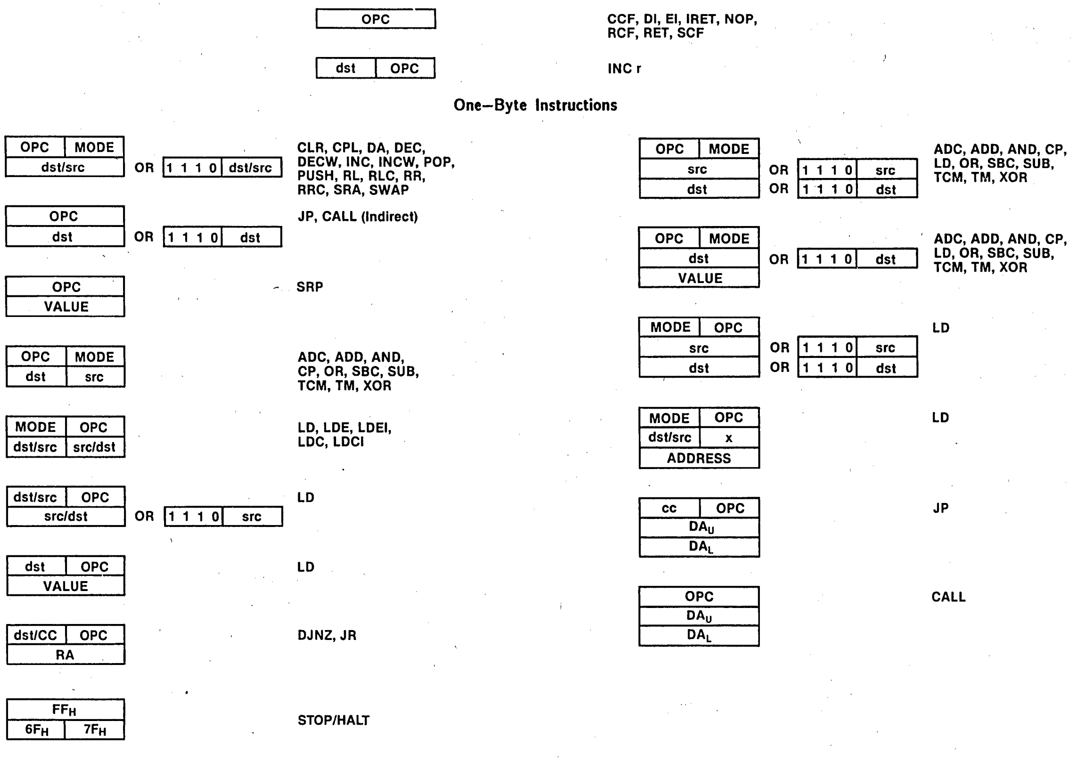
Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*	Set or cleared according to operation
—	Unaffected
X	Undefined

CONDITION CODES

Value	Mnemonic	Meaning	Flags Set
1000		Always true	—
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	—

INSTRUCTION FORMATS



One–Byte Instructions

Two–Byte Instructions

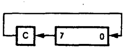
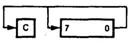
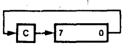
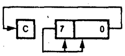
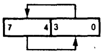
Three–Byte Instructions

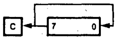
INSTRUCTION SUMMARY

Instruction and Operation	Addr Mode dst	src	Opcode Byte (Hex)	C	Z	S	V	D	H
ADC dst,src	(Note 1)		1□	*	*	*	*	0	*
dst ← dst + src + C									
ADD dst,src	(Note 1)		0□	*	*	*	*	0	*
dst ← dst + src									
AND dst,src	(Note 1)		5□	—	*	*	0	—	—
dst ← dst AND src									
CALL dst	DA		D6	—	—	—	—	—	—
SP ← SP − 2	IRR		D4						
@SP ← PC; PC ← dst									
CCF			EF	*	—	—	—	—	—
C ← NOT C									
CLR dst	R		B0	—	—	—	—	—	—
dst ← 0	IR		B1						
COM dst	R		60	—	*	*	0	—	—
dst ← NOT dst	IR		61						
CP dst,src	(Note 1)		A□	*	*	*	*	—	—
dst − src									

Instruction and Operation	Addr Mode dst	src	Opcode Byte (Hex)	C	Z	S	V	D	H
JP cc,dst	DA		cD	—	—	—	—	—	—
if cc is true			c = 0 − F						
PC ← dst	IRR		30						
JR cc,dst	RA		cB	—	—	—	—	—	—
if cc is true,			c = 0 − F						
PC ← PC + dst									
Range: +127, −128									
LD dst,src	r	Im	rC	—	—	—	—	—	—
dst ← src	r	R	r8						
	R	r	r9						
			r = 0 − F						
	r	X	C7						
	X	r	D7						
	r	Ir	E3						
	Ir	r	F3						
	R	R	E4						
	R	IR	E5						
	R	IM	E6						
	IR	IM	E7						
	IR	R	F5						

INSTRUCTION SUMMARY (Continued)

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
DA dst	R		40	*	*	*	X	—	—
dst ← DA dst	IR		41						
DEC dst	R		00	—	*	*	*	—	—
dst ← dst − 1	IR		01						
DECW dst	RR		80	—	*	*	*	—	—
dst ← dst − 1	IR		81						
DI									
IMR (7) ← 0			8F	—	—	—	—	—	—
DJNZ r,dst	RA		rA	—	—	—	—	—	—
r ← r − 1			r = 0 − F						
if r ≠ 0									
PC ← PC + dst									
Range: +127, −128									
EI			9F	—	—	—	—	—	—
IMR (7) ← 1									
HALT			7F						
INC dst	r		rE	—	*	*	*	—	—
dst ← dst + 1			r = 0 − F						
	R		20						
	IR		21						
INCW dst	RR		A0	—	*	*	*	—	—
dst ← dst + 1	IR		A1						
IRET			BF	*	*	*	*	*	*
FLAGS ← @SP; SP ← SP + 1									
PC ← @SP; SP ← SP + 2; IMR (7) ← 1									
RLC dst 	R		10	*	*	*	*	—	—
	IR		11						
RR dst 	R		E0	*	*	*	*	—	—
	IR		E1						
RRC dst 	R		C0	*	*	*	*	—	—
	IR		C1						
SBC dst,src	(Note 1)		3□	*	*	*	*	1	*
dst ← dst ← src ← C									
SCF			DF	1	—	—	—	—	—
C ← 1									
SRA dst 	R		D0	*	*	*	0	—	—
	IR		D1						
SRP src		Im	31	—	—	—	—	—	—
RP ← src									
STOP			6F						
SUB dst,src	(Note 1)		2□	*	*	*	*	1	*
dst ← dst ← src									
SWAP dst 	R		F0	X	*	*	X	—	—
	IR		F1						
TCM dst,src	(Note 1)		6□	—	*	*	0	—	—
(NOT dst) AND src									

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
LDC dst,src	r	Irr	C2	—	—	—	—	—	—
dst ← src	Irr	r	D2						
LDCI dst,src	Ir	Irr	C3	—	—	—	—	—	—
dst ← src	Irr	Ir	D3						
r ← r + 1; rr ← rr + 1									
LDE dst,src	r	Irr	82	—	—	—	—	—	—
dst ← src	Irr	r	92						
LDEI dst,src	Ir	Irr	83	—	—	—	—	—	—
dst ← src	Irr	Ir	93						
r ← r + 1; rr ← rr + 1									
NOP			FF	—	—	—	—	—	—
OR dst,src	(Note 1)		4□	—	*	*	0	—	—
dst ← dst OR src									
POP dst	R		50	—	—	—	—	—	—
dst ← @SP;	IR		51						
SP ← SP + 1									
PUSH src		R	70	—	—	—	—	—	—
SP ← SP − 1; @SP ← src		IR	71						
RCF			CF	0	—	—	—	—	—
C ← 0									
RET			AF	—	—	—	—	—	—
PC ← @SP; SP ← SP + 2									
RL dst 	R		90	*	*	*	*	—	—
	IR		91						
TM dst,src	(Note 1)		7□	—	*	*	0	—	—
dst AND src									
XOR dst,src	(Note 1)		B□	—	*	*	0	—	—
dst ← dst XOR src									

NOTE: These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the left of the applicable addressing mode pair.

For example, the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

Addr Mode		Lower Opcode Nibble
dst	src	
r	r	2
r	Ir	3
R	R	4
R	IR	5
R	IM	6
IR	IM	7

REGISTERS

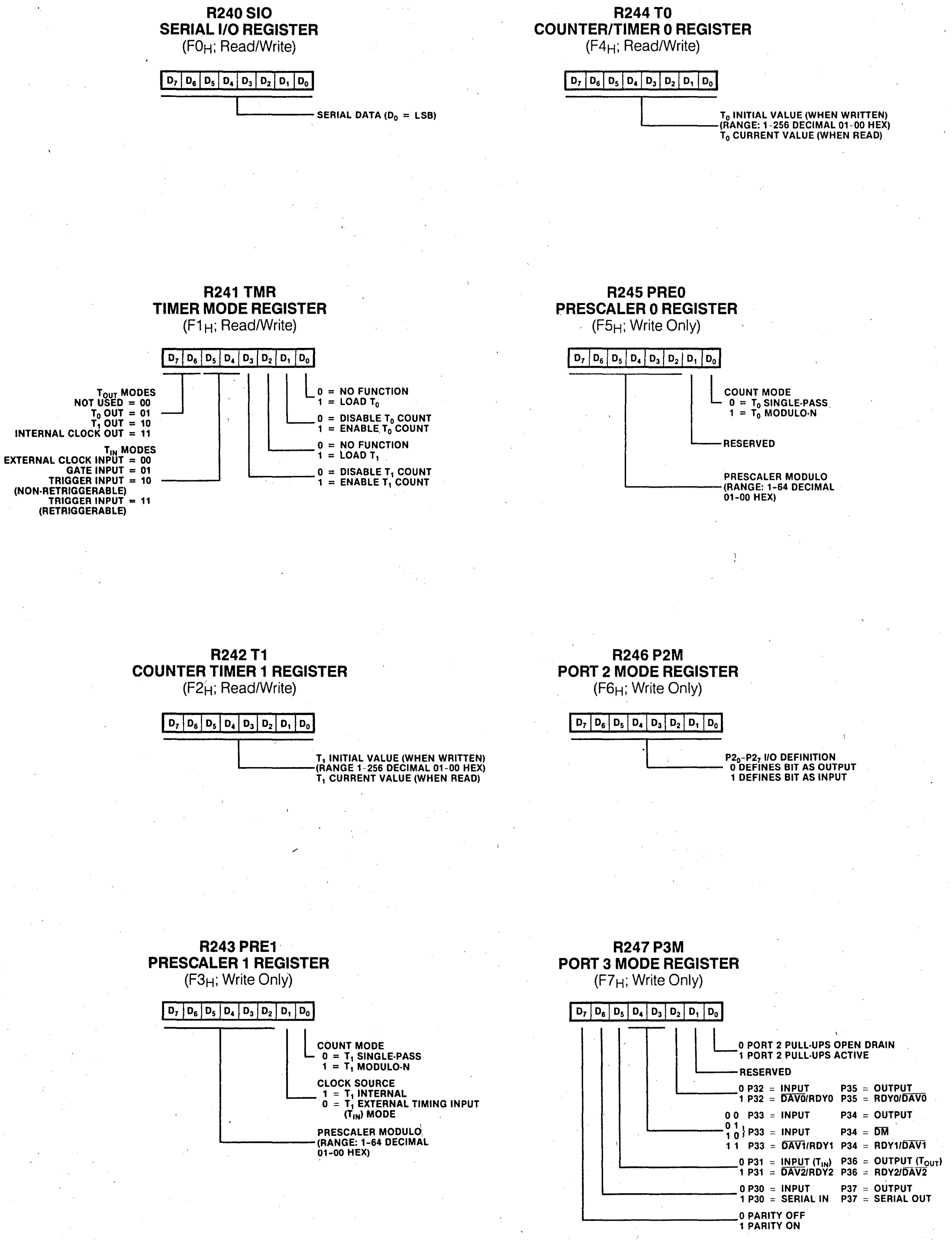
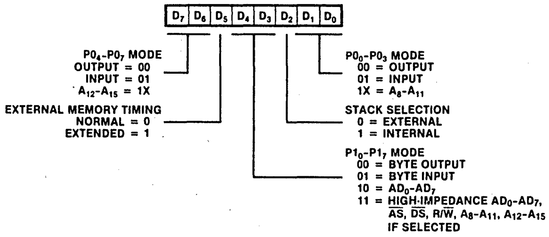

R240 SIO
SERIAL I/O REGISTER
(F0$_H$; Read/Write)
D$_7$ D$_6$ D$_5$ D$_4$ D$_3$ D$_2$ D$_1$ D$_0$
SERIAL DATA (D$_0$ = LSB)
R244 T0
COUNTER/TIMER 0 REGISTER
(F4$_H$; Read/Write)
D$_7$ D$_6$ D$_5$ D$_4$ D$_3$ D$_2$ D$_1$ D$_0$
T$_0$ INITIAL VALUE (WHEN WRITTEN)
(RANGE: 1-256 DECIMAL 01-00 HEX)
T$_0$ CURRENT VALUE (WHEN READ)
R241 TMR
TIMER MODE REGISTER
(F1$_H$; Read/Write)
D$_7$ D$_6$ D$_5$ D$_4$ D$_3$ D$_2$ D$_1$ D$_0$
T$_{OUT}$ MODES
NOT USED = 00
T$_0$ OUT = 01
T$_1$ OUT = 10
INTERNAL CLOCK OUT = 11
T$_{IN}$ MODES
EXTERNAL CLOCK INPUT = 00
GATE INPUT = 01
TRIGGER INPUT = 10
(NON-RETRIGGERABLE)
TRIGGER INPUT = 11
(RETRIGGERABLE)
0 = NO FUNCTION
1 = LOAD T$_0$
0 = DISABLE T$_0$ COUNT
1 = ENABLE T$_0$ COUNT
0 = NO FUNCTION
1 = LOAD T$_1$
0 = DISABLE T$_1$ COUNT
1 = ENABLE T$_1$ COUNT
R245 PRE0
PRESCALER 0 REGISTER
(F5$_H$; Write Only)
D$_7$ D$_6$ D$_5$ D$_4$ D$_3$ D$_2$ D$_1$ D$_0$
COUNT MODE
0 = T$_0$ SINGLE-PASS
1 = T$_0$ MODULO-N
RESERVED
PRESCALER MODULO
(RANGE: 1-64 DECIMAL
01-00 HEX)
R242 T1
COUNTER TIMER 1 REGISTER
(F2$_H$; Read/Write)
D$_7$ D$_6$ D$_5$ D$_4$ D$_3$ D$_2$ D$_1$ D$_0$
T$_1$ INITIAL VALUE (WHEN WRITTEN)
(RANGE 1-256 DECIMAL 01-00 HEX)
T$_1$ CURRENT VALUE (WHEN READ)
R246 P2M
PORT 2 MODE REGISTER
(F6$_H$; Write Only)
D$_7$ D$_6$ D$_5$ D$_4$ D$_3$ D$_2$ D$_1$ D$_0$
P2$_0$-P2$_7$ I/O DEFINITION
0 DEFINES BIT AS OUTPUT
1 DEFINES BIT AS INPUT
R243 PRE1
PRESCALER 1 REGISTER
(F3$_H$; Write Only)
D$_7$ D$_6$ D$_5$ D$_4$ D$_3$ D$_2$ D$_1$ D$_0$
COUNT MODE
0 = T$_1$ SINGLE-PASS
1 = T$_1$ MODULO-N
CLOCK SOURCE
1 = T$_1$ INTERNAL
0 = T$_1$ EXTERNAL TIMING INPUT
(T$_{IN}$) MODE
PRESCALER MODULO
(RANGE: 1-64 DECIMAL
01-00 HEX)
R247 P3M
PORT 3 MODE REGISTER
(F7$_H$; Write Only)
D$_7$ D$_6$ D$_5$ D$_4$ D$_3$ D$_2$ D$_1$ D$_0$
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
RESERVED
0 P32 = INPUT P35 = OUTPUT
1 P32 = $\overline{DAV0}$/RDY0 P35 = RDY0/$\overline{DAV0}$
0 0 P33 = INPUT P34 = OUTPUT
0 1 } 1 0 P33 = INPUT P34 = $\overline{DM}$
1 1 P33 = $\overline{DAV1}$/RDY1 P34 = RDY1/$\overline{DAV1}$
0 P31 = INPUT (T$_{IN}$) P36 = OUTPUT (T$_{OUT}$)
1 P31 = $\overline{DAV2}$/RDY2 P36 = RDY2/$\overline{DAV2}$
0 P30 = INPUT P37 = OUTPUT
1 P30 = SERIAL IN P37 = SERIAL OUT
0 PARITY OFF
1 PARITY ON

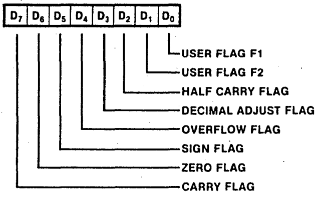
Figure 11. Control Registers

REGISTERS (Continued)

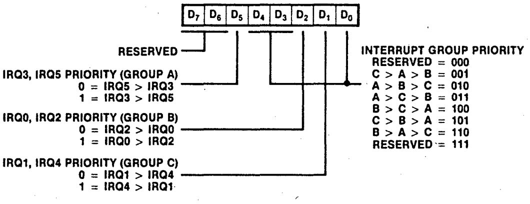
R248 P01M
PORT 0 AND 1 MODE REGISTER
($F8_H$; Write Only)



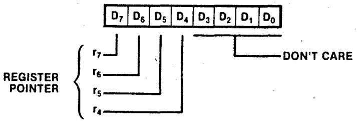
R252 FLAGS
FLAG REGISTER
(FC_H; Read/Write)



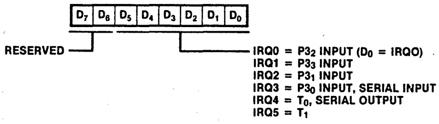
R249 IPR
INTERRUPT PRIORITY REGISTER
($F9_H$; Write Only)



R253 RP
REGISTER POINTER
(FD_H; Read/Write)

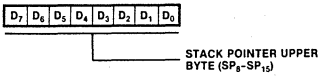


R250 IRQ
INTERRUPT REQUEST REGISTER
(FA_H; Read/Write)



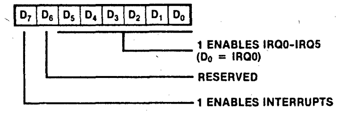
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
RESERVED
IRQ0 = $P3_2$ INPUT (D_0 = IRQ0)
IRQ1 = $P3_3$ INPUT
IRQ2 = $P3_1$ INPUT
IRQ3 = $P3_0$ INPUT, SERIAL INPUT
IRQ4 = T_0, SERIAL OUTPUT
IRQ5 = T_1

R254 SPH
STACK POINTER
(FE_H; Read/Write)



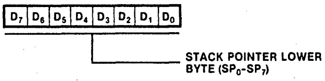
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
STACK POINTER UPPER BYTE (SP_8-SP_{15})

R251 IMR
INTERRUPT MASK REGISTER
(FB_H; Read/Write)



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
1 ENABLES IRQ0-IRQ5 (D_0 = IRQ0)
RESERVED
1 ENABLES INTERRUPTS

R255 SPL
STACK POINTER
(FF_H; Read/Write)



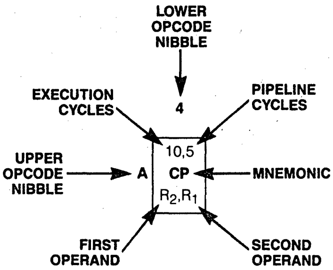
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
STACK POINTER LOWER BYTE (SP_0-SP_7)

Figure 11. Control Registers (Continued)

OPCODE MAP

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6.5 **DEC** R_1	6.5 **DEC** IR_1	6.5 **ADD** r_1,r_2	6.5 **ADD** r_1,Ir_2	10.5 **ADD** R_2,R_1	10.5 **ADD** IR_2,R_1	10.5 **ADD** R_1,IM	10.5 **ADD** IR_1,IM	6.5 **LD** r_1,R_2	6.5 **LD** r_2,R_1	12/10.5 **DJNZ** r_1,RA	12/10.0 **JR** cc,RA	6.5 **LD** r_1,IM	12/10.0 **JP** cc,DA	6.5 **INC** r1	
1	6.5 **RLC** R_1	6.5 **RLC** IR_1	6.5 **ADC** r_1,r_2	6.5 **ADC** r_1,Ir_2	10.5 **ADC** R_2,R_1	10.5 **ADC** IR_2,R_1	10.5 **ADC** R_1,IM	10.5 **ADC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
2	6.5 **INC** R_1	6.5 **INC** IR_1	6.5 **SUB** r_1,r_2	6.5 **SUB** r_1,Ir_2	10.5 **SUB** R_2,R_1	10.5 **SUB** IR_2,R_1	10.5 **SUB** R_1,IM	10.5 **SUB** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
3	8.0 **JP** IRR_1	6.1 **SRP** IM	6.5 **SBC** r_1,r_2	6.5 **SBC** r_1,Ir_2	10.5 **SBC** R_2,R_1	10.5 **SBC** IR_2,R_1	10.5 **SBC** R_1,IM	10.5 **SBC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
4	8.5 **DA** R_1	8.5 **DA** IR_1	6.5 **OR** r_1,r_2	6.5 **OR** r_1,Ir_2	10.5 **OR** R_2,R_1	10.5 **OR** IR_2,R_1	10.5 **OR** R_1,IM	10.5 **OR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
5	10.5 **POP** R_1	10.5 **POP** IR_1	6.5 **AND** r_1,r_2	6.5 **AND** r_1,Ir_2	10.5 **AND** R_2,R_1	10.5 **AND** IR_2,R_1	10.5 **AND** R_1,IM	10.5 **AND** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
6	6.5 **COM** R_1	6.5 **COM** IR_1	6.5 **TCM** r_1,r_2	6.5 **TCM** r_1,Ir_2	10.5 **TCM** R_2,R_1	10.5 **TCM** IR_2,R_1	10.5 **TCM** R_1,IM	10.5 **TCM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	6,0 **STOP**
7	10/12.1 **PUSH** R_2	12/14.1 **PUSH** IR_2	6.5 **TM** r_1,r_2	6.5 **TM** r_1,Ir_2	10.5 **TM** R_2,R_1	10.5 **TM** IR_2,R_1	10.5 **TM** R_1,IM	10.5 **TM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	7,0 **HALT**
8	10.5 **DECW** RR_1	10.5 **DECW** IR_1	12.0 **LDE** r_1,Irr_2	18.0 **LDEI** Ir_1,Irr_2					↓	↓	↓	↓	↓	↓	↓	6.1 **DI**
9	6.5 **RL** R_1	6.5 **RL** IR_1	12.0 **LDE** r_2,Irr_1	18.0 **LDEI** Ir_2,Irr_1					↓	↓	↓	↓	↓	↓	↓	6.1 **EI**
A	10.5 **INCW** RR_1	10.5 **INCW** IR_1	6.5 **CP** r_1,r_2	6.5 **CP** r_1,Ir_2	10.5 **CP** R_2,R_1	10.5 **CP** IR_2,R_1	10.5 **CP** R_1,IM	10.5 **CP** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	14.0 **RET**
B	6.5 **CLR** R_1	6.5 **CLR** IR_1	6.5 **XOR** r_1,r_2	6.5 **XOR** r_1,Ir_2	10.5 **XOR** R_2,R_1	10.5 **XOR** IR_2,R_1	10.5 **XOR** R_1,IM	10.5 **XOR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	16.0 **IRET**
C	6.5 **RRC** R_1	6.5 **RRC** IR_1	12.0 **LDC** r_1,Irr_2	18.0 **LDCI** Ir_1,Irr_2				10.5 **LD** r_1,x,R_2	↓	↓	↓	↓	↓	↓	↓	6.5 **RCF**
D	6.5 **SRA** R_1	6.5 **SRA** IR_1	12.0 **LDC** r_2,Irr_1	18.0 **LDCI** Ir_2,Irr_1	20.0 **CALL*** IRR_1		20.0 **CALL** DA	10.5 **LD** r_2,x,R_1	↓	↓	↓	↓	↓	↓	↓	6.5 **SCF**
E	6.5 **RR** R_1	6.5 **RR** IR_1		6.5 **LD** r_1,IR_2	10.5 **LD** R_2,R_1	10.5 **LD** IR_2,R_1	10.5 **LD** R_1,IM	10.5 **LD** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	6.5 **CCF**
F	8.5 **SWAP** R_1	8.5 **SWAP** IR_1		6.5 **LD** Ir_1,r_2		10.5 **LD** R_2,IR_1			↓	↓	↓	↓	↓	↓	↓	6.0 **NOP**
Bytes per Instruction	2	2	2	2	3	3	3	3	2	2	2	2	2	3	1	1



Legend:
R = 8-bit address
r = 4-bit address
R_1 or r_1 = Dst address
R_2 or r_2 = Src address

Sequence:
Opcode, First Operand, Second Operand

NOTE: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

ABSOLUTE MAXIMUM RATINGS

Voltages on all pins with respect
to GND . . . . −0.3V to +7.0V
Operating Ambient
Temperature . . . . .See Ordering Information
Storage Temperature . . . . −65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin.

Standard conditions are as follows:

- **+ 4.5V ≤ Vcc ≤ +5.5V**
- GND = 0V
- 0°C ≤ T_A ≤ +70°C **for S (Standard Temperature)**

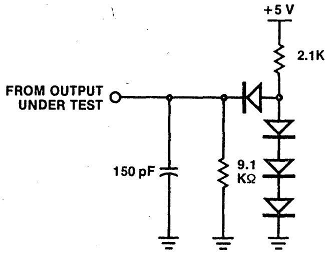


Figure 12. Test Load 1

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8	V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	−0.3	0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	**2.4**	V_{CC}	V	
V_{IL}	Input Low Voltage	−0.3	0.8	V	
V_{RH}	Reset Input High Voltage	3.8	V_{CC}	V	
V_{RL}	Reset Input Low Voltage	−0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −250 μA
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = +2.0 mA
I_{IL}	Input Leakage	−10	10	μA	0V ≤ V_{IN} ≤ + 5.25V
I_{OL}	Output Leakage	−10	10	μA	0V ≤ V_{IN} ≤ + 5.25V
I_{IR}	Reset Input Current		**80**	μA	V_{CC} = +5.25V, V_{RL} = 0V
I_{CC}	Supply Current		**50**	mA	All outputs and I/O pins floating
I_{CC1}	Standby Current		**3**	mA	Halt Mode
I_{CC2}	Standby Current		**10**	μA	Stop Mode

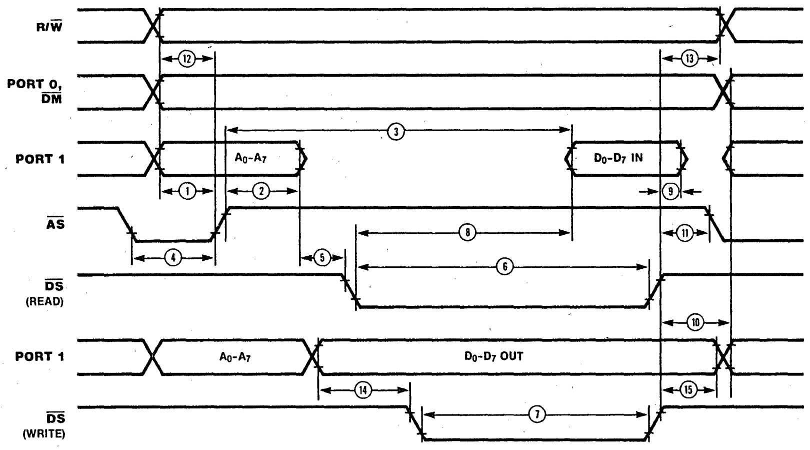


Figure 13. External I/O or Memory Read/Write

AC CHARACTERISTICS

External I/O or Memory Read and Write Timing

			12 MHz		16 MHz		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes
1	TdA(AS)	Address Valid to $\overline{AS}$ ↑Delay	35		20		2,3
2	TdAS(A)	$\overline{AS}$ ↑ to Address Float Delay	45		30		2,3
3	TdAS(DR)	$\overline{AS}$ ↑ to Read Data Required Valid		220		180	1,2,3
4	TwAS	AS Low Width	55		35		2,3
5	TdAz(DS)	Address Float to $\overline{DS}$ ↓	0		0		
6	TwDSR	$\overline{DS}$ (Read) Low Width	185		135		1,2,3
7	TwDSW	$\overline{DS}$ (Write) Low Width	110		80		1,2,3
8	TdDSR(DR).	$\overline{DS}$ ↓ to Read Data Required Valid		130		75	1,2,3
9	ThDR(DS)	Read Data to $\overline{DS}$ ↑ Hold Time	0		0		2,3
10	TdDS(A)	$\overline{DS}$ ↑ to Address Active Delay	45		35		2,3
11	TdDS(AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓Delay	55		25		2,3
12	TdR/W(AS)	R/$\overline{W}$ Valid to $\overline{AS}$ ↑ Delay	30		20		2,3
13	TdDS(R/W)	$\overline{DS}$ ↑ to R/W Not Valid	35		25		2,3
14	TdDW(DSW)	Write Data Valid to $\overline{DS}$ (Write) ↓ Delay	35		25		2,3
15	TdDS(DW)	$\overline{DS}$ ↑ to Write Data Not Valid Delay	35		25		2,3
16	TdA(DR)	Address Valid to Read Data Required Valid	255		200		1,2,3
17	TdAS(DS)	AS to DS Delay	55		40		2,3

NOTES:

1. Delay times given are for **a 16 MHz** crystal input frequency. For lower frequencies, the change in clock periods must be added to the delay time.
2. Data Strobe Width is given for **a 16 MHz** crystal input frequency. For lower frequencies the change in three clock periods must be added to obtain the minimum width. The Data Strobe Width varies according to the instruction being executed.
3. Address Strobe and Data Strobe to Data In Valid delay times represent memory system access times and are given for **a 16 MHz** crystal input frequency. For lower frequencies, the change in four clock periods must be added to TdAS (DI) and the change in three clock periods added to TdDS(DI).

* All units in nanoseconds (ns).
† Test Load 1
° All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".

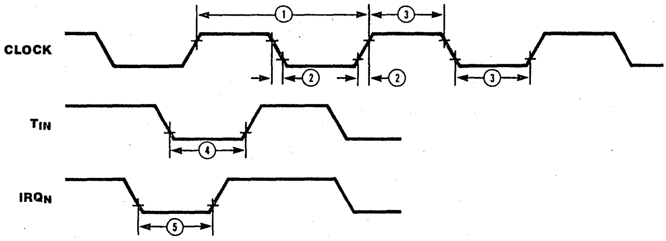


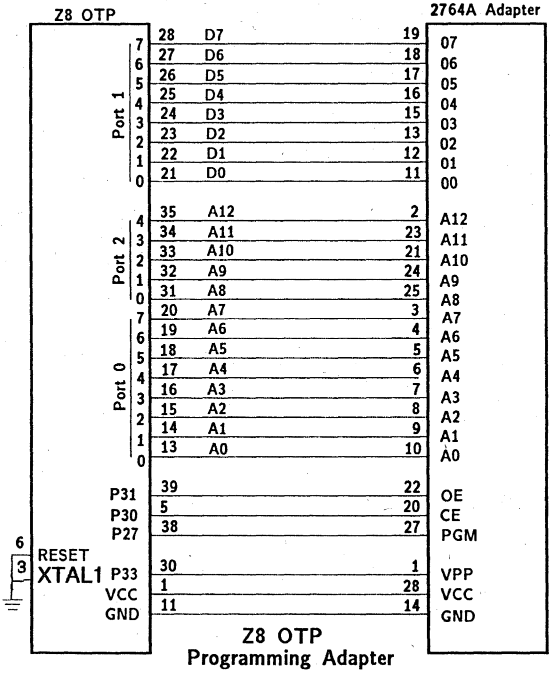
Figure 14. Additional Timing

AC CHARACTERISTICS

Additional Timing Table

Number	Symbol	Parameter			12 MHz Min	12 MHz Max	16 MHz Min	16 MHz Max	Notes
1	TpC	Input Clock Period			83	1000	62.5	1000	1
2	TrC,TfC	Clock Input Rise and Fall Times				15		10	1
3	TwC	Input Clock Width			70		21		1
4	TwTinL	Timer Input Low Width			70		50		2
5	TwTinH	Timer Input High Width			3TpC		3TpC		2
6	TpTin	Timer Input Period	8TpC		8TpC		8TpC		2
7	TrTin,TfTin	Timer Input Rise and Fall Times		100		100		100	2
8A	TwIL	Interrupt Request Input Low Time	100		70		50		2,4
8B	TwIL	Interrupt Request Input Low Time	3TpC		3TpC		3TpC		2,5
9	TwIH	Interrupt Request Input High Time	3TpC		3TpC		3TpC		2,3

NOTES:
1. Clock timing references use 3.8V for a logic "1" and 0.8V for a logic "0".
2. Timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
3. Interrupt request via Port 3.
* Units in nanoseconds (ns).



Z8 OTP
Programming Adapter

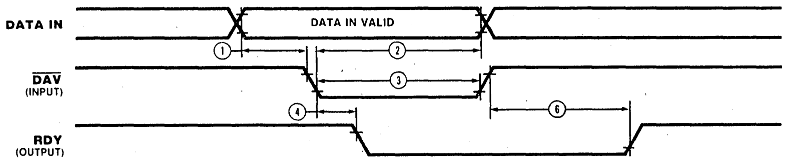


Figure 15a. Input Handshake Timing

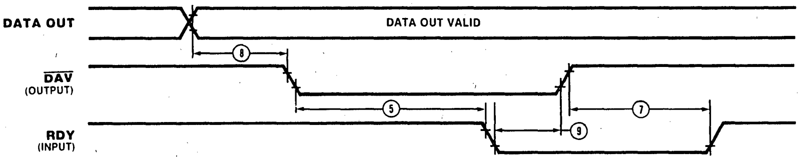


Figure 15b. Output Handshake Timing

AC CHARACTERISTICS

Handshake Timing

			12 MHz		16 MHz		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes
1	TsDI(DAV)	Data In Setup Time	0		0		
2	ThDI(DAV)	Data In Hold Time	160		145		
3	TwDAV	Data Available Width	120		110		
4	TdDAVIf(RDY)	DAV ↓ Input to RDY ↓ Delay		120		115	1,2
5	TdDAVOf(RDY)	DAV ↓ Output to RDY ↓ Delay	0		0		1,3
6	TdDAVIr(RDY)	DAV ↑ Input to RDY ↑ Delay		120		115	1,2
7	TdDAVOr(RDY)	DAV ↑ Output to RDY ↑ Delay	0		0		1,3
8	TdDO(DAV)	Data Out to DAV ↓ Delay	30		30		1
9	TdRDY(DAV)	Rdy ↓ Input to DAV ↑ Delay	0	140	0	130	1

NOTES:
1. Test load 1
2. Input handshake
3. Output handshake
† All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
* Units in nanoseconds (ns).

Zilog **Product Specification**

November 1987

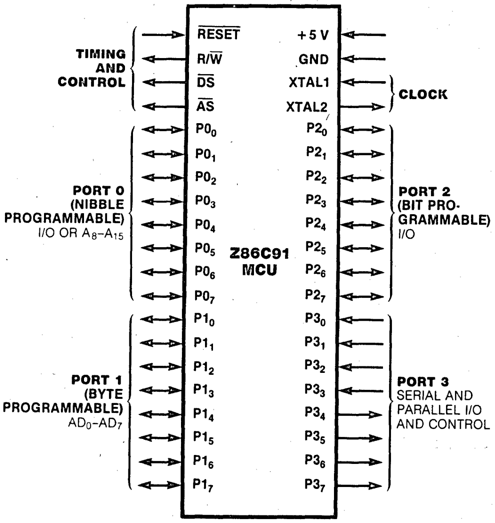
Z86C91 CMOS ROMless Z8® Microcomputer

FEATURES

- □ Complete microcomputer, 24 I/O lines, and up to 64K bytes of addressable external space each for program and data memory.
- □ 256-byte register file, including 236 general-purpose registers, 3 I/O port registers, and 16 status and control registers.
- □ Vectored, priority interrupts for I/O, counter/timers, and UART.
- □ On-chip oscillator that accepts crystal or external clock drive.
- □ Full-duplex UART and two programmable 8-bit counter/timers, each with a 6-bit programmable prescaler.
- □ Register Pointer so that short, fast instructions can access any one of the sixteen working-register groups.
- □ Single +5V power supply—all I/O pins TTL compatible.
- □ **12 and 16 MHz**
- □ CMOS process
- □ Standby modes—Halt and Stop

GENERAL DESCRIPTION

The Z86C91 is a CMOS ROMless version of the Z8 single-chip microcomputer. It offers all the outstanding features of the Z8 family architecture except an on-chip program ROM. Use of external memory rather than a preprogrammed ROM enables this Z8 microcomputer to be used in low volume applications or where code flexibility is required.



TIMING AND CONTROL
RESET
R/W
DS
AS
+5 V
GND
XTAL1
XTAL2
CLOCK
PORT 0 (NIBBLE PROGRAMMABLE) I/O OR A_8-A_{15}
$P0_0$ $P0_1$ $P0_2$ $P0_3$ $P0_4$ $P0_5$ $P0_6$ $P0_7$
PORT 1 (BYTE PROGRAMMABLE) AD_0-AD_7
$P1_0$ $P1_1$ $P1_2$ $P1_3$ $P1_4$ $P1_5$ $P1_6$ $P1_7$
Z86C91 MCU
$P2_0$ $P2_1$ $P2_2$ $P2_3$ $P2_4$ $P2_5$ $P2_6$ $P2_7$
PORT 2 (BIT PROGRAMMABLE) I/O
$P3_0$ $P3_1$ $P3_2$ $P3_3$ $P3_4$ $P3_5$ $P3_6$ $P3_7$
PORT 3 SERIAL AND PARALLEL I/O AND CONTROL

Figure 1. Pin Functions

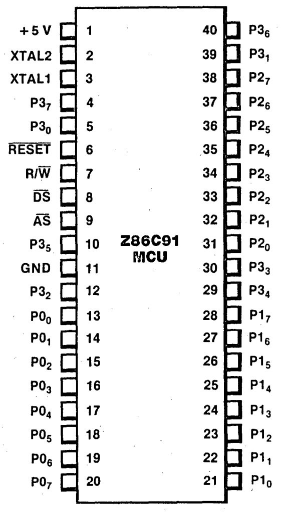


Figure 2a. 40-pin Dual-In-Line Package (DIP), Pin Assignments

The Z86C91 can provide up to 16 output address lines, thus permitting an address space of up to 64K bytes of data or program memory. Eight address outputs (AD_0-AD_7) are provided by a multiplexed, 8-bit, Address/Data bus. The remaining 8 bits can be provided by the software configuration of Port 0 to output address bits A_8-A_{15}.

Available address space can be doubled (up to 128K bytes) by programming bit 4 of Port 3 ($P3_4$) to act as a data memory select output ($\overline{DM}$). The two states of $\overline{DM}$ together with the 16 address outputs can define separate data and memory address spaces of up to 64K bytes each.

There are 256 bytes of RAM located on-chip and organized as a register file of 236 general-purpose registers, 16 control and status registers, and three I/O port registers. This register file can be divided into sixteen groups of 16 working registers each. Configuring the register file in this manner allows the use of short format instructions; in addition, any of the individual registers can be accessed directly.

The pin functions and the pin assignments of the Z86C91 package are illustrated in Figures 1 and 2.

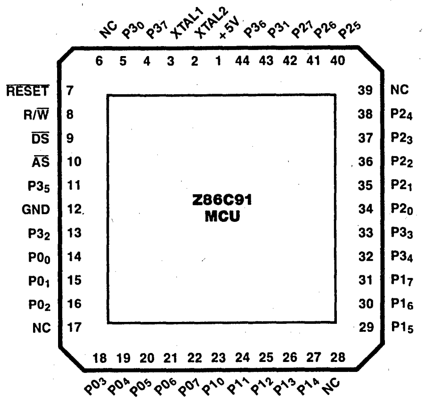


Figure 2b. 44-pin Chip Carrier, Pin Assignments

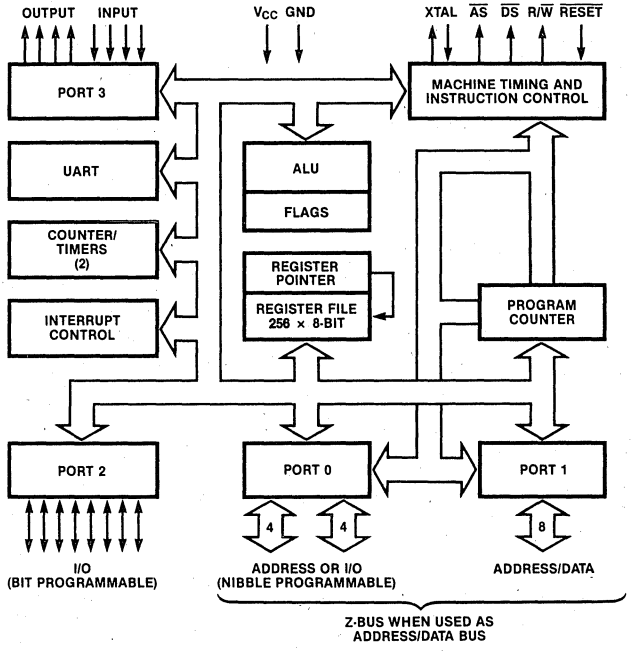


Figure 3. Functional Block Diagram

ARCHITECTURE

Architecture is characterized by a flexible I/O scheme, an efficient register and address space structure and a number of ancillary features that are helpful in many applications.

Microcomputer applications demand powerful I/O capabilities. The Z86C91 fulfills this with 24 pins available for input and output. These lines are grouped into three ports of eight lines each and are configurable under software control to provide timing, status signals, serial or parallel I/O with or without handshake, and an address bus for interfacing external memory.

Three basic address spaces are available: program memory, data memory and the register file (internal). The 256-byte random-access register file is composed of 236 general-purpose registers, three I/O port registers, and 16 control and status registers.

To unburden the program from coping with real-time problems such as serial data communication and counting/timing, an asynchronous receiver/transmitter (UART) and two counter/timers with a large number of user-selectable modes are offered on-chip. Hardware support for the UART is minimized because one of the on-chip timers supplies the bit rate. Figure 3 shows the block diagram.

POWER DOWN INSTRUCTIONS

The Z86C91 has two instructions to reduce power consumption during standby operation. HALT turns off the processor and UART while the counter/timers and external interrupts IRQ0, IRQ1, and IRQ2 remain active.

When an interrupt occurs the processor resumes execution after servicing the interrupt. STOP turns off the clock to the entire Z86C91 and reduces the standby current to 10 microamps. The stop mode is terminated by reset, which causes the processor to restart the application program at address 12.

PIN DESCRIPTION

$\overline{\text{AS}}$. *Address Strobe* (output, active Low). Address Strobe is pulsed once at the beginning of each machine cycle. Addresses output via Port 1 for all external program or data memory transfers are valid at the trailing edge of $\overline{\text{AS}}$.

$\overline{\text{DS}}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each external memory transfer.

$P0_0$-$P0_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. *I/O Port Lines* (input/outputs, TTL-compatible). These 24 lines are divided into three 8-bit I/O ports that can be configured under program control for I/O or external memory interface (Figure 3).

$P1_0$-$P1_7$. *Address/Data Port* (bidirectional). Multiplexed address (A_0-A_7) and data (D_0-D_7) lines used to interface with program and data memory.

$\overline{\text{RESET}}$. *Reset* (input, active Low). $\overline{\text{RESET}}$ initializes the Z86C91. After RESET the MCU is in the extended memory mode. When $\overline{\text{RESET}}$ is deactivated, program execution begins from program location $000C_H$.

R/$\overline{\text{W}}$. *Read/Write* (output). R/$\overline{\text{W}}$ is Low when the Z86C91 is writing to external program or data memory.

XTAL1, XTAL2. *Crystal 1, Crystal 2* (time-base input and output). These pins connect a parallel-resonant crystal to the on-chip clock oscillator and buffer.

ADDRESS SPACES

Program Memory. The Z86C91 addresses 64K bytes of external program memory space (Figure 4).

The first 12 bytes of program memory are reserved for the interrupt vectors. These locations contain six 16-bit vectors that correspond to the six available interrupts. Program execution begins at location $000C_H$ after a reset.

Data Memory. The Z86C91 can address 64K bytes of external data memory. External data memory may be included with or separated from the external program memory space. $\overline{\text{DM}}$, an optional I/O function that can be programmed to appear on pin $P3_4$, is used to distinguish between data and program memory space.

Register File. The 256-byte register file includes three I/O port registers (R0, R2, R3), 236 general-purpose registers (R4-R239) and 16 control and status registers (R240-R255). These registers are assigned the address locations shown in Figure 5.

Z86C91 instructions can access registers directly or indirectly with an 8-bit address field. This also allows short 4-bit register addressing using the Register Pointer (one of the control registers). In the 4-bit mode, the register file is divided into sixteen working-register groups, each occupying 16 contiguous locations (Figure 5). The Register Pointer addresses the starting location of the active working-register group (Figure 6).

Note: Register Bank E0-EF can only be accessed through working register and indirect addressing modes.

Stacks. Either the internal register file or the external data memory can be used for the stack. A 16-bit Stack Pointer (R254 and R255) is used for the external stack, which can reside anywhere in data memory. An 8-bit Stack Pointer (R255) is used for the internal stack that resides within the 236 general-purpose registers (R4-R239).

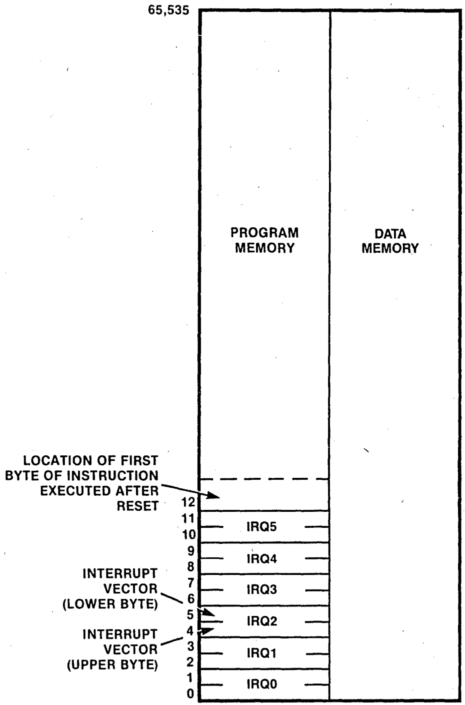

65,535
PROGRAM MEMORY
DATA MEMORY
LOCATION OF FIRST BYTE OF INSTRUCTION EXECUTED AFTER RESET
12
11
10
9
8
7
6
5
4
3
2
1
0
IRQ5
IRQ4
IRQ3
IRQ2
IRQ1
IRQ0
INTERRUPT VECTOR (LOWER BYTE)
INTERRUPT VECTOR (UPPER BYTE)

Figure 4. Z86C91 Program Memory Map

DECIMAL		HEX	IDENTIFIERS
255	STACK POINTER (BITS 7-0)	FF	SPL
254	STACK POINTER (BITS 15-8)	FE	SPH
253	REGISTER POINTER	FD	RP
252	PROGRAM CONTROL FLAGS	FC	FLAGS
251	INTERRUPT MASK REGISTER	FB	IMR
250	INTERRUPT REQUEST REGISTER	FA	IRQ
249	INTERRUPT PRIORITY REGISTER	F9	IPR
248	PORTS 0-1 MODE	F8	P01M
247	PORT 3 MODE	F7	P3M
246	PORT 2 MODE	F6	P2M
245	T0 PRESCALER	F5	PRE0
244	TIMER/COUNTER 0	F4	T0
243	T1 PRESCALER	F3	PRE1
242	TIMER/COUNTER 1	F2	T1
241	TIMER MODE	F1	TMR
240	SERIAL I/O	F0	SIO
239		EF	
	GENERAL-PURPOSE REGISTERS		
4		04	
3	PORT 3	03	P3
2	PORT 2	02	P2
1	PORT 1	01	P1
0	PORT 0	00	P0

Figure 5. The Register File

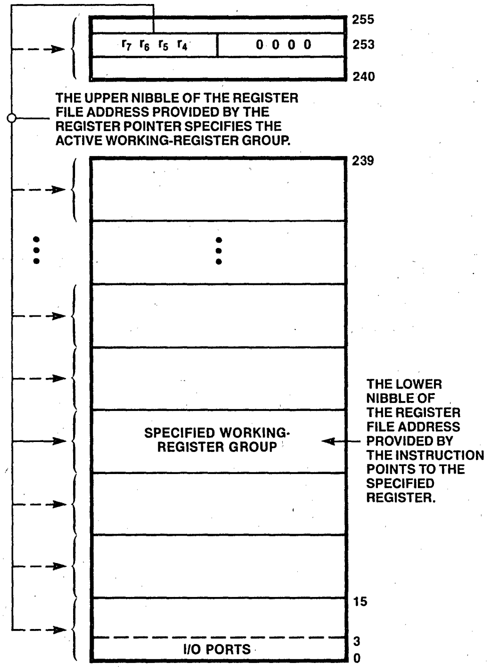

255
r_7 r_6 r_5 r_4
0 0 0 0
253
240
THE UPPER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE REGISTER POINTER SPECIFIES THE ACTIVE WORKING-REGISTER GROUP.
239
SPECIFIED WORKING-REGISTER GROUP
THE LOWER NIBBLE OF THE REGISTER FILE ADDRESS PROVIDED BY THE INSTRUCTION POINTS TO THE SPECIFIED REGISTER.
15
3
0
I/O PORTS

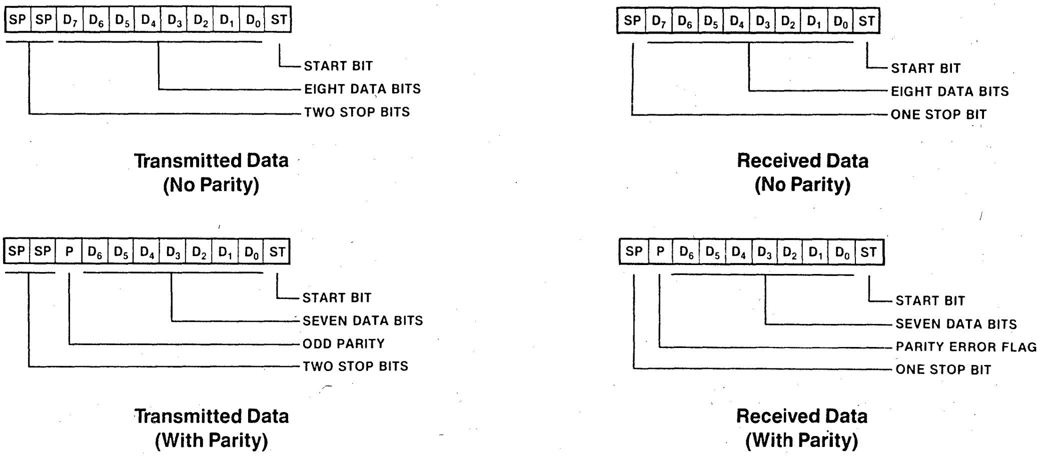
Figure 6. The Register Pointer

SERIAL INPUT/OUTPUT

Port 3 lines $P3_0$ and $P3_7$ can be programmed as serial I/O lines for full-duplex serial asynchronous receiver/transmitter operation. The bit rate is controlled by Counter/Timer 0, with a maximum rate of 93.75K bits/second at 12 MHz.

The Z86C91 automatically adds a start bit and two stop bits to transmitted data (Figure 7). Odd parity is also available as an option. Eight data bits are always transmitted, regardless of parity selection. If parity is enabled, the eighth data bit is used as the odd parity bit. An interrupt request (IRQ4) is generated on all transmitted characters.

Received data must have a start bit, eight data bits, and at least one stop bit. If parity is on, bit 7 of the received data is replaced by a parity error flag. Received characters generate the IRQ3 interrupt request.



Transmitted Data (No Parity)

Received Data (No Parity)

Transmitted Data (With Parity)

Received Data (With Parity)

Figure 7. Serial Data Formats

COUNTER/TIMERS

The Z86C91 contains two 8-bit programmable counter/timers (T_0 and T_1), each driven by its own 6-bit programmable prescaler. The T_1 prescaler can be driven by internal or external clock sources; however, the T_0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrements the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request—IRQ4 (T_0) or IRQ5 (T_1)—is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single-pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read any time without disturbing their value or count mode.

The clock source for T_1 is user-definable; it can be either the internal microprocessor clock divided by four, or an external signal input via Port 3. The Timer Mode register configures the external timer input as an external clock, a trigger input that can be retriggerable or nonretriggerable, or as a gate input for the internal clock. The counter/timers can be programmably cascaded by connecting the T_0 output to the input of T_1. Port 3 line $P3_6$ also serves as a timer output (T_{OUT}) through which T_0, T_1 or the internal clock can be output.

I/O PORTS

The Z86C91 has 24 lines available for input and output. These lines are grouped into three ports of eight lines each and are configurable as input, output or address. Under software control, the ports can be programmed to provide address outputs, timing, status signals, serial I/O, and parallel I/O with or without handshake. All ports have active pull-ups and pull-downs compatible with TTL loads.

Port 1 is a dedicated Z-BUS® compatible memory interface. The operations of Port 1 are supported by the Address Strobe ($\overline{AS}$) and Data Strobe ($\overline{DS}$) lines, and by the Read/Write (R/$\overline{W}$) and Data Memory ($\overline{DM}$) control lines. The low-order program and data memory addresses (A_0-A_7) are output through Port 1 (Figure 8) and are multiplexed with data in/out (D_0-D_7). Instruction fetch and data memory read/write operations are done through this port.

Port 1 cannot be used as a register nor can a handshake mode be used with this port.

The Z86C91 wakes up with the 8 bits of Port 1 configured as address outputs for external memory. If more than eight address lines are required, additional lines can be obtained by programming Port 0 bits as address bits. The least-significant four bits of Port 0 can be configured to supply address bits A_8-A_{11} for 4K byte addressing or both nibbles of Port 0 can be configured to supply address bits A_8-A_{15} for 64K byte addressing.

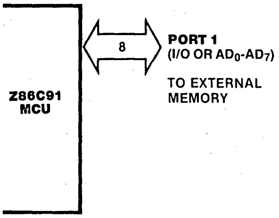


Figure 8. Port 1

Port 0 can be programmed as a nibble I/O port, or as an address port for interfacing external memory (Figure 9). When used as an I/O port, Port 0 can be placed under handshake control. In this configuration, Port 3 lines $P3_2$ and $P3_5$ are used as the handshake controls DAV_0 and RDY_0. Handshake signal assignment is dictated by the I/O direction of the upper nibble $P0_4$-$P0_7$.

For external memory references, Port 0 can provide address bits A_8-A_{11} (lower nibble) or A_8-A_{15} (lower and upper nibbles) depending on the required address space. If the address range requires 12 bits or less, the upper nibble of Port 0 can be programmed independently as I/O while the lower nibble is used for addressing.

Port 0 lines are configured as address lines A_8-A_{15} after a Reset. If one or both nibbles are needed for I/O operation, they must be configured by writing to the Port 0 Mode register.

To permit the use of slow memory, an automatic wait mode of two oscillator clock cycles is configured for bus timing after each reset. The initialization routine could include reconfiguration to eliminate this extended timing mode.

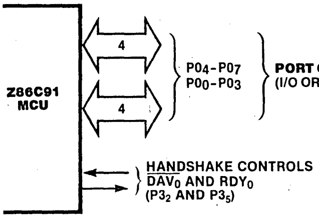


Figure 9. Port 0

Port 2 bits can be programmed independently as input or output (Figure 10). This port is always available for I/O operations. In addition, Port 2 can be configured to provide open-drain outputs.

Like Port 0, Port 2 may also be placed under handshake control. In this configuration, Port 3 lines $P3_1$ and $P3_6$ are used as the handshake controls lines $\overline{DAV_2}$ and RDY_2. The handshake signal assignment for Port 3 lines $P3_1$ and $P3_6$ is dictated by the direction (input or output) assigned to bit 7 of Port 2.

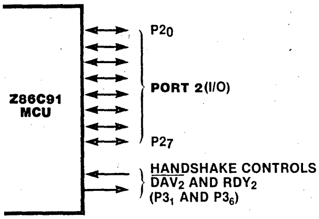


Figure 10. Port 2

Port 3 lines can be configured as I/O or control lines (Figure 11). In either case, the direction of the eight lines is fixed as four input ($P3_0$-$P3_3$) and four output ($P3_4$-$P3_7$). For serial I/O, lines $P3_0$ and $P3_7$ are programmed as serial in and serial out, respectively.

Port 3 can also provide the following control functions: handshake for Ports 0 and 2 ($\overline{DAV}$ and RDY); four external interrupt request signals (IRQ0-IRQ3); timer input and output signals (T_{IN} and T_{OUT}) and Data Memory Select ($\overline{DM}$).

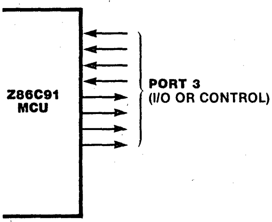


Figure 11. Port 3

INTERRUPTS

The Z86C91 allows six different interrupts from eight sources: the four Port 3 lines $P3_0$-$P3_3$, Serial In, Serial Out, and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask register globally or individually enables or disables the six interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All interrupts are vectored through locations in program memory. When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the Program Counter and status flags, and accesses the program memory vector location reserved for that interrupt. This memory location and the next byte contain the 16-bit address of the interrupt service routine for that particular interrupt request. The Z86C91 takes 26 system clock cycles to enter an interrupt subroutine.

Polled interrupt systems are also supported. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the Interrupt Request register polled to determine which of the interrupt requests needs service.

CLOCK

The on-chip oscillator has a high-gain, parallel-resonant amplifier for connection to a crystal or to any suitable external clock source (XTAL1 = Input, XTAL2 = Output).

The crystal source is connected across XTAL1 and XTAL2, using the recommended capacitance (C_L = 15 pf maximum) from each pin to ground. The specifications for the crystal are as follows:

- AT cut, parallel-resonant
- Fundamental type
- Series resistance, $R_S \leqslant 100\Omega$
- **16 MHz maximum**

INSTRUCTION SET NOTATION

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "←". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

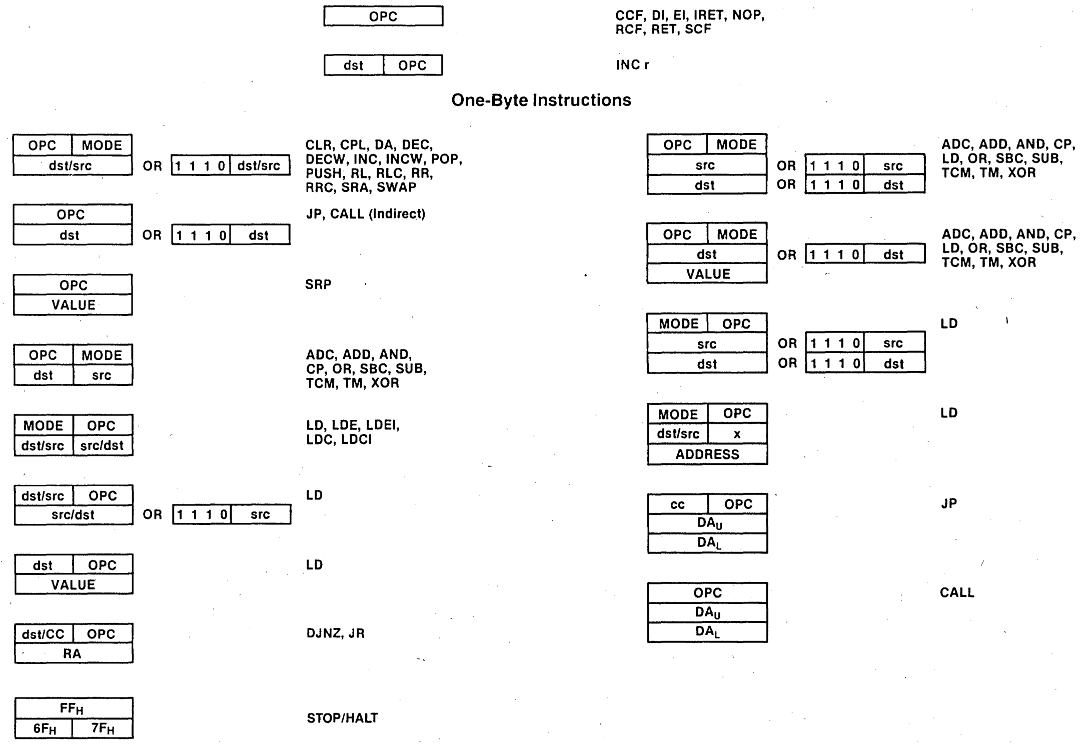
C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*****	Set or cleared according to operation
—	Unaffected
X	Undefined

CONDITION CODES

Value	Mnemonic	Meaning	Flags Set
1000		Always true	—
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	—


OPC
CCF, DI, EI, IRET, NOP, RCF, RET, SCF
dst | OPC
INC r
OPC | MODE / dst/src OR 1 1 1 0 | dst/src
CLR, CPL, DA, DEC, DECW, INC, INCW, POP, PUSH, RL, RLC, RR, RRC, SRA, SWAP
OPC / dst OR 1 1 1 0 | dst
JP, CALL (Indirect)
OPC / VALUE
SRP
OPC | MODE / dst | src
ADC, ADD, AND, CP, OR, SBC, SUB, TCM, TM, XOR
MODE | OPC / dst/src | src/dst
LD, LDE, LDEI, LDC, LDCI
dst/src | OPC / src/dst OR 1 1 1 0 | src
LD
dst | OPC / VALUE
LD
dst/CC | OPC / RA
DJNZ, JR
FFH / 6FH | 7FH
STOP/HALT
OPC | MODE / src OR 1 1 1 0 | src / dst OR 1 1 1 0 | dst
ADC, ADD, AND, CP, LD, OR, SBC, SUB, TCM, TM, XOR
OPC | MODE / dst OR 1 1 1 0 | dst / VALUE
ADC, ADD, AND, CP, LD, OR, SBC, SUB, TCM, TM, XOR
MODE | OPC / src OR 1 1 1 0 | src / dst OR 1 1 1 0 | dst
LD
MODE | OPC / dst/src | x / ADDRESS
LD
cc | OPC / DAU / DAL
JP
OPC / DAU / DAL
CALL

One-Byte Instructions

Two-Byte Instructions

Three-Byte Instructions

Figure 12. Instruction Formats

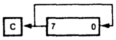
INSTRUCTION SUMMARY

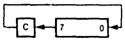
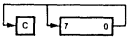
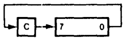
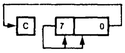
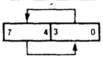
Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	Flags Affected C Z S V D H
ADC dst,src dst ← dst + src + C	(Note 1)		1□	* * * * 0 *
ADD dst,src dst ← dst + src	(Note 1)		0□	* * * * 0 *
AND dst,src dst ← dst AND src	(Note 1)		5□	— * * 0 — —
CALL dst SP ← SP − 2 @SP ← PC; PC ← dst	DA IRR		D6 D4	— — — — — —
CCF C ← NOT C			EF	* — — — — —
CLR dst dst ← 0	R IR		B0 B1	— — — — — —
COM dst dst ← NOT dst	R IR		60 61	— * * 0 — —
CP dst,src dst − src	(Note 1)		A□	* * * * — —
DA dst dst ← DA dst	R IR		40 41	* * * X — —
DEC dst dst ← dst − 1	R IR		00 01	— * * * — —
DECW dst dst ← dst − 1	RR IR		80 81	— * * * — —
DI IMR (7) ← 0			8F	— — — — — —

DJNZ r,dst
r ← r − 1
if r ≠ 0
PC ← PC + dst
Range: +127, −128
	RA		rA
			r = 0 − F
				— — — — — —

EI IMR (7) ← 1			9F	— — — — — —
HALT			7F	— — — — — —
INC dst dst ← dst + 1	r R IR		rE r = 0 − F 20 21	— * * * — —
INCW dst dst ← dst + 1	RR IR		A0 A1	— * * * — —

INSTRUCTION SUMMARY (Continued)

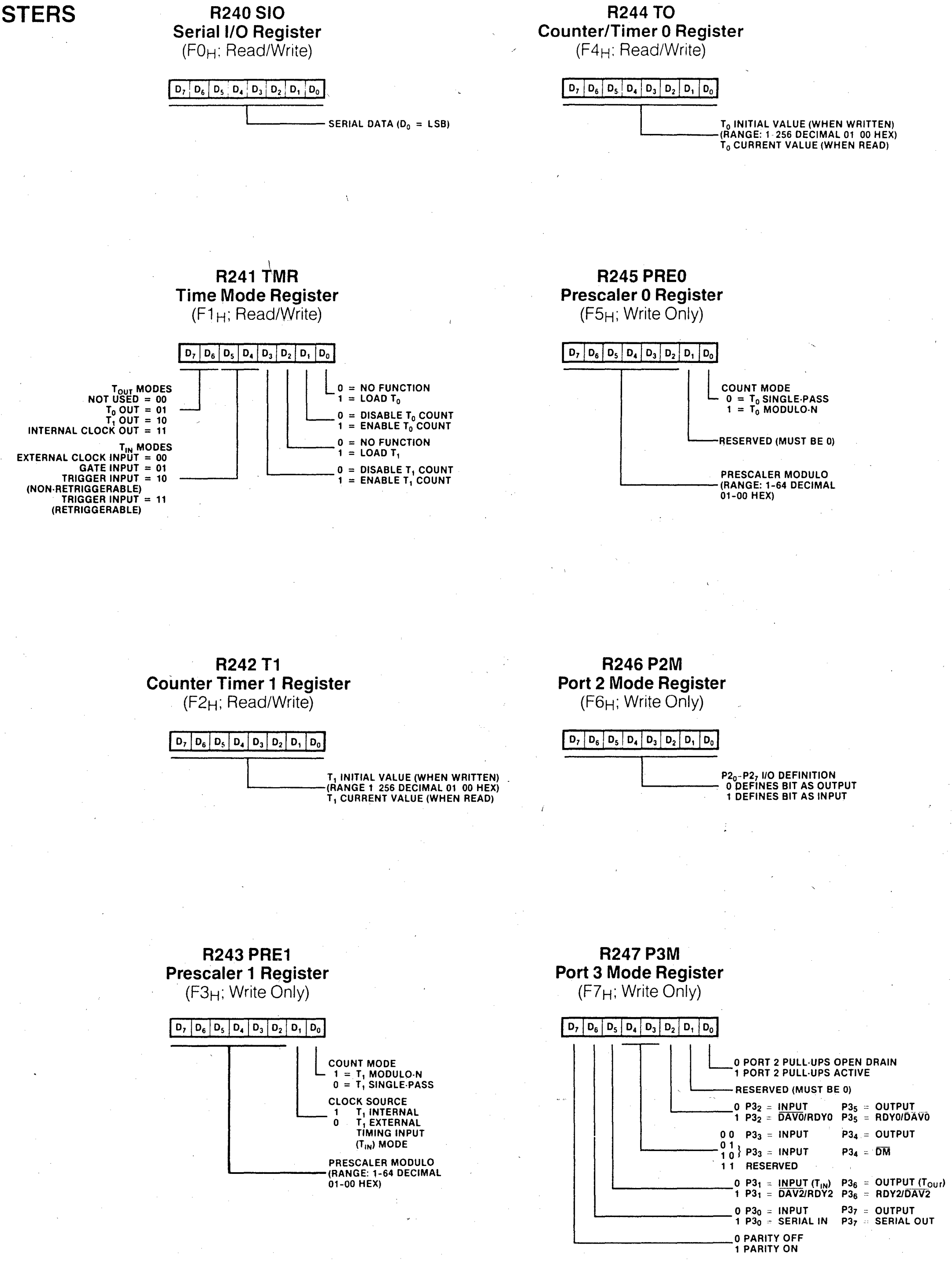
Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
IRET FLAGS ← @SP; SP ← SP + 1 PC ← @SP; SP ← SP + 2; IMR (7) ← 1			BF	*	*	*	*	*	*
JP cc,dst if cc is true PC ← dst	DA IRR		cD c = 0 – F 30	—	—	—	—	—	—
JR cc,dst if cc is true, PC ← PC + dst Range: +127, –128	RA		cB c = 0 – F	—	—	—	—	—	—
LD dst,src dst ← src	r r R r X r Ir R R R IR IR	Im R r X r Ir r R IR IM IM R	rC r8 r9 r = 0 – F C7 D7 E3 F3 E4 E5 E6 E7 F5	—	—	—	—	—	—
LDC dst,src dst ← src	r Irr	Irr r	C2 D2	—	—	—	—	—	—
LDCI dst,src dst ← src r ← r + 1; rr ← rr + 1	Ir Irr	Irr Ir	C3 D3	—	—	—	—	—	—
LDE dst,src dst ← src	r Irr	Irr r	82 92	—	—	—	—	—	—
LDEI dst,src dst ← src r ← r + 1; rr ← rr + 1	Ir Irr	Irr Ir	83 93	—	—	—	—	—	—
NOP			FF	—	—	—	—	—	—
OR dst,src dst ← dst OR src	(Note 1)		4□	—	*	*	0	—	—
POP dst dst ← @SP; SP ← SP + 1	R IR		50 51	—	—	—	—	—	—
PUSH src SP ← SP – 1; @SP ← src		R IR	70 71	—	—	—	—	—	—
RCF C ← 0			CF	0	—	—	—	—	—
RET PC ← @SP; SP ← SP + 2			AF	—	—	—	—	—	—
RL dst 	R IR		90 91	*	*	*	*	—	—

Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
RLC dst 	R IR		10 11	*	*	*	*	—	—
RR dst 	R IR		E0 E1	*	*	*	*	—	—
RRC dst 	R IR		C0 C1	*	*	*	*	—	—
SBC dst,src dst ← dst ← src ← C	(Note 1)		3□	*	*	*	*	1	*
SCF C ← 1			DF	1	—	—	—	—	—
SRA dst 	R IR		D0 D1	*	*	*	0	—	—
SRP src RP ← src		Im	31	—	—	—	—	—	—
STOP			6F	—	—	—	—	—	—
SUB dst,src dst ← dst ← src	(Note 1)		2□	*	*	*	*	1	*
SWAP dst 	R IR		F0 F1	X	*	*	X	—	—
TCM dst,src (NOT dst) AND src	(Note 1)		6□	—	*	*	0	—	—
TM dst,src dst AND src	(Note 1)		7□	—	*	*	0	—	—
XOR dst,src dst ← dst XOR src	(Note 1)		B□	—	*	*	0	—	—

NOTE: These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the left of the applicable addressing mode pair.

For example, the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

Addr Mode		Lower
dst	src	Opcode Nibble
r	r	2
r	Ir	3
R	R	4
R	IR	5
R	IM	6
IR	IM	7



R240 SIO
Serial I/O Register
(F0H: Read/Write)

R244 TO
Counter/Timer 0 Register
(F4H: Read/Write)

D7 D6 D5 D4 D3 D2 D1 D0
T0 INITIAL VALUE (WHEN WRITTEN)
(RANGE: 1-256 DECIMAL 01-00 HEX)
T0 CURRENT VALUE (WHEN READ)

R241 TMR
Time Mode Register
(F1H; Read/Write)

R245 PRE0
Prescaler 0 Register
(F5H; Write Only)

R242 T1
Counter Timer 1 Register
(F2H; Read/Write)

D7 D6 D5 D4 D3 D2 D1 D0
T1 INITIAL VALUE (WHEN WRITTEN)
(RANGE 1 256 DECIMAL 01 00 HEX)
T1 CURRENT VALUE (WHEN READ)

R246 P2M
Port 2 Mode Register
(F6H; Write Only)

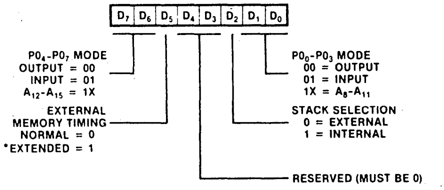
R243 PRE1
Prescaler 1 Register
(F3H; Write Only)

R247 P3M
Port 3 Mode Register
(F7H; Write Only)

D7 D6 D5 D4 D3 D2 D1 D0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
RESERVED (MUST BE 0)
0 P32 = INPUT P35 = OUTPUT
1 P32 = DAV0/RDY0 P35 = RDY0/DAV0
0 0 P33 = INPUT P34 = OUTPUT
0 1
1 0 } P33 = INPUT P34 = DM
1 1 RESERVED
0 P31 = INPUT (TIN) P36 = OUTPUT (TOUT)
1 P31 = DAV2/RDY2 P36 = RDY2/DAV2
0 P30 = INPUT P37 = OUTPUT
1 P30 = SERIAL IN P37 = SERIAL OUT
0 PARITY OFF
1 PARITY ON

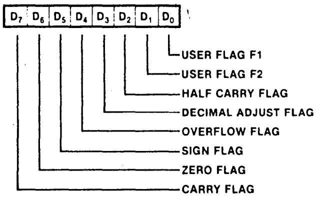
Figure 13. Control Registers

R248 P01M
Port 0 Mode Register
(F8H; Write Only)

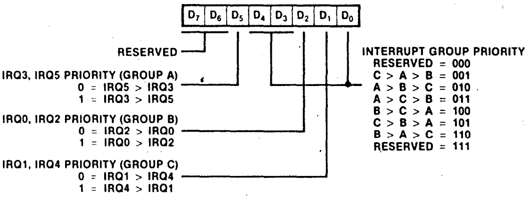


*ALWAYS EXTENDED TIMING AFTER RESET

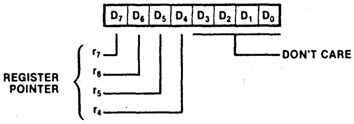
R252 FLAGS
Flag Register
(FCH; Read/Write)



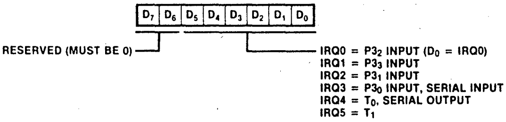
R249 IPR
Interrupt Priority Register
(F9H; Write Only)



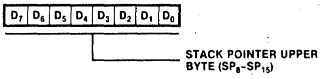
R253 RP
Register Pointer
(FDH; Read/Write)



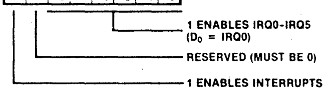
R250 IRQ
Interrupt Request Register
(FAH; Read/Write)



R254 SPH
Stack Pointer
(FEH; Read/Write)



R251 IMR
Interrupt Mask Register
(FBH; Read/Write)



R255 SPL
Stack Pointer
(FFH; Read/Write)

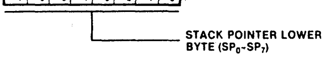
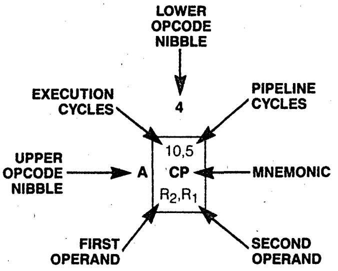


Figure 13. Control Registers (Continued)

OPCODE MAP

	Lower Nibble (Hex)															
Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6.5 **DEC** R_1	6.5 **DEC** IR_1	6.5 **ADD** r_1,r_2	6.5 **ADD** r_1,Ir_2	10.5 **ADD** R_2,R_1	10.5 **ADD** IR_2,R_1	10.5 **ADD** R_1,IM	10.5 **ADD** IR_1,IM	6.5 **LD** r_1,R_2	6.5 **LD** r_2,R_1	12/10.5 **DJNZ** r_1,RA	12/10.0 **JR** cc,RA	6.5 **LD** r_1,IM	12/10.0 **JP** cc,DA	6.5 **INC** r1	
1	6.5 **RLC** R_1	6.5 **RLC** IR_1	6.5 **ADC** r_1,r_2	6.5 **ADC** r_1,Ir_2	10.5 **ADC** R_2,R_1	10.5 **ADC** IR_2,R_1	10.5 **ADC** R_1,IM	10.5 **ADC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
2	6.5 **INC** R_1	6.5 **INC** IR_1	6.5 **SUB** r_1,r_2	6.5 **SUB** r_1,Ir_2	10.5 **SUB** R_2,R_1	10.5 **SUB** IR_2,R_1	10.5 **SUB** R_1,IM	10.5 **SUB** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
3	8,0 **JP** IRR_1	6,1 **SRP** IM	6,5 **SBC** r_1,r_2	6,5 **SBC** r_1,Ir_2	10,5 **SBC** R_2,R_1	10,5 **SBC** IR_2,R_1	10,5 **SBC** R_1,IM	10,5 **SBC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
4	8,5 **DA** R_1	8,5 **DA** IR_1	6,5 **OR** r_1,r_2	6,5 **OR** r_1,Ir_2	10,5 **OR** R_2,R_1	10,5 **OR** IR_2,R_1	10,5 **OR** R_1,IM	10,5 **OR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
5	10,5 **POP** R_1	10,5 **POP** IR_1	6,5 **AND** r_1,r_2	6,5 **AND** r_1,Ir_2	10,5 **AND** R_2,R_1	10,5 **AND** IR_2,R_1	10,5 **AND** R_1,IM	10,5 **AND** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
6	6,5 **COM** R_1	6,5 **COM** IR_1	6,5 **TCM** r_1,r_2	6,5 **TCM** r_1,Ir_2	10,5 **TCM** R_2,R_1	10,5 **TCM** IR_2,R_1	10,5 **TCM** R_1,IM	10,5 **TCM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	6,0 **STOP**
7	10/12,1 **PUSH** R_2	12/14,1 **PUSH** IR_2	6,5 **TM** r_1,r_2	6,5 **TM** r_1,Ir_2	10,5 **TM** R_2,R_1	10,5 **TM** IR_2,R_1	10,5 **TM** R_1,IM	10,5 **TM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	7,0 **HALT**
8	10,5 **DECW** RR_1	10,5 **DECW** IR_1	12,0 **LDE** r_1,Irr_2	18,0 **LDEI** Ir_1,Irr_2					↓	↓	↓	↓	↓	↓	↓	6.1 **DI**
9	6,5 **RL** R_1	6,5 **RL** IR_1	12,0 **LDE** r_2,Irr_1	18,0 **LDEI** Ir_2,Irr_1					↓	↓	↓	↓	↓	↓	↓	6.1 **EI**
A	10,5 **INCW** RR_1	10,5 **INCW** IR_1	6,5 **CP** r_1,r_2	6,5 **CP** r_1,Ir_2	10,5 **CP** R_2,R_1	10,5 **CP** IR_2,R_1	10,5 **CP** R_1,IM	10,5 **CP** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	14.0 **RET**
B	6,5 **CLR** R_1	6,5 **CLR** IR_1	6,5 **XOR** r_1,r_2	6,5 **XOR** r_1,Ir_2	10,5 **XOR** R_2,R_1	10,5 **XOR** IR_2,R_1	10,5 **XOR** R_1,IM	10,5 **XOR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	16.0 **IRET**
C	6,5 **RRC** R_1	6,5 **RRC** IR_1	12,0 **LDC** r_1,Irr_2	18,0 **LDCI** Ir_1,Irr_2				10,5 **LD** r_1,x,R_2	↓	↓	↓	↓	↓	↓	↓	6.5 **RCF**
D	6,5 **SRA** R_1	6,5 **SRA** IR_1	12,0 **LDC** r_2,Irr_1	18,0 **LDCI** Ir_2,Irr_1	20,0 **CALL*** IRR_1		20,0 **CALL** DA	10,5 **LD** r_2,x,R_1	↓	↓	↓	↓	↓	↓	↓	6.5 **SCF**
E	6,5 **RR** R_1	6,5 **RR** IR_1		6,5 **LD** r_1,IR_2	10,5 **LD** R_2,R_1	10,5 **LD** IR_2,R_1	10,5 **LD** R_1,IM	10,5 **LD** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	6.5 **CCF**
F	8,5 **SWAP** R_1	8,5 **SWAP** IR_1		6,5 **LD** Ir_1,r_2		10,5 **LD** R_2,IR_1			↓	↓	↓	↓	↓	↓	↓	6.0 **NOP**
Bytes per Instruction	2				3				2					3	1	



Legend:
R = 8-bit address
r = 4-bit address
R_1 or r_1 = Dst address
R_2 or r_2 = Src address

Sequence:
Opcode, First Operand, Second Operand

NOTE: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

ABSOLUTE MAXIMUM RATINGS

Voltages on all pins except $\overline{\text{RESET}}$
with respect to GND . . . . − 0.3V to + 7.0V
Operating Ambient
Temperature. . . . . .See Ordering Information
Storage Temperature . . . . − 65 °C to + 150 °C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the referenced pin.

Standard conditions are as follows:

- $+4.5V \leqslant V_{CC} \leqslant +5.5V$
- GND = 0V
- 0 °C $\leqslant T_A \leqslant$ + 70 °C for S (Standard temperature)
- − 40 °C $\leqslant T_A \leqslant$ + 100 °C for E (Extended temperature)

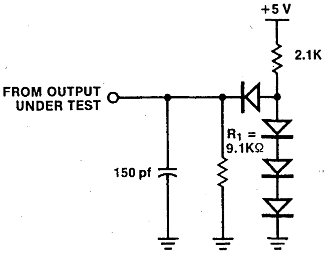


Figure 14. Test Load 1

DC CHARACTERISTICS

Symbol	Parameter	Min	Typ	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8		V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	− 0.3		0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{IL}	Input Low Voltage	− 0.3		0.8	V	
V_{RH}	Reset Input High Voltage	3.8		V_{CC}	V	
V_{RL}	Reset Input Low Voltage	− 0.3		0.8	V	
V_{OH}	Output High Voltage	2.4			V	I_{OH} = − 250 μA
V_{OH}	**Output High Voltage**	**V_{CC} -100mV**			**V**	**I_{CC} = -100μA**
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = + 2.0 mA
I_{IL}	Input Leakage	− 10		10	μA	V_{IN} = 0V, 5.25V
I_{OL}	Output Leakage	− 10		10	μA	V_{IN} = 0V, 5.25V
I_{IR}	Reset Input Current			− 50	μA	V_{CC} = + 5.25V, V_{RL} = 0V
I_{CC}	Supply Current			30	mA	All outputs and I/O pins floating
I_{CC1}	Standby Current		**5**		mA	Halt Mode
I_{CC2}	Standby Current			**10**	μA	Stop Mode

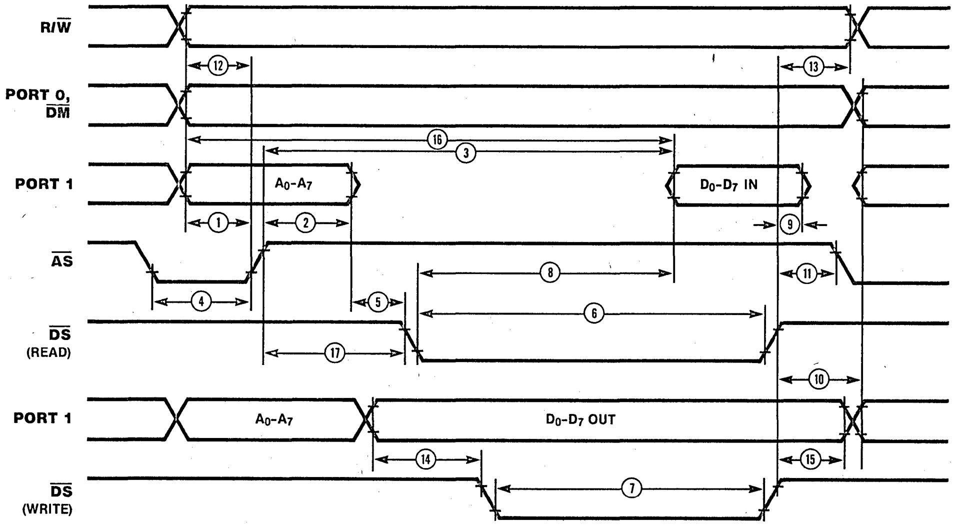


Figure 15. External I/O or Memory Read/Write Timing

AC CHARACTERISTICS

External I/O or Memory Read and Write Timing

			12 MHz		16 MHz		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes
1	TdA(AS)	Address Valid to $\overline{AS}$ ↑Delay	35		20		2,3
2	TdAS(A)	$\overline{AS}$ ↑ to Address Float Delay	45		30		2,3
3	TdAS(DR)	$\overline{AS}$ ↑ to Read Data Required Valid		220		180	1,2,3
4	TwAS	$\overline{AS}$ Low Width	55		35		2,3
5	TdAz(DS)	Address Float to $\overline{DS}$ ↓	0		0		
6	TwDSR	$\overline{DS}$ (Read) Low Width	185		135		1,2,3
7	TwDSW	$\overline{DS}$ (Write) Low Width	110		80		1,2,3
8	TdDSR(DR).	$\overline{DS}$ ↓ to Read Data Required Valid		130		75	1,2,3
9	ThDR(DS)	Read Data to $\overline{DS}$ ↑ Hold Time	0		0		2,3
10	TdDS(A)	$\overline{DS}$ ↑ to Address Active Delay	45		35		2,3
11	TdDS(AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓Delay	55		25		2,3
12	TdR/W(AS)	R/$\overline{W}$ Valid to $\overline{AS}$ ↑ Delay	30		20		2,3
13	TdDS(R/W)	$\overline{DS}$ ↑ to R/$\overline{W}$ Not Valid	35		25		2,3
14	TdDW(DSW)	Write Data Valid to $\overline{DS}$ (Write) ↓ Delay	35		25		2,3
15	TdDS(DW)	$\overline{DS}$ ↑ to Write Data Not Valid Delay	35		25		2,3
16	TdA(DR)	Address Valid to Read Data Required Valid		255		200	1,2,3
17	TdAS(DS)	$\overline{AS}$ ↑ to $\overline{DS}$ ↓ Delay	55		40		2,3

NOTES:
1. When using extended memory timing add 2 TpC.
2. Timing numbers given are for minimum TpC.
3. See clock cycle time dependent characteristics table.
4. 16 MHz timing is preliminary and subject to change.

* All units in nanoseconds (ns).
† Test Load 1
° All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".

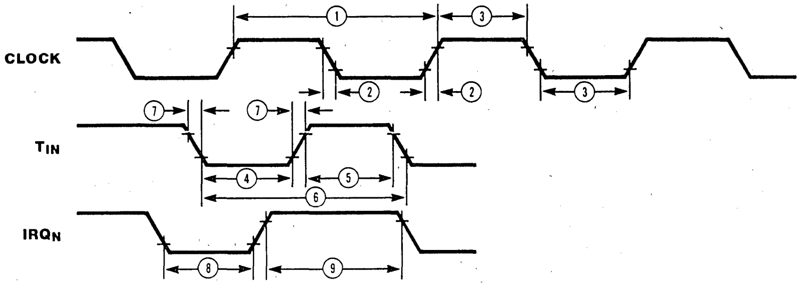


Figure 16. Additional Timing

AC CHARACTERISTICS

Additional Timing Table

Number	Symbol	Parameter	8 MHz Min	8 MHz Max	12 MHz Min	12 MHz Max	16 MHz Min	16 MHz Max	Notes
1	TpC	Input Clock Period	125	1000	83	1000	62.5	1000	1
2	TrC,TfC	Clock Input Rise and Fall Times		25		15		10	1
3	TwC	Input Clock Width	37		70		21		1
4	TwTinL	Timer Input Low Width	100		70		50		2
5	TwTinH	Timer Input High Width	3TpC		3TpC		3TpC		2
6	TpTin	Timer Input Period	8TpC		8TpC		8TpC		2
7	TrTin,TfTin	Timer Input Rise and Fall Times		100		100		100	2
8A	TwIL	Interrupt Request Input Low Time	100		70		50		2,4
8B	TwIL	Interrupt Request Input Low Time	3TpC		3TpC		3TpC		2,5
9	TwIH	Interrupt Request Input High Time	3TpC		3TpC		3TpC		2,3

NOTES:
1. Clock timing references use 3.8V for a logic "1" and 0.8V for a logic "0".
2. Timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
3. Interrupt request via Port 3.
4. Interrupt request via Port 3 ($P3_1$-$P3_3$)
5. Interrupt request via Port 3 ($P3_0$)
6. 16 MHz timing is preliminary and subject to change.
* Units in nanoseconds (ns).

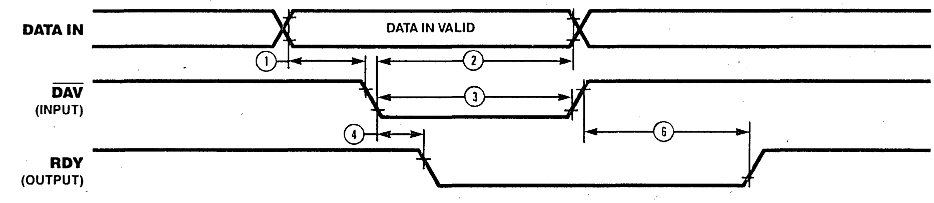


Figure 17a. Input Handshake Timing

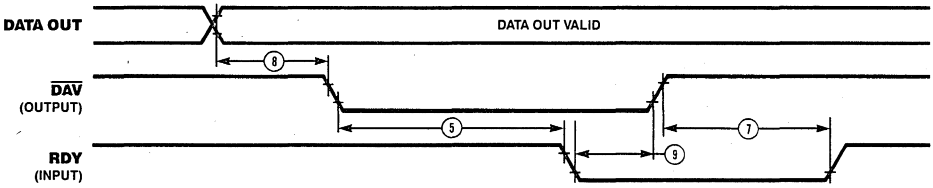


Figure 17b. Output Handshake Timing

AC CHARACTERISTICS

Handshake Timing

			8, 12, 16 MHz		
Number	Symbol	Parameter	Min	Max	Notes
1	TsDI(DAV)	Data In Setup Time	0		
2	ThDI(DAV)	Data In Hold Time	145		
3	TwDAV	Data Available Width	110		
4	TdDAVIf(RDY)	DAV ↓ Input to RDY ↓ Delay		115	1,2
5	TdDAVOf(RDY)	DAV ↓ Output to RDY ↓ Delay	0		1,3
6	TdDAVIr(RDY)	DAV ↑ Input to RDY ↑ Delay		115	1,2
7	TdDAVOr(RDY)	DAV ↑ Output to RDY ↑ Delay	0		1,3
8	TdDO(DAV)	Data Out to DAV ↓ Delay	Tpc		1
9	TdRDY(DAV)	Rdy ↓ Input to DAV ↑ Delay	0	130	1

NOTES:
1. Test load 1
2. Input handshake
3. Output handshake
4. 16 MHz timing is preliminary and subject to change.
† All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
* Units in nanoseconds (ns).

MEMORY SPACE AND REGISTER ORGANIZATION

Memory Space

The Z8 can address up to 126K bytes of program and data memory separately from the on chip registers. The 16-bit program counter provides for 64K bytes of program memory, the first 2K bytes of which are internal to the Z8. The remaining 62K bytes of program memory are located externally and can be implemented with ROM, EPROM, or RAM.

The 62K bytes of data memory are also located external to the Z8 and begin with location 2048. The two address spaces, program memory and data memory, are individually selected by the Data Memory Select output (D̅M̅) which is available from Port 3.

The Program Memory Map and the Data Memory Map are shown in Figure 2.

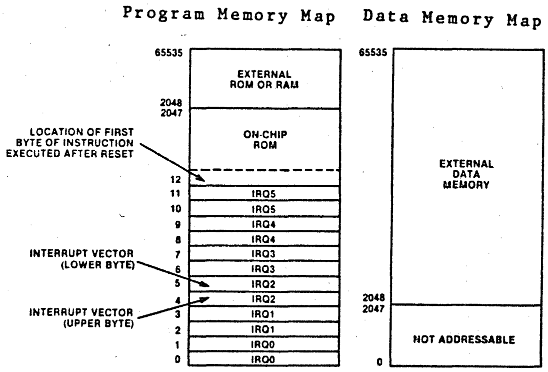

Program Memory Map Data Memory Map

Figure 2 Program Memory Map And Data Memory Map

External memory access is accomplished by the Z8 through its I/O Ports. When less than 256 bytes of external memory are required, Port 1 is programmed for the multiplexed address/data mode (ADØ-AD7). In this configuration 8-bits of address and 8-bits of data are time multiplexed on the 8 I/O lines for memory transfers. The memory "handshake" control lines are provided by the Address Strobe (A̅S̅), Data Strobe (D̅S̅), and the Read/Write (R/W̅) pins on the Z8. If program and data are included in the external memory space, the Data Memory Select (D̅M̅) function may be programmed into the Port 3 Mode register. When this is done, the D̅M̅ signal is available on line 4 of the Port 3 (P34) to select between program and data memory for external memory operations.

Port 0 is used to provide the additional address bits for external memory beyond the first 256 locations up to a full 16-bits of external memory address. It becomes immediately obvious that the first 8-bits of external memory address from Port 1 must be latched externally to the Z8 so that program or data may be transferred over the same 8 lines during the external memory transaction machine cycle. The A̅S̅, D̅S̅, and R/W̅ control lines simplify the required interface logic. The timing for external memory transactions is given in Figure 3.

Registers

The Z8 has 144 8-bit registers including four Port registers (R0-R3), 124 general purpose registers (R4-R127), and 16 control and status register (R240-R255). The 144 registers are all located in the same 8-bit address space to allow any Z8 instruction to operate on them. The 124 general purpose registers can function as accumulators, address pointers, or index registers. The registers are read when they are referenced as source registers, and written when they are referenced as destination registers. Registers may be addressed directly with an 8-bit address, or indirectly through another register with an 8-bit address, or with a 4-bit address and Register Pointer.

The entire Z8 register space may be divided into 16 contiguous Working Register Areas, each having 16 registers. A control register, called the Register Pointer, may be loaded with the most significant nibble of a Working Register Area address. The Register Pointer provides for the selection of the Working Register Area, and allows registers within that area to be selected with a 4-bit address.

The Z8 register organization is shown in Figure 4.

Stacks

The Z8 provides for stack operations through the use of a stack pointer, and the stack may be located in the internal register space or in the external data memory space. The "stack selection" bit (D2) in the Port 0-1 Mode control register selects an internal or external stack. When the stack is located internally, register 255 contains an 8-bit stack pointer and register 254 is available as a general purpose register. If an external stack is used, register 255 or registers 254 and 255 may be used as the stack pointer depending on the anticipated "depth" of the stack. When registers 254 and 255 are both used, the stack pointer is a full 16-bits wide. The CALL, IRET, RET, PUSH, and

POP instructions are Z8 instructions which include implicit stack operations.

I/O STRUCTURE

Parallel I/O

The Z8 microcomputer has 32 lines of I/O arranged as four 8-bit ports. All of the I/O ports are TTL compatible and are configurable as input, output, input/output, or address/data. The handshake control lines for Ports 0, 1, and 2 are bits from Port 3 that have been programmed through a Mode control register, except for $\overline{AS}$, $\overline{DS}$, and R/$\overline{W}$ which are available as separate Z8 pins. The I/O ports are accessed as separate internal registers by the Z8. Ports 0 and 1 share one Mode control register, and Ports 2 and 3 each have a Mode control register for configuring the port.

Port 0 can be programmed to be an I/O port or as an address output port. More specifically Port 0 can be configured to be an 8-bit I/O port, or a 4-bit address output port (A8-A11) for external memory and one 4-bit I/O port, or an 8-bit address output port (A8-A15) for external memory.

Port 1 can be programmed as an I/O port (with or without handshake), or an address/data port (ADØ-AD7) for interfacing with external memory. If Port 1 is programmed to be an address/data port, it cannot be accessed as a register.

Port 2 can be configured as individual input or output bits, and Port 3 can be programmed to be parallel I/O bits, and/or serial I/O bits, and/or handshake control lines for the other ports. Figure 5 shows the port Mode registers.

The off chip expansion capability using Ports 0 and 1 offers the added feature of being Z-Bus compatible. All Z-Bus compatible peripheral chips that are available now, and will be available in the future, will interface directly with the Z8 multiplexed address/data bus.

Serial I/O

As memtioned in the last section, Port 3 can be programmed to be a serial I/O port with bits 0 and 7, the serial input and serial output lines respectively. The serial I/O capability provides for full duplex asynchronous serial data at rates up to 62.5K bits per second. The transmitted format is one start bit, eight data bits including odd parity (if parity is enabled), and two stop bits. The received data format is one start bit, eight data bits and at least one stop bit. If parity is enabled, the eighth data bit received (bit 7) is replaced by a parity error flag which indicates a parity error if it is set to a ONE.

Timer/Counter T_0 is the baud rate generator and runs at 16 times the serial data bit rate. The receiver is double duffered and an internal interrupt (IRQ3) is generated when a character is loaded into the receive buffer register. A different internal interrupt (IRQ4) is generated when a character is transmitted.

COUNTER/TIMERS

The Z8 has two 8-bit programmable counter/timers, each of which is driven by a programmable 6-bit prescaler. The T_1 prescaler can be driven by internal or external clock sources, and the T_0 prescaler is driven by the internal clock only. The two prescalers and the two counters are loaded through four control registers (see Figure 4) and when a counter/timer reaches the "end of count" a timer interrupt is generated (IRQ4 for T_0, and IRQ5 for T_1). The counter/timers can be programmed to stop upon reaching the end of count, or to reload and continue counting. Since either counter (one at a time) can have its output available external to the Z8, and Counter/Timer T_1 can have an external input, the two counters can be cascaded.

Port 3 can be programmed to provide timer outputs for external time base generation or trigger pulses.

INTERRUPT STRUCTURE

The Z8 provides for six interrupts from eight different sources including four Port 3 lines (P30-P33), serial in, serial out, and two counter/timers. These interrupts can be masked and prioritized using the Interrupt Mask Register (register 251) and the Interrupt Priority Register (register 249). All interrupts can be disabled with the master interrupt enable bit in the Interrupt Mask Register.

Each of the six interrupts has a 16-bit interrupt vector that points to its interrupt service routine. These six 2-byte vectors are placed in the first twelve locations in the program memory space (see Figure 2).

When simultaneous interrupts occur for enabled interrupt sources, the Interrupt Priority Register determines which interrupt is serviced first. The priority is programmable in a way that is described by Figure 6.

When an interrupt is recognized by the Z8, all other interrupts are disabled, the program counter and program control flags are saved, and the program counter is loaded with the corresponding interrupt vector. Interrupts must be re-enabled by the user upon entering the service

A Programmer's Guide to the Z8™ Microcomputer

Zilog

Application Note

Doll Freund

October 1980

SECTION 1 Introduction

The Z8 is the first microcomputer to offer both a highly integrated microcomputer on a single chip and a fully expandable microprocessor for I/O-and memory-intensive applications. The Z8 has two timer/counters, a UART, 2K bytes internal ROM, and a 144-byte internal register file including 124 bytes of RAM, 32 bits of I/O, and 16 control and status registers. In addition, the Z8 can address up to 124K bytes of external program and data memory, which can provide full, memory-mapped I/O capability.

This application note describes the important features of the Z8, with software examples that illustrate its power and ease of use. It is divided into sections by topic; the reader need not read each section sequentially, but may skip around to the sections of current interest.

It is assumed that the reader is familiar with the Z8 and its assembly language, as described in the following documents:

- *Z8 Technical Manual* (03-3047-02)
- *Z8 PLZ/ASM Assembly Language Programming Manual* (03-3023-02)

SECTION 2 Accessing Register Memory

The Z8 register space consists of four I/O ports, 16 control and status registers, and 124 general-purpose registers. The general-purpose registers are RAM areas typically used for accumulators, pointers, and stack area. This section describes these registers and how they are used. Bit manipulation and stack operations affecting the register space are discussed in Sections 4 and 5, respectively.

2.1 Registers and Register Pairs. The Z8 supports 8-bit registers and 16-bit register pairs. A register pair consists of an even-numbered register concatenated with the next higher numbered register (%00 and %01, %02 and %03, ... %7E and %7F, %F0 and %F1, ... %FE and %FF). A register pair must be addressed by reference to the even-numbered register. For example,

%F1 and %F2 is not a valid register pair;
%F0 and %F1 is a valid register pair, addressed by reference to %F0.

Register pairs may be incremented (INCW) and decremented (DECW) and are useful as pointers for accessing program and external data memory. Section 3 discusses the use of register pairs for this purpose.

Any instruction which can reference or modify an 8-bit register can do so to any of the 144 registers in the Z8, regardless of the inherent nature of that register. Thus, I/O ports, control, status, and general-purpose registers may all be accessed and manipulated without the need for special-purpose instructions. Similarly, instructions which reference or modify a 16-bit register pair can do so to any of the valid 72 register pairs. The only exceptions to this rule are:

- The DJNZ (decrement and jump if non-zero) instruction may successfully operate on the general-purpose RAM registers (%04–%7F) only.
- Six control registers are write-only registers and therefore, may be modified only by such instructions as LOAD, POP, and CLEAR. Instructions such as OR and AND require that the current contents of the operand be readable and therefore will not function properly on the write-only registers. These registers are the following: *the timer/counter prescaler registers PRE0 and PRE1, the port mode registers P01M, P2M, and P3M, the interrupt priority register IPR.*

2. Accessing Register Memory (Continued)

2.2 Register Pointer. Within the register addressing modes provided by the Z8, a register may be specified by its full 8-bit address (0-%7F, %F0-%FF) or by a short 4-bit address. In the latter case, the register is viewed as one of 16 working registers within a working register group. Such a group must be aligned on a 16-byte boundary and is addressed by Register Pointer RP (%FD). As an example, assume the Register Pointer contains %70, thus pointing to the working register group from %70 to %7F. The LD instruction may be used to initialize register %76 to an immediate value in one of two ways:

```
LD  %76,#1  !8-bit register address is given
             by instruction (3 byte instruc-
             tion)!
    or
LD  R6,#1   !4-bit working register address
             is given by instruction; 4-bit
             working register group
             address is given by Register
             Pointer (2 byte instruction)!
```

The address calculation for the latter case is illustrated in Figure 1. Notice that 4-bit working-register addressing offers code compactness and fast execution compared to its 8-bit counterpart.

To modify the contents of the Register Pointer, the Z8 provides the instruction

```
SRP  #value
```

Execution of this instruction will load the upper four bits of the Register Pointer; the lower four bits are always set to zero. Although a load instruction such as

```
LD  RP,#value
```

could be used to perform the same function, SRP provides execution speed (six vs. ten cycles) and code space (two vs. three bytes) advantages over the LD instruction. The instruction

```
SRP  #%70
```

is used to set the Register Pointer for the above example.

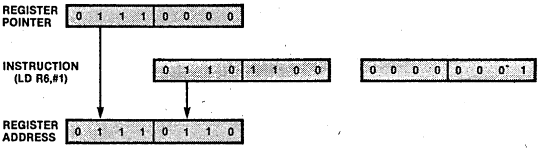


Figure 1. Address Calculation Using the Register Pointer

2.3 Context Switching. A typical function performed during an interrupt service routine is context switching. Context switching refers to the saving and subsequent restoring of the program counter, status, and registers of the interrupted task. During an interrupt machine cycle, the Z8 automatically saves the Program Counter and status flags on the stack. It is the responsibility of the interrupt service routine to preserve the register space. The recommended means to this end is to allocate a specific portion of the register file for use by the service routine. The service routine thus preserves the register space of the interrupted task by avoiding modification of registers not allocated as its own. The most efficient scheme with which to implement this function in the Z8 is to allocate a working register group (or portion thereof) to the interrupt service routine. In this way, the preservation of the interrupted task's registers is solely a matter of saving the Register Pointer on entry to the service routine, setting the Register Pointer to its own working register group, and restoring the Register Pointer prior to exiting the service routine. For example, assume such a register allocation scheme has been implemented in which the interrupt service routine for IRQ0 may access only working register Group 4 (registers %40-%4F). The service routine for IRQ0 should be headed by the code sequence:

```
PUSH RP    !preserve Register Pointer of
            interrupted task!
SRP  #%40  !address working register
            group 4!
```

Before exiting, the service routine should execute the instruction

```
POP  RP
```

to restore the Register Pointer to its entry value.

It should be noted that the technique described above need not be restricted to interrupt service routines. Such a technique might prove efficient for use by a subroutine requiring intermediate registers to produce its outputs. In this way, the calling task can assume that its environment is intact upon return from the subroutine.

2. Accessing Register Memory (Continued)

2.4 Addressing Mode. The Z8 provides three addressing modes for accessing the register space: Direct Register, Indirect Register, and Indexed.

2.4.1 Direct Register Addressing. This addressing mode is used when the target register address is known at assembly time. Both long (8-bit) register addressing and short (4-bit) working register addressing are supported in this mode. Most instructions supporting this mode provide access to single 8-bit registers. For example:

```
LD   %FE,#HI STACK
              !load register %FE (SPH) with
               the upper 8-bits of the label
               STACK!
AND 0,MASK__REG
              !AND register 0 with register
               named MASK__REG!
OR   1,R5   !OR register 1 with working
               register 5!
```

Increment word (INCW) and decrement word (DECW) are the only two Z8 instructions which access 16-bit operands. These instructions are illustrated below for the direct register addressing mode.

```
INCW  RR0  !increment working register
            pair R0, R1:
            R1 ◄- R1 + 1
            R0 ◄- R0 + carry!
DECW  %7E
           !decrement working register
            pair %7E, %7F:
            %7F ◄- %7F - 1
            %7E ◄- %7E - carry!
```

Note that the instruction

```
INCW  RR5
```

will be flagged as an error by the assembler (RR5 not even-numbered).

2.4.2 Indirect Register Addressing. In this addressing mode, the operand is pointed to by the register whose 8-bit register address or 4-bit working register address is given by the instruction. This mode is used when the target register address is not known at assembly time and must be calculated during program execution. For example, assume registers %60-%7F contain a buffer for output to the serial line via repetitive calls to procedure SERIAL__OUT. SERIAL__OUT expects working register 0 to hold the output character. The following instructions illustrate the use of the indirect addressing mode to accomplish this task:

```
  LD     R1,#%20
              !working register 1 is the byte
               counter: output %20 bytes!
  LD     R2,#%60
              !working register 2 is the buf-
               fer pointer register!
out__again:
  LD     R0,@R2
              !load into working register 0
               the byte pointed to by working
               register 2!
  INC    R2   !increment pointer!
  CALL   SERIAL__OUT
              !output the byte!
  DJNZ   R1,out __again
              !loop till done!
```

Indirect addressing may also be used for accessing a 16-bit register pair via the INCW and DECW instructions. For example,

```
INCW  @R0  !increment the register pair
            whose address is contained in
            working register 0!
DECW  @%7F
           !decrement the register pair
            whose address is contained in
            register %7F!
```

The contents of registers R0 and %7F should be even numbers for proper access; when referencing a register pair, the least significant address bit is forced to the appropriate value by the Z8. However, the register used to point to the register pair need not be an even-numbered register.

Since the indirect addressing mode permits calculation of a target address prior to the desired register access, this mode may be used to simulate other, more complex addressing modes. For example, the instruction

```
SUB  4,BASE(R5)
```

requires the indexed addressing mode which is not directly supported by the Z8 SUBtract instruction. This instruction can be simulated as follows:

```
LD   R6,#BASE
            !working register 6 has the
             base address!
ADD  R6,R5  !calculate the target address!
SUB  4,@R6  !now use indirect addressing to
             perform the actual subtract!
```

Any available register or working register may be used in place of R6 in the above example.

2.4.3 Indexed Addressing. The indexed addressing mode is supported by the load instruction (LD) for the transference of bytes between a working register and another register. The effective address of the latter register is given by the instruction which is offset by the contents of a designated working (index)

2. Accessing Register Memory (Continued)

register. This addressing mode provides efficient memory usage when addressing consecutive bytes in a block of register memory, such as a table or a buffer. The working register used as the index in the effective address calculation can serve the additional role of counter for control of a loop's duration.

For example, assume an ASCII character buffer exists in register memory starting at address BUF for LENGTH bytes. In order to determine the logical length of the character string, the buffer should be scanned backward until the first nonoccurrence of a blank character. The following code sequence may be used to accomplish this task:

```
    LD    R0,#LENGTH
                  !length of buffer!
                  !starting at buffer end, look for
                   1st non-blank!
loop:
    LD    R1,BUF-1(R0)
    CP    R1,#' '
    JR    ne,found
                  !found non-blank!
    DJNZ  R0,loop
                  !look at next!
all__blanks:      !length = 0!
found:
```

5 instructions
12 bytes
1.5 μs overhead
10.5 μs (average) per character tested

At labels "all__blanks" and "found," R0 contains the length of the character string. These labels may refer to the same location, but they are shown separately for an application where special processing is required for a string of zero length. To perform this task without indexed addressing would require a code sequence such as:

```
    LD    R1,#BUF+LENGTH-1
    LD    R0,#LENGTH
                  !starting at buffer end, look for
                   1st non-blank!
loop1:
    CP    @R1,#' '
    JR    ne,found1
                  !found non-blank!
    DEC   R1      !dec pointer!
    DJNZ  R0,loop1
                  !are we done?!
all__blanks1:     !length = 0!
found1:
```

6 instructions
13 bytes
3 μs overhead
9.5 μs (average) per character tested

The latter method requires one more byte of program memory than the former, but is faster by four execution cycles (1 μs) per character tested.

As an alternate example, assume a buffer exists as described above, but it is desired to scan this buffer forward for the first occurrence of an ASCII carriage return. The following illustrates the code to do this:

```
    LD    R0,#-LENGTH
                  !starting at buffer start, look for
                   1st carriage return (=%0D)!
next:
    LD    r1,BUF+LENGTH(R0)
    CP    R1,#%0D
    JR    eq,cr   !found it!
    INC   R0      !update counter/index!
    JR    nz,next
                  !try again!
cr:
    ADD   R0,#LENGTH
                  !R0 has length to CR!
```

7 instructions
16 bytes
1.5 μs overhead
12 μs (average) per character tested

SECTION 3

Accessing Program and External Data Memory

In a single instruction, the Z8 can transfer a byte between register memory and either program or external data memory. Load Constant (LDC) and Load Constant and Increment (LDCI) reference program memory; Load External (LDE) and Load External and Increment (LDEI) reference external data memory. These instructions require that a working register pair contain the address of the byte in either program or external data memory to be accessed by the instruction (indirect working register pair addressing mode). The register byte operand is specified by using the direct working register addressing mode in LDC and LDE or the indirect working register addressing mode in LDCI and LDEI. In addition to performing the designated byte transfer, LDCI and LDEI automatically increment both the indirect registers specified by the instruction. These instructions are therefore efficient for performing block moves between register and either program or external data memory. Since the indirect addressing mode is used to specify the operand address within program or external data memory, more complex addressing modes may be simulated as discussed earlier in Section 2.4.2. For example, the instruction

```
    LDC   R3,BASE(R2)
```

requires the indexed addressing mode, where

3. Accessing Program and External Data Memory (Continued)

BASE is the base address of a table in program memory and R2 contains the offset from table start to the desired table entry. The following code sequence simulates this instruction with the use of two additional registers (R0 and R1 in this example).

```
LD     R0,#HI BASE
LD     R1,#LO BASE
                 !RR0 has table start address!
ADD    R1,R2
ADC    R0,#0
                 !RR0 has table entry address!
LDC    R3,@RR0
                 !R3 has the table entry!
```

3.1 Configuring the Z8 for I/O Applications vs. Memory Intensive Applications. The Z8 offers a high degree of flexibility in memory and I/O intensive applications. Thirty-two port bits are provided of which 16, 12, eight, or zero may be configured as address bits to external memory. This allows for addressing of 62K, 4K or 256 bytes of external memory, which can be expanded to 124K, 8K, or 512 bytes if the Data Memory Select output ($\overline{DM}$) is used to distinguish between program and data memory accesses. The following instructions illustrate the code sequence required to configure the Z8 with 12 external addressing lines and to enable the Data Memory Select output.

```
LD      P01M,#%(2)00010010
                !bit 3-4: enable AD0-AD7;
                 bit 0-1: enable A8-A11!
LD      P3M,#%(2)00001000
                !bit 3-4: enable DM!
```

The two bytes following the mode selection of ports 0 and 1 should not reference external memory due to pipelining of instructions within the Z8. Note that the load instruction to P3M satisfies this requirement (providing that it resides within the internal 2K bytes of memory).

3.2 LDC and LDE. To illustrate the use of the Load Constant (LDC) and Load External (LDE) instructions, assume there exists a hardware configuration with external memory and Data Memory Select enabled. The following module illustrates a program for tokenizing an ASCII input buffer. The program assumes there is a list of delimiters (space, comma, tab, etc.) in program memory at address DELIM for COUNT bytes (accessed via LDC) and that an ASCII input buffer exists in external data memory (accessed via LDE). The program scans the input buffer from the current location and returns the start address of the next token (i.e. the address of the first nondelimiter found) and the length of that token (number of characters from token start to next delimiter).

```
Z8ASM      2.0
LOC      OBJ CODE    STMT SOURCE STATEMENT

                        1 SCAN     MODULE
                        2 CONSTANT
                        3  COUNT   :=       6
                        4 GLOBAL
                        5          $SECTION PROGRAM
P 0000 20  3B  2C       6 DELIM    ARRAY    [COUNT BYTE]    :=
P 0003 2E  0A  0D
                        7                    [' ' , ';' , ',' , '.' , %0A , %0D]
                        8
P 0006                  9 scan     PROCEDURE
                       10 !#####################################################
                       11  Purpose =       To find the next token within an
                       12                  ASCII buffer.
                       13
                       14  Input =         RR0 = address of current location
                       15                        within input buffer in external
                       16                        memory.
                       17
                       18  Output =        RR4 = address of start of next token
                       19                  RR0 = address of new token's ending
                       20                        delimiter
                       21                  R2  = length of token
                       22                  R3  = ending delimiter
                       23                  R6,R7,R8,R9 destroyed
                       24
                       25 #####################################################!
                       26 ENTRY
P 0006 B0  E2          27         clr      R2       !init. length counter!
                       28        DO
P 0008 82  30          29         LDE      R3,@RR0  !get byte from input buffer!
P 000A A0  E0          30         incw     RR0      !increment pointer!
P 000C D6  002E'       31         call     check    !look for non-delimiter!
P 000F FD  0015'       32         IF  C THEN
P 0012 8D  0018'       33           EXIT            !found token start!
                       34         FI
P 0015 8D  0008'       35        OD
```

3. Accessing Program and External Data Memory (Continued)

```
P 0018 48   E0
P 001A 58   E1             ld     R4,R0
                           ld     R5,R1    !RR4 = token starting addr!
                          DO
P 001C 2E                  inc    R2       !inc. length counter!
P 001D 82   30             LDE    R3,@RR0  !get next input byte!
P 001F D6   002E'          call   check    !look for delimiter!
P 0022 7D   0028'          IF  NC THEN
P 0025 8D   002D'            EXIT          !found token end!
                           FI
P 0028 A0   E0             incw   RR0      !point to next byte!
P 002A 8D   001C'         OD

P 002D AF                  ret
P 002E               END   scan

P 002E               check   PROCEDURE
                     !******************************************************
                      Purpose =       compare current character with
                                      delimiter table until table
                                      end or match found

                      input =         DELIM = start address of table
                                      COUNT = length of that table
                                      R3 = byte to be scrutinized

                      output =        Carry flag = 1 => input byte
                                      is not a delimiter (no match found)

                                      Carry flag = 0 => input byte
                                      is a delimiter (match found)
                                      R6,R7,R8,R9 destroyed

                     ******************************************************!
                     ENTRY
P 002E 6C   00*                ld     R6,#HI DELIM
P 0030 7C   00*                ld     R7,#LO DELIM    !RR6 points to
                                                       delimiter list!
P 0032 8C   06                 ld     R8,#COUNT       !R8 = length of list!
                     here:
P 0034 C2   96                 LDC    R9,@RR6         !get table entry!
P 0036 A0   E6                 incw   RR6             !point to next entry!
P 0038 A2   93                 cp     R9,R3           !R3 = delimiter?!
P 003A 6B   03                 jr     eq,bye          !yes. carry = 0!
P 003C 8A   F6                 djnz   R8,here         !next entry!
P 003E DF                      scf                    !table done. R3
                                                       not a delimiter!
                     bye:
P 003F AF                      ret
P 0040               END       check
                     END       SCAN

    0 ERRORS
ASSEMBLY COMPLETE
```

27 instructions
58 bytes
Execution time is a function of the number of leading delimiters before token start (x) and the number of characters in the token (y): 123 μs overhead + 59x μs + 102y μs (average) per token

3.3 LDCI. A common function performed in Z8 applications is the initialization of the register space. The most obvious approach to this function is the coding of a sequence of "load register with immediate value" instructions (each occupying three program bytes for a register or two program bytes for a working register). This approach is also the most efficient technique for initializing less than eight consecutive registers or 14 consecutive working registers. For a larger register block, the

3. Accessing Program and External Data Memory (Continued)

LDCI instruction provides an economical means of initializing consecutive registers from an initialization table in program memory. The following code excerpt illustrates this technique of initializing control registers %F2 through %FF from a 14-byte array (INIT__tab) in program memory:

```
  SRP   #%00
              !RP not %F0!
  LD    R6,#HI INIT__tab
  LD    R7,#LO INIT__tab
  LD    R8,#%F2
              !1st reg to be initialized!
  LD    R9,#14
              !length of register block!
loop:
  LDCI  @R8,@RR6
              !load a register from the
              init table!
  DJNZ  R9,loop
              !continue till done!
```

7 instructions
14 bytes
7.5 μs overhead
7.5 μs per register initialized

3.4 LDEI. The LDEI instruction is useful for moving blocks of data between external and register memory since auto-increment is performed on both indirect registers designated by the instruction. The following code excerpt illustrates a register buffer being saved at address %40 through %60 into external memory at address SAVE:

```
  LD    R10,#HI SAVE
              !external memory!
  LD    R11,#LO SAVE
              !address!
  LD    R8,#%40
              !starting register!
  LD    R9,#%21
              !number of registers to save in
              external data memory!
loop:
  LDEI  @RR10,@R8
              !init a register!
  DJNZ  R9,loop
              !until done!
```

6 instructions
12 bytes
6 μs overhead
7.5 μs per register saved

SECTION 4

Bit Manipulations

Support of the test and modification of an individual bit or group of bits is required by most software applications suited to the Z8 microcomputer. Initializing and modifying the Z8 control registers, polling interrupt requests, manipulating port bits for control of or communication with attached devices, and manipulation of software flags for internal control purposes are all examples of the heavy use of bit manipulation functions. These examples illustrate the need for such functions in all areas of the Z8 register space. These functions are supported in the Z8 primarily by six instructions:

- Test under Mask (TM)
- Test Complement under Mask (TCM)
- AND
- OR
- XOR
- Complement (COM)

These instructions may access any Z8 register, regardless of its inherent type (control, I/O, or general purpose), with the exception of the six write-only control registers (PRE0, PRE1, P01M, P2M, P3M, IPR) mentioned earlier in Section 2.1. Table 1 summarizes the function performed on the destination byte by each of the above instructions. All of these instructions, with the exception of COM, require a mask operand. The "selected" bits referenced in Table 1 are those bits in the destination operand for which the corresponding mask bit is a logic 1.

Opcode	Use
TM	To test selected bits for logic 0
TCM	To test selected bits for logic 1
AND	To reset all but selected bits to logic 0
OR	To set selected bits to logic 1
XOR	To complement selected bits
COM	To complement all bits

Table 1. Bit Manipulation Instruction Usage

The instructions AND, OR, XOR, and COM have functions common to today's microprocessors and therefore are not described in depth here. However, examples of the use of these instructions are laced throughout the remainder of this document, thus giving an integrated view of their uses in common functions. Since they are unique to the Z8, the functions of Test under Mask and Test Complement under Mask, are discussed in more detail next.

4.1 Test under Mask (TM). The Test under Mask instruction is used to test selected bits for logic 0. The logical operation performed is

destination AND source

Neither source nor destination operand is modified; the FLAGS control register is the only register affected by this instruction. The zero flag (Z) is set if all selected bits are logic 0; it is reset otherwise. Thus, if the selected destination bits are either all logic 1 or a combination of 1s and 0s, the zero flag would be cleared by this instruction. The sign flag (S) is either set or reset to reflect the result of the

4. Bit Manipulations (Continued)

AND operation; the overflow flag (V) is always reset. All other flags are unaffected. Table 2 illustrates the flag settings which result from the TM instruction on a variety of source and destination operand combinations. Note that a given TM instruction will never result in both the Z and S flags being set.

4.2 Test Complement under Mask. The Test Complement under Mask instruction is used to test selected bits for logic 1. The logical operation performed is

(NOT destination) AND source.

As in Test under Mask, the FLAGS control register is the only register affected by this operation. The zero flag (Z) is set if all selected destination bits are 1; it is reset otherwise. The sign flag (S) is set or reset to reflect the result of the AND operation; the overflow flag (V) is always reset. Table 3 illustrates the flag settings which result from the TCM instruction on a variety of source and destination operand combinations. As with the TM instruction, a given TCM instruction will never result in both the Z and S flags being set.

Destination	Source	Flags		
(binary)	(binary)	Z	S	V
10001100	01110000	1	0	0
01111100	01110000	0	0	0
10001100	11110000	0	1	0
11111100	11110000	0	1	0
00011000	10100001	1	0	0
01000000	10100001	1	0	0

Table 2. Effects of the TM Instruction

Destination	Source	Flags		
(binary)	(binary)	Z	S	V
10001100	01110000	0	0	0
01111100	01110000	1	0	0
10001100	11110000	0	0	0
11111100	11110000	1	0	0
00011000	10100001	0	1	0
01000000	10100001	0	1	0

Table 3. Effects of the TCM Instruction

SECTION 5

Stack Operations

The Z8 stack resides within an area of data memory (internal or external). The current address in the stack is contained in the stack pointer, which decrements as bytes are pushed onto the stack, and increments as bytes are popped from it. The stack pointer occupies two control register bytes (%FE and %FF) in the Z8 register space and may be manipulated like any other register. The stack is useful for subroutine calls, interrupt service routines, and parameter passing and saving. Figure 2 illustrates the downward growth of a stack as bytes are pushed onto it.

5.1 Internal vs. External Stack. The location of the stack in data memory may be selected to be either internal register memory or external data memory. Bit 2 of control register P01M (%F8) controls this selection. Register pair SPH (%FE), SPL (%FF) serves as the stack pointer for an external stack. Register SPL is the stack pointer for an internal stack. In the latter configuration, SPH is available for use as a data register. The following illustrates a code sequence that initializes external stack operations:

```
LD P01M,#%(2)00000000
                !bit 2: select external stack!
LD SPH,#HI STACK
LD SPL,#LO STACK
```

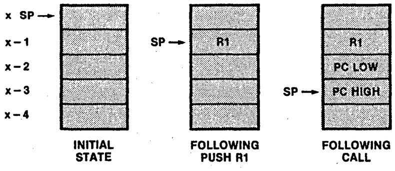


Figure 2. Growth of a Stack

5.2 CALL. A subroutine call causes the current Program Counter (the address of the byte following the CALL instruction) to be pushed onto the stack. The Program Counter is loaded with the address specified by the CALL instruction. This address may be a direct address or an indirect register pair reference. For example,

```
LABEL 1: CALL   %4F98
                !direct addressing: PC is
                 loaded with the hex value
                 4F98;
                 address LABEL 1+3 is pushed
                 onto the stack!
LABEL 2: CALL   @RR4
                !indirect addressing: PC is
                 loaded with the contents of
                 working register pair R4, R5;
                 address LABEL 2+2 is pushed
                 onto the stack!
```

5. Stack Operations (Continued)

```
LABEL 3: CALL   @%7E
                 !indirect addressing: PC is
                  loaded with the contents of
                  register pair %7E, %7F;
                  address LABEL 3+2 is pushed
                  onto the stack!
```

5.3 RET. The return (RET) instruction causes the top two bytes to be popped from the stack and loaded into the Program Counter. Typically, this is the last instruction of a subroutine and thus restores the PC to the address following the CALL to that subroutine.

5.4 Interrupt Machine Cycle. During an interrupt machine cycle, the PC followed by the status flags is pushed onto the stack. (A more detailed discussion of interrupt processing is provided in Section 6.)

5.5 IRET. The interrupt return (IRET) instruction causes the top byte to be popped from the stack and loaded into the status flag register, FLAGS (%FC); the next two bytes are then popped and loaded into the Program Counter. In this way, status is restored and program execution continues where it had left off when the interrupt was recognized.

5.6 PUSH and POP. The PUSH and POP instructions allow the transfer of bytes between the stack and register memory, thus providing program access to the stack for saving and restoring needed values and passing parameters to subroutines.

Execution of a PUSH instruction causes the stack pointer to be decremented by 1; the operand byte is then loaded into the location pointed to by the decremented stack pointer. Execution of a POP instruction causes the byte addressed by the stack pointer to be loaded into the operand byte; the stack pointer is then incremented by 1. In both cases, the operand byte is designated by either a direct register address or an indirect register reference. For example:

```
PUSH  R1    !direct address: push working
             register 1 onto the stack!
POP   5     !direct address: pop the top
             stack byte into register 5!
PUSH  @R4   !indirect address: pop the top
             stack byte into the byte
             pointed to by working reg-
             ister 4!
PUSH  @17   !indirect address: push onto
             the stack the byte pointed to
             by register 17!
```

SECTION 6

Interrupts

The Z8 recognizes six different interrupts from four internal and four external sources, including internal timer/counters, serial I/O, and four Port 3 lines. Interrupts may be individually or globally enabled/disabled via Interrupt Mask Register IMR (%FB) and may be prioritized for simultaneous interrupt resolution via Interrupt Priority Register IPR (%F9). When enabled, interrupt request processing automatically vectors to the designated service routine. When disabled, an interrupt request may be polled to determine when processing is needed.

6.1 Interrupt Initialization. Before the Z8 can recognize interrupts following RESET, some initialization tasks must be performed. The initialization routine should configure the Z8 interrupt requests to be enabled/disabled, as required by the target application and assigned a priority (via IPR) for simultaneous enabled-interrupt resolution. An interrupt request is enabled if the corresponding bit in the IMR is set (= 1) and interrupts are globally enabled (bit 7 of IMR = 1). An interrupt request is disabled if the corresponding bit in the IMR is reset (= 0) or interrupts are globally disabled (bit 7 of IMR = 0).

A RESET of the Z8 causes the contents of the Interrupt Request Register IRQ (%FA) to be held to zero until the execution of an EI instruction. Interrupts that occur while the Z8 is in this initial state will not be recognized, since the corresponding IRQ bit cannot be set. The EI instruction is specially decoded by the Z8 to enable the IRQ; simply setting bit 7 of IMR is therefore *not* sufficient to enable interrupt processing following RESET. However, subsequent to this initial EI instruction, interrupts may be globally enabled either by the instruction

```
EI          !enable interrupts!
```

or by a register manipulation instruction such as

```
OR     IMR,#%80
```

To globally disable interrupts, execute the instruction

```
DI          !disable interrupts!
```

This will cause bit 7 of IMR to be reset.

Interrupts *must* be globally disabled prior to any modification of the IMR, IPR or enabled bits of the IRQ (those corresponding to enabled interrupt requests), unless it can be *guaranteed* that an enabled interrupt will not occur during the processing of such instructions. Since interrupts represent the occurrence of events asynchronous to program execution, it is highly unlikely that such a guarantee can be made reliably.

6. Interrupts (Continued)

6.2 Vectored Interrupt Processing. Enabled interrupt requests are processed in an automatic vectored mode in which the interrupt service routine address is retrieved from within the first 12 bytes of program memory. When an enabled interrupt request is recognized by the Z8, the Program Counter is pushed onto the stack (low order 8 bits first, then high-order 8 bits) followed by the FLAGS register (#%FC). The corresponding interrupt request bit is reset in IRQ, interrupts are globally disabled (bit 7 of IMR is reset), and an indirect jump is taken on the word in location 2x, 2x + 1 (x = interrupt request number, 0≤x≤5). For example, if the bytes at addresses %0004 and %0005 contain %05 and %78 respectively, the interrupt machine cycle for IRQ2 will cause program execution to continue at address %0578.

When interrupts are sampled, more than one interrupt may be pending. The Interrupt Priority Register (IPR) controls the selection of the pending interrupt with highest priority. While this interrupt is being serviced, a higher-priority interrupt may occur. Such interrupts may be allowed service within the current interrupt service routine (nested) or may be held until the current service routine is complete (non-nested).

To allow nested interrupt processing, interrupts must be selectively enabled upon entry to an interrupt service routine. Typically, only higher-priority interrupts would be allowed to nest within the current interrupt service. To do this, an interrupt routine must "know" which interrupts have a higher priority than the current interrupt request. Selection of such nesting priorities is usually a reflection of the priorities established in the Interrupt Priority Register (IPR). Given this data, the first instructions executed in the service routine should be to save the current Interrupt Mask Register, mask off all interrupts of lower and equal priority, and globally enable interrupts (EI). For example, assume that service of interrupt requests 4 and 5 are nested within the service of interrupt request 3. The following illustrates the code required to enable IRQ4 and IRQ5:

```
CONSTANT
          INT__MASK__3          :=          %(2) 00110000
GLOBAL
IRQ3__service        PROCEDURE              ENTRY
!service routine for IRQ3!
          PUSH IMR                          !save Interrupt Mask Register!
                    !interrupts were globally disabled during the interrupt
                     machine cycle - no DI is needed prior to modification of IMR!
          AND  IMR,#INT__MASK__3            !disable all but IRQ4 & 5!
          EI
          !...!     !service interrupt!
                    !interrupts are globally enabled now — must disable them prior to
                     modification of IMR!
          DI
          POP  IMR                          !restore entry IMR!
          IRET
END IRQ3__service
```

Note that IRQ4 and IRQ5 are enabled by the above sequence only if their respective IMR bits = 1 on entry to IRQ3__service.

The service routine for an interrupt whose processing is to be completed without interruption should not allow interrupts to be nested within it. Therefore, it need not modify the IMR, since interrupts are disabled automatically during the interrupt machine cycle.

The service routine for an enabled interrupt is typically concluded with an IRET instruction, which restores the FLAGS register and Program Counter from the top of the stack and globally enables interrupts. To return from an interrupt service routine without re-enabling interrupts, the following code sequence could be used:

```
POP   FLAGS
            !FLAGS ◄─ @SP!
RET         !PC ◄─ @SP!
```

This accomplishes all the functions of IRET, except that IMR is not affected.

6.3 Polled Interrupt Processing Disabled interrupt requests may be processed in a polled mode, in which the corresponding bits of the Interrupt Request Register (IRQ) are examined by the software. When an interrupt request bit is found to be a logic 1, the interrupt should be processed by the appropriate

6. Interrupts
(Continued)

service routine. During such processing, the interrupt request bit in the IRQ must be cleared by the software in order for subsequent interrupts on that line to be distinguished from the current one. If more than one interrupt request is to be processed in a polled mode, polling should occur in the order of established priorities. For example, assume that IRQ0, IRQ1, and IRQ4 are to be polled and that established priorities are, from high to low, IRQ4, IRQ0, IRQ1. An instruction sequence like the following should be used to poll and service the interrupts:

```
!...!
 !poll interrupt inputs here!
            TCM     IRQ, #%(2)00010000              !IRQ4 need service?!
            JR      NZ, TEST0                       !no!
            CALL    IRQ4__service                   !yes!
 TEST0:     TCM     IRQ, #%(2)00000001              !IRQ0 need service?!
            JR      NZ, TEST1                       !no!
            CALL    IRQ0__service                   !yes!
 TEST1:     TCM     IRQ, #%(2)00000010              !IRQ1 need service?!
            JR      NZ, DONE                        !no!
            CALL    IRQ1__service                   !yes!
 DONE:      !...!

 IRQ4__service      PROCEDURE            ENTRY
            !...!
            AND     IRQ, #%(2)11101111              !clear IRQ4!
            !...!
            RET
 END IRQ4__service

 IRQ0__service      PROCEDURE            ENTRY
            !...!
            AND     IRQ, #%(2)11111110              !clear IRQ0!
            !...!
            RET
 END IRQ0__service

 IRQ1__service      PROCEDURE            ENTRY
            !...!
            AND     IRQ, #%(2)11111101              !clear IRQ1!
            !...!
            RET
 END IRQ1__service
 !...!
```

SECTION 7

Timer/Counter Functions

The Z8 provides two 8-bit timer/counters, T_0 and T_1, which are adaptable to a variety of application needs and thus allow the software (and external hardware) to be relieved of the bulk of such tasks. Included in the set of such uses are:

- Interval delay timer
- Maintenance of a time-of-day clock
- Watch-dog timer
- External event counting
- Variable pulse train output
- Duration measurement of external event
- Automatic delay following external event detection

Each timer/counter is driven by its own 6-bit prescaler, which is in turn driven by the internal Z8 clock divided by four. For T_1, the internal clock may be gated or triggered by an external event or may be replaced by an external clock input. Each timer/counter may operate in either single-pass or continuous mode where, at end-of-count, either counting stops or the counter reloads and continues counting. The counter and prescaler registers may be altered individually while the timer/counter is running; the software controls whether the new values are loaded immediately or when end-of-count (EOC) is reached.

Although the timer/counter prescaler registers (PRE0 and PRE1) are write-only, there is a technique by which the timer/

7. Timer/ Counter Functions (Continued)

counters may simulate a readable prescaler. This capability is a requirement for high resolution measurement of an event's duration. The basic approach requires that one timer/counter be initialized with the desired counter and prescaler values. The second timer/counter is initialized with a counter equal to the prescaler of the first timer/counter and a prescaler of 1. The second timer/counter must be programmed for continuous mode. With both timer/counters driven by the internal clock and started and stopped simultaneously, they will run synchronous to one another; thus, the value read from the second counter will always be equivalent to the prescaler of the first.

7.1 Time/Count Interval Calculation To determine the time interval (i) until EOC, the equation

$$i = t \times p \times v$$

characterizes the relation between the prescaler (p), counter (v), and clock input period (t); t is given by

$$1/(XTAL/8)$$

where XTAL is the Z8 input clock frequency; p is in the range 1 – 64; v is in the range 1 – 256. When programming the prescaler and counter registers, the maximum load value is truncated to six and eight bits, respectively, and is therefore programmed as zero. For an input clock frequency of 8 MHz, the prescaler and counter register values may be programmed to time an interval in the range

$$1\ \mu s \times 1 \times 1 \leq i \leq 1\ \mu s \times 64 \times 256$$

$$1\ \mu s \leq i \leq 16.384\ ms$$

To determine the count (c) until EOC for T_1 with external clock input, the equation

$$c = p \times v$$

characterizes the relation between the T_1 prescaler (p) and the T_1 counter (v). The divide-by-8 on the input frequency is bypassed in this mode. The count range is

$$1 \times 1 \leq c \leq 64 \times 256$$

$$1 \leq c \leq 16{,}384$$

7.2 T_{OUT} Modes. Port 3, bit 6 ($P3_6$) may be configured as an output (T_{OUT}) which is dynamically controlled by one of the following:

- T_0
- T_1
- Internal clock

When driven by T_0 or T_1, T_{OUT} is reset to a logic 1 when the corresonding load bit is set in timer control register TMR (%F1) and toggles on EOC from the corresponding counter. When T_{OUT} is driven by the internal clock, that clock is directly output on $P3_6$.

While programmed as T_{OUT}, $P3_6$ is disabled from being modified by a write to port register %03; however, its current output may be examined by the Z8 software by a read to port register %03.

7.3 T_{IN} Modes. Port 3, bit 1 ($P3_1$) may be configured as an input (T_{IN}) which is used in conjunction with T_1 in one of four modes:

- External clock input
- Gate input for internal clock
- Nonretriggerrable input for internal clock
- Retriggerable input for internal clock

For the latter two modes, it should be noted that the existence of a synchronizing circuit within the Z8 causes a delay of two to three internal clock periods following an external trigger before clocking of the counter actually begins.

Each High-to-Low transition on T_{IN} will generate interrupt request IRQ2, regardless of the selected T_{IN} mode or the enabled/disabled state of T_1. IRQ2 must therefore be masked or enabled according to the needs of the application.

The "external clock input" T_{IN} mode supports the counting of external events, where an event is seen as a High-to-Low transition on T_{IN}. Interrupt request IRQ5 is generated on the n^{th} occurrence (single-pass mode) or on every n^{th} occurrence (continuous mode) of that event.

The "gate input for internal clock" T_{IN} mode provides for duration measurement of an external event. In this mode, the T_1 prescaler is driven by the Z8 internal clock, gated by a High level on T_{IN}. In other words, T_1 will count while T_{IN} is High and stop counting while T_{IN} is Low. Interrupt request IRQ2 is generated on the High-to-Low transition on T_{IN}. Interrupt request IRQ5 is generated on T_1 EOC. This mode may be used when the width of a High-going pulse needs to be measured. In this mode, IRQ2 is typically the interrupt request of most importance, since it signals the end of the pulse being measured. If IRQ5 is generated prior to IRQ2 in this mode, the pulse width on T_{IN} is too large for T_1 to measure in a single pass.

The "nonretriggerable input" T_{IN} mode provides for automatic delay timing following an external event. In this mode, T_1 is loaded and clocked by the Z8 internal clock following the first High-to-Low transition on T_{IN} after T_1 is enabled. T_{IN} transitions that occur after this point do not affect T_1. In single-pass mode, the

7. Timer/ Counter Functions (Continued)

enable bit is reset on EOC; further T_{IN} transitions will not cause T_1 to load and begin counting until the software sets the enable bit again. In continuous mode, EOC does not modify the enable bit, but the counter is reloaded and counting continues immediately; IRQ5 is generated every EOC until software resets the enable bit. This T_{IN} mode may be used, for example, to time the line feed delay following end of line detection on a printer or to delay data sampling for some length of time following a sample strobe.

The "retriggerable input" T_{IN} mode will load and clock T_1 with the Z8 internal clock on every occurrence of a High-to-Low transition on T_{IN}. T_1 will time-out and generate interrupt request IRQ5 when the programmed time interval (determined by T_1 prescaler and load register values) has elapsed since the last High-to-Low transition on T_{IN}. In single-pass mode, the enable bit is reset on EOC; further T_{IN} transitions will not cause T_1 to load and begin counting until the software sets the enable bit again. In continuous mode, EOC does not modify the enable bit, but the counter is reloaded and counting continues immediately; IRQ5 is generated at every EOC until the software resets the enable bit. This T_{IN} mode may provide such functions as watch-dog timer (e.g., interrupt if conveyor belt stopped or clock pulse missed), or keyboard time-out (e.g., interrupt if no input in x ms).

7.4 Examples. Several possible uses of the timer/counters are given in the following four examples.

7.4.1 Time of Day Clock. The following module illustrates the use of T_1 for maintenance of a time of day clock, which is kept in binary format in terms of hours, minutes, seconds, and hundredths of a second. It is desired that the clock be updated once every hundredth of a second; therefore, T_1 is programmed in continuous mode to interrupt 100 times a second. Although T_1 is used for this example, T_0 is equally suited for the task.

The procedure for initializing the timer (TOD_INIT), the interrupt service routine (TOD) which updates the clock, and the interrupt vector for T_1 end-of-count (IRQ_5) are illustrated below. XTAL = 7.3728 MHz is assumed.

```
Z8ASM     2.0
LOC     OBJ CODE    STMT SOURCE STATEMENT

                       1 TIMER1  MODULE
                       2 CONSTANT
                       3  HOUR    :=       R12
                       4  MINUTE  :=       R13
                       5  SECOND  :=       R14
                       6  HUND    :=       R15
                       7         $SECTION PROGRAM
                       8 GLOBAL
                       9 !IRQ5 interrupt vector!
                      10         $ABS     10
P 0000 000F'          11 IRQ_5   ARRAY    [1 WORD]  :=     [TOD]
                      12
                      13         $REL
P 000C                14 TOD_INIT         PROCEDURE
                      15 ENTRY
P 0000 E6  F3  93     16         LD       PRE1,#%(2)10010011
                      17                           !bit 2-7: prescaler = 36;
                      18                            bit 1: internal clock;
                      19                            bit 0: continuous mode!
P 0003 E6  F2  00     20         LD       T1,#0     !(256) time-out =
                      21                            1/100 second!
P 0006 46  F1  0C     22         OR       TMR,#%0C  !load, enable T1!
P 0009 8F             23         DI
P 000A 46  FB  20     24         OR       IMR,#%20  !enable T1 interrupt!
P 000D 9F             25         EI
P 000E AF             26         RET
P 000F                27 END     TOD_INIT
                      28
P 000F                29 TOD     PROCEDURE
                      30 ENTRY
P 000F 70  FD         31         PUSH     RP
                      32 !Working register file %10 to %1F contains
                      33  the time of day clock!
P 0011 31  10         34         SRP      #%10
P 0013 FE             35         INC      HUND              !1 more .01 sec!
P 0014 A6  EF  64     36         CP       HUND,#100         !full second yet?!
P 0017 EB  13         37         JR       NE,TOD_EXIT       !jump if no!
P 0019 B0  EF         38         CLR      HUND
P 001B EE             39         INC      SECOND            !1 more second!
P 001C A6  EE  3C     40         CP       SECOND,#60        !full minute yet?!
P 001F EB  0B         41         JR       NE,TOD_EXIT       !jump if no!
```

7. Timer/ Counter Functions (Continued)

```
P 0021 B0  EE          42          CLR      SECOND
P 0023 DE              43          INC      MINUTE         !1 more minute!
P 0024 A6  ED  3C      44          CP       MINUTE,#60     !full hour yet?!
P 0027 EB  03          45          JR       NE,TOD_EXIT    !jump if no!
P 0029 B0  ED          46          CLR      MINUTE
P 002B CE              47          INC      HOUR
                       48 TOD_EXIT:
P 002C 50  FD          49          POP      RP             !restore entry RP!
P 002E BF              50          IRET
P 002F                 51 END      TOD
                       52 END      TIMER1

  0 ERRORS
ASSEMBLY COMPLETE
```

TOD_INIT:
7 instructions
15 bytes
16 µs

TOD:
17 instruction
32 bytes
19.5 µs (average) including interrupt response time

7.4.2 Variable Frequency, Variable Pulse Width Output. The following module illustrates one possible use of T_{OUT}. Assume it is necessary to generate a pulse train with a 10% duty cycle, where the output is repetitively high for 1.6 ms and then low for 14.4 ms. To do this, T_{OUT} is controlled by end-of-count from T_1, although T_0 could alternately be chosen. This example makes use of the Z8 feature that allows a timer's counter register to be modified without disturbing the count in progress. In continuous mode, the new value is loaded when T_1 reaches EOC. T_1 is first loaded and enabled with values to generate the short interval. The counter register is then immediately modified with the value to generate the long interval; this value is loaded into the counter automatically on T_1 EOC. The prescaler selected value must be the same for both long and short intervals. Note that the initial loading of the T_1 counter register is followed by setting the T_1 load bit of timer control register TMR (%F1); this action causes T_{OUT} to be reset to a logic 1 output. Each subsequent modification of the T_1 counter register does not affect the current T_{OUT} level, since the T_1 load bit is NOT altered by the software. The new value is loaded on EOC, and T_{OUT} will toggle at that time. The T_1 interrupt service routine should simply modify the T_1 counter register with the new value, alternating between the long and short interval values.

In the example which follows, bit 0 of register %04 is used as a software flag to indicate which value was loaded last. This module illustrates the procedure for T_1/T_{OUT} initialization (PULSE_INIT), the T_1 interrupt service routine (PULSE), and the interrupt vector for T_1 EOC (IRQ_5). XTAL = 8 MHz is assumed.

```
Z8ASM     2.0
LOC     OBJ CODE   STMT SOURCE STATEMENT

                      1 TIMER2  MODULE
                      2         $SECTION PROGRAM
                      3 GLOBAL
                      4 !IRQ5 interrupt vector!
                      5         $ABS     10
P 0000 0017'          6 IRQ_5   ARRAY    [1 WORD]  :=     [PULSE]
                      7
                      8         $REL
P 000C                9 PULSE_INIT       PROCEDURE
                     10 ENTRY
P 0000 E6  F3  03    11         LD       PRE1,#%(2)00000011
                     12                           !bit 2-7: prescaler = 64;
                     13                            bit 1: internal clock;
                     14                            bit 0: continuous mode!
P 0003 E6  F7  00    15         LD       P3M,#00  !bit 5: let P36 be Tout!
P 0006 E6  F2  19    16         LD       T1,#25          !for short interval!
P 0009 8F            17         DI
P 000A 46  FB  20    18         OR       IMR,#%(2)00100000  !enable T1 interrupt!
P 000D E6  F1  8C    19         LD       TMR,#%(2)10001100
                     20                           !bit 6-7: Tout controlled
                     21                                       by T1;
                     22                            bit 3: enable T1;
                     23                            bit 2: load T1 !
                     24 !Set long interval counter, to be loaded on T1 EOC!
P 0010 E6  F2  E1    25         LD       T1,#225
                     26 !Clear alternating flag for PULSE!
```

7. Timer/ Counter Functions (Continued)

```
P 0013 B0  04         27          CLR      %04                 != 0 : 25 next;
                      28                                        = 1 : 225 next !
P 0015 9F             29          EI
P 0016 AF             30          RET
P 0017                31 END      PULSE_INIT
                      32
                      33
P 0017                34 PULSE    PROCEDURE
                      35 ENTRY
P 0017 E6  F2  E1     36          LD       T1,#225             !new load value!
P 001A B6  04  01     37          XOR      %04,#1              !which value next?!
P 001D 6B  03         38          JR       Z,PULSE_EXIT        !should be 225!
P 001F E6  F2  19     39          LD       T1,#25              !should be 25!
                      40 PULSE_EXIT:
P 0022 BF             41          IRET
P 0023                42 END      PULSE
                      43 END      TIMER2

   0 ERRORS
ASSEMBLY COMPLETE
```

PULSE_INIT:	*PULSE:*
10 instructions	*5 instructions*
23 bytes	*12 bytes*
23 μs	*25 μs (average) including interrupt response time*

7.4.3 Cascaded Timer/Counters. For some applications it may be necessary to measure a greater time interval than a single timer/counter can measure (16.384 ms). In this case, T_{IN} and T_{OUT} may be used to cascade T_0 and T_1 to function as a single unit. T_{OUT}, programmed to toggle on T_0 end-of-count, should be wired back to T_{IN}, which is selected as the external clock input for T_1. With T_0 programmed for continuous mode, T_{OUT} (and therefore T_{IN}) goes through a High-to-Low transition (causing T_1 to count) on every other T_0 EOC. Interrupt request IRQ5 is generated when the programmed time interval has elapsed. Interrupt requests IRQ2 (generated on every T_{IN} High-to-Low transition) and IRQ4 (generated on T_0 EOC) are of no importance in this application and are therefore disabled.

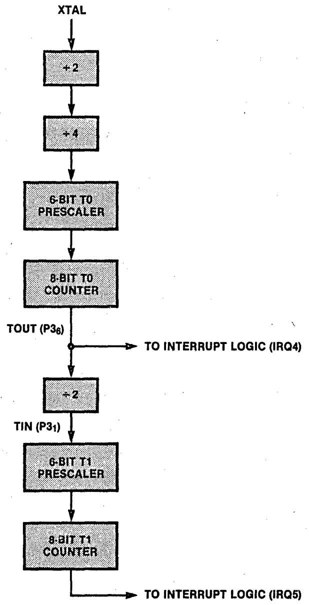


Figure 3. Cascaded Timer/Counters

To determine the time interval (i) until EOC, the equation

$$i = t \times p0 \times v0 \times (2 \times p1 \times v1 - 1)$$

characterizes the relation between the T_0 prescaler (p0) and counter (v0), the T_1 prescaler (p1) and counter (v1), and the clock input period (t); t is defined in Section 7.1. Assuming XTAL = 8 MHz, the measurable time interval range is

$$1\ \mu s \times 1 \times 1 \times (2 \times 1 - 1) \leq i \leq$$
$$1\ \mu s \times 64 \times 256 \times (2 \times 64 \times 256 - 1)$$

$$1\ \mu s \leq i \leq 536.854528\ s$$

Figure 3 illustrates the interconnection between T_0 and T_1. The following module illustrates the procedure required to initialize the timers for a 1.998 second delay interval:

```
Z8ASM     2.0
LOC    OBJ CODE    STMT SOURCE STATEMENT

                      1 TIMER3  MODULE
                      2 GLOBAL
P 0000                3 TIMER_16        PROCEDURE
                      4 ENTRY
P 0000 E6  F3  28     5         LD      PRE1,#%(2)00101000
                      6                           !bit 2-7: prescaler = 10;
                      7                            bit 1: external clock;
                      8                            bit 0: single-pass mode!
P 0003 E6  F7  00     9         LD      P3M,#00   !bit 5: let P36 be Tout!
P 0006 E6  F2  64    10         LD      T1,#100         !T1 counter register!
P 0009 E6  F5  29    11         LD      PRE0,#%(2)00101001
                     12                           !bit 2-7: prescaler = 10;
                     13                            bit 0: continuous mode!
P 000C E6  F4  64    14         LD      T0,#100         !T0 counter register!
P 000F 8F            15         DI
P 0010 56  FB  2B    16         AND     IMR,#%(2)00101011  !disable IRQ2 (Tin);
                     17                                        and IRQ4 (T0) !
P 0013 46  FB  20    18         OR      IMR,#%(2)00100000  !enable IRQ5 (T1)!
P 0016 9F            19         EI
P 0017 E6  F1  4F    20         LD      TMR,#%(2)01001111
                     21                           !bit 6-7: Tout controlled
                     22                                     by T0;
                     23                            bit 4-5: Tin mode is ext.
                     24                                     clock input;
                     25                            bit 3: enable T1;
                     26                            bit 2: load T1;
                     27                            bit 1: enable T0;
                     28                            bit 0: load T0 !
P 001A AF            29         RET
P 001B               30 END     TIMER_16
                     31 END     TIMER3

   0 ERRORS
ASSEMBLY COMPLETE
```

11 instructions
27 bytes
26.5 µs

7.4.4 Clock Monitor. T_1 and T_{IN} may be used to monitor a clock line (in a diskette drive, for example) and generate an interrupt request when a clock pulse is missed. To accomplish this, the clock line to be monitored is wired to $P3_1$ (T_{IN}). T_{IN} should be programmed as a retriggerable input to T_1, such that each falling edge on T_{IN} will cause T_1 to reload and continue counting. If T_1 is programmed to time-out after an interval of one-and-a-half times the clock period being monitored, T_1 will time-out and generate interrupt request IRQ5 only if a clock pulse is missed.

The following module illustrates the procedure for initializing T_1 and T_{IN} (MONITOR__INIT) to monitor a clock with a period of 2 µs. XTAL = 8 MHz is assumed. Note that this example selects single-pass rather than continuous mode for T_1. This is to prevent a continuous stream of IRQ5 interrupt requests in the event that the monitored clock fails completely. Rather, the interrupt service routine (CLK__ERR) is left with the choice of whether or not to re-enable the monitoring. Also shown is the T_1 interrupt vector (IRQ__5).

```
Z8ASM     2.0
LOC    OBJ CODE    STMT SOURCE STATEMENT

                      1 TIMER4  MODULE
                      2         $SECTION PROGRAM
                      3 GLOBAL
                      4 !IRQ5 interrupt vector!
                      5         $ABS    10
P 0000 0015'          6 IRQ_5   ARRAY   [1 WORD]  :=    [CLK_ERR]
                      7
                      8         $REL
P 000C                9 MONITOR_INIT    PROCEDURE
                     10 ENTRY
P 0000 E6  F3  04    11         LD      PRE1,#%(2)00000100
                     12                           !bit 2-7: prescaler = 1;
                     13                            bit 1: external clock;
                     14                            bit 0: single-pass mode!
P 0003 E6  F7  00    15         LD      P3M,#00   !bit 5: let P36 be Tout!
P 0006 E6  F2  03    16         LD      T1,#3           !T1 load register,
                     17                                 = 1.5 * 2 usec   !
```

7. Timer/ Counter Functions (Continued)

```
P 0009 8F           18          DI
P 000A 56  FB  3B   19          AND    IMR,#%(2)00111011  !disable IRQ2 (Tin)!
P 000D 46  FB  20   20          OR     IMR,#%(2)00100000  !enable IRQ5 (T1)!
P 0010 9F           21          EI
                    22
P 0011 E6  F1  38   23          LD     TMR,#%(2)00111000
                    24                             !bit 4-5: Tin mode is
                    25                                  retrig. input;
                    26                              bit 3: enable T1 !
P 0014 AF           27          RET
P 0015              28 END      MONITOR_INIT
                    29
                    30
P 0015              31 CLK_ERR PROCEDURE
                    32 ENTRY
                    33          !...!          !handle the missed clock!
                    34
                    35 !if clock monitoring should continue...!
P 0015 46  F1  08   36          OR     TMR,#%(2)00001000
                    37                             !bit 3: enable T1 !
P 0018 BF           38          IRET
P 0019              39 END      CLK_ERR
                    40 END      TIMER4

   0 ERRORS
ASSEMBLY COMPLETE
```

MONITOR_INIT:	*CLK_ERR:*
9 instructions	*2 + instructions*
21 bytes	*4 + bytes*
21.5 μs	*18.5 + μs including interrupt response time*

SECTION 8

I/O Functions

The Z8 provides 32 I/O lines mapped into registers 0-3 of the internal register file. Each nibble of port 0 is individually programmable as input, output, or address/data lines (A_{15}-A_{12}, A_{11}-A_8). Port 1 is programmable as a single entity to provide input, output, or address/data lines (AD_7-AD_0). The operating modes for the bits of Ports 0 and 1 are selected by control register P01M (%F8). Selection of I/O lines as address/data lines supports access to external program and data memory; this is discussed in Section 3. Each bit of Port 2 is individually programmable as an input or an output bit. Port 2 bits programmed as outputs may also be programmed (via bit 0 of P3M) to all have active pull-ups or all be open-drain (active pull-ups inhibited). In Port 3, four bits ($P3_0$-$P3_3$) are fixed as inputs, and four bits ($P3_4$-$P3_7$) are fixed as outputs, but their functions are programmable. Special functions provided by Port 3 bits are listed in Table 4. Use of the Data Memory select output is discussed in Section 3; uses of T_{IN} and T_{OUT} are discussed in Section 7.

Function	Bit	Signal
Handshake	$P3_1$	$\overline{\text{DAV}}$2/RDY2
	$P3_2$	$\overline{\text{DAV}}$0/RDY0
	$P3_3$	$\overline{\text{DAV}}$1/RDY1
	$P3_4$	RDY1/$\overline{\text{DAV}}$1
	$P3_5$	RDY0/$\overline{\text{DAV}}$0
	$P3_6$	RDY2/$\overline{\text{DAV}}$2
Interrupt Request	$P3_0$	IRQ3
	$P3_1$	IRQ2
	$P3_2$	IRQ0
	$P3_3$	IRQ1
Counter/ Timer	$P3_1$	T_{IN}
	$P3_6$	T_{OUT}
Data Memory Select Status Out	$P3_4$	$\overline{\text{DM}}$
Serial I/O	$P3_0$	Serial In
	$P3_7$	Serial Out

Table 4. Port 3 Special Functions

8.1 Asynchronous Receiver/Transmitter Operation. Full-duplex, serial asynchronous receiver/transmitter operation is provided by the Z8 via $P3_7$ (output) and $P3_0$ (input) in conjunction with control register SIO (%F0), which is actually two registers: receiver buffer and transmitter buffer. Counter/Timer T_0 provides the clock for control of the bit rate.

The Z8 always receives and transmits eight bits between start and stop bits. However, if parity is enabled, the eighth bit (D_7) is replaced by the odd-parity bit when transmitted and a parity-error flag (= 1 if error) when received. Table 5 illustrates the state of the parity bit/parity error flag during serial I/O with parity enabled.

Although the Z8 directly supports either odd parity or no parity for serial I/O operation, even parity may also be provided with additional software support. To receive and transmit with even parity, the Z8 should be configured for serial I/O with odd parity disabled. The Z8 software must calculate parity

8. I/O Functions (Continued)

Character Loaded Into SIO	Transmitted To Serial Line	Received From Serial Line	Character Transferred To SIO	Note*
11000011	01000011	01000011	01000011	no error
11000011	01000011	01000111	11000111	error
01111000	11111000	11111000	01111000	no error
01111000	11111000	01111000	11111000	error

Table 5. Serial I/O With Odd Parity

* Left-most bit is D7

and modify the eighth bit prior to the load of a character into SIO and then modify a parity error flag following the load of a character from SIO. All other processing required for serial I/O (e.g. buffer management, error handling, etc.) is the same as that for odd parity operations.

To configure the Z8 for Serial I/O, it is necessary to:

- Enable $P3_0$ and $P3_7$ for serial I/O and select parity,
- Set up T_0 for the desired bit rate,
- Configure IRQ3 and IRQ4 for polled or automatic interrupt mode,
- Load and enable T_0.

To enable $P3_0$ and $P3_7$ for serial I/O, bit 6 of P3M (R247) is set. To enable odd parity, bit 7 of P3M is set; to disable it, the bit is reset. For example, the instruction

```
LD      P3M,#%40
```

will enable serial I/O, but disable parity. The instruction

```
LD      P3M,#%C0
```

will enable serial I/O, and enable odd parity.

In the following discussions, bit rate refers to all transmitted bits, including start, stop, and parity (if enabled). The serial bit rate is given by the equation:

$$\text{bit rate} = \frac{\text{input clock frequency}}{(2 \times 4 \times T_0 \text{ prescaler} \times T_0 \text{ counter} \times 16)}$$

The final divide-by-16 is incurred for serial communications, since in this mode T_0 runs at 16 times the bit rate in order to synchronize the data stream. To configure the Z8 for a specific bit rate, appropriate values must first be selected for T_0 prescaler and T_0 counter by the above equation; these values are then programmed into registers T_0 (%F4) and PRE0 (%F5) respectively. Note that PRE0 also controls the continuous vs. single-pass mode for T_0; continuous mode should be selected for serial I/O. For example, given an input clock frequency of 7.3728 MHz and a selected bit rate of 9600 bits per second, the equation is satisfied by T_0 counter = 2 and prescaler = 3. The following code sequence will configure the T_0 counter and T_0 prescaler registers:

```
LD  T0,#2   !T0 counter = 2!
LD  PRE0,#%(2)00001101
              !bit 2-7: prescaler = 3; bit 0:
              continuous mode!
```

Interrupt request 3 (IRQ3) is generated whenever a character is transferred into the receive buffer; interrupt request 4 (IRQ4) is generated whenever a character is transferred out of the transmit buffer. Before accepting such interrupt requests, the Interrupt Mask, Request, and Priority Registers (IMR, IRQ, and IPR) must be programmed to configure the mode of interrupt response. The section on Interrupt Processing provides a discussion of interrupt configurations.

To load and enable T_0, set bits 0 and 1 of the timer mode register (TMR) via an instruction such as

```
OR   TMR,#%03
```

This will cause the T_0 prescaler and counter registers (PRE0 and T_0) to be transferred to the T_0 prescaler and counter. In addition, T_0 is enabled to count, and serial I/O operations will commence.

Characters to be output to the serial line should be written to serial I/O register SIO (%F0). IRQ4 will be generated when all bits have been transferred out.

Characters input from the serial line may be read from SIO. IRQ3 will be generated when a full character has been transferred into SIO.

The following module illustrates the receipt of a character and its immediate echo back to the serial line. It is assumed that the Z8 has been configured for serial I/O as described above, with IRQ3 (receive) enabled to interrupt, and IRQ4 (transmit) configured to be polled. The received character is stored in a circular buffer in register memory from address %42 to %5F. Register %41 contains the address of the next available buffer position and should have been initialized by some earlier routine to #%42.

```
Z8ASM     2.0
LOC     OBJ CODE    STMT SOURCE STATEMENT

                       1 SERIAL_IO        MODULE
                       2 CONSTANT
                       3  next_addr       :=      %41
                       4  start           :=      %42
                       5  length          :=      %1E
                       6 $SECTION PROGRAM
                       7 GLOBAL
                       8 !IRQ3 vector!
                       9         $ABS     6
P 0006 0000'          10 IRQ_3   ARRAY [1 WORD] := [GET_CHARACTER]
                      11
                      12         $REL     0
P 0000                13 GET_CHARACTER   PROCEDURE       ENTRY
                      14
                      15 !Serial I/O receive interrupt service!
                      16 !Echo received character and wait for
                      17  echo completion!
P 0000 E4  F0  F0     18         ld       SIO,SIO         !echo!
                      19
                      20 !save it in circular buffer!
P 0003 F5  F0  41     21         ld       @next_addr,SIO  !save in buffer!
P 0006 20  41         22         inc      next_addr       !point to next position!
P 0008 A6  41  60     23         cp       next_addr,#start+length
                      24                                  !wrap-around yet?!
P 000B EB  03         25         jr       ne,echo_wait    !no.!
P 000D E6  41  42     26         ld       next_addr,#start !yes. point to start!
                      27 !now, wait for echo complete!
                      28 echo_wait:
P 0010 66  FA  10     29         tcm      IRQ,#%10        !transmitted yet?!
P 0013 EB  FB         30         jr       nz,echo_wait    !not yet!
                      31
P 0015 56  FA  EF     32         and      IRQ,#%EF        !clear IRQ4!
P 0018 BF             33         IRET                     !return from interrupt!
P 0019                34 END     GET_CHARACTER
                      35 END     SERIAL_IO

   0 ERRORS
ASSEMBLY COMPLETE
```

10 instructions
25 bytes
35.5 µs + 5.5 µs for each additional pass through the echo_wait loop, including interrupt response time

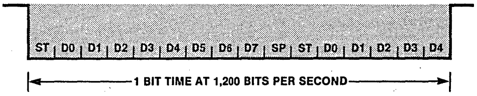
8.2 Automatic Bit Rate Detection. In a typical system, where serial communication is required (e.g. system with a terminal), the desired bit rate is either user-selectable via a switch bank or nonvariable and "hard-coded" in the software. As an alternate method of bit-rate detection, it is possible to automatically determine the bit rate of serial data received by measuring the length of a start bit. The advantage of this method is that it places no requirements on the hardware design for this function and provides a convenient (automatic) operator interface.

In the technique described here, the serial channel of the Z8 is initialized to expect a bit rate of 19,200 bits per second. The number of bits (n) received through Port pin P30 for each bit transmitted is expressed by

$$n = 19{,}200/b$$

where b = transmission bit rate. For example, if the transmission bit rate were 1200 bits per second, each incoming bit would appear to the receiving serial line as 19,200/1200 or 16 bits.

The following example is capable of distinguishing between the bit rates shown in Table 6 and assumes an input clock frequency of 7.3728 MHz, a T_0 prescaler of 3, and serial I/O enabled with parity disabled. This example requires that a character with its low order bit = 1 (such as a carriage return) be sent to the serial channel. The start bit of this character can be measured by counting the number of zero bits collected before the low order 1 bit. The number of zero bits actually collected into data bits by the serial channel is less than n (as given in the above equation), due to the detection of start and stop bits. Figure 4 illustrates the collection (at 19,200


ST D0 D1 D2 D3 D4 D5 D6 D7 SP ST D0 D1 D2 D3 D4
1 BIT TIME AT 1,200 BITS PER SECOND

ST = START BIT SP = STOP BIT Dn = DATA BIT n

EACH INTERVAL SHOWN = 1 BIT TIME
AT 19,200 BITS PER SECOND

Figure 4. Collection of a Start Bit Transmitted at at 19,200 BPS

Bit Rate	Number of Bits Received Per Bit Transmitted	Number of 0 Bits Collected as Data Bits		T_0 Counter	
		dec	binary	dec	binary
19200	1	0	00000000	1	00000001
9600	2	1	00000001	2	00000010
4800	4	3	00000011	4	00000100
2400	8	7	00000111	8	00001000
1200	16	13	00001101	16	00010000
600	32	25	00011001	32	00100000
300	64	49	00110001	64	01000000
150	128	97	01100001	128	10000000

Table 6. Inputs to the Automatic Bit Rate Detection Algorithm

bits per second) of a zero bit transmitted to the Z8 at 1,200 bits per second. Notice that only 13 of the 16 zero bits received are collected as data bits.

Once the number of zero bits in the start bit has been collected and counted, it remains to translate this count into the appropriate T_0 counter value and program that value into T_0 (%F4). The patterns shown in the two binary columns of Table 6 are utilized in the algorithm for this translation.

As a final step, if incoming data is to commence immediately, it is advisable to wait until the remainder of the current "elongated" character has been received, thus "flushing" the serial line. This can be accomplished either via a software loop, or by programming T_1 to generate an interrupt request after the appropriate amount of time has elapsed. Since a character is composed of eight bits plus a minimum of one stop bit following the start bit, the length of time to delay may be expressed as

$$(9 \times n)/b$$

where n and b are as defined above. The following module illustrates a sample program for automatic bit rate detection.

```
Z8ASM      2.0
LOC     OBJ CODE    STMT SOURCE STATEMENT

                       1 bit_rate         MODULE
                       2 EXTERNAL
                       3 DELAY    PROCEDURE
                       4 GLOBAL
P 0000                 5 main     PROCEDURE
                       6          ENTRY
P 0000 8F              7          di                     !disable interrupts!
P 0001 56  FB  77      8          and      IMR,#%77      !IRQ3 polled mode!
P 0004 56  FA  F7      9          and      IRQ,#%F7      !clear IRQ3!
P 0007 E6  F7  40     10          ld       P3M,#%40      !enable serial I/O!
P 000A E6  F4  01     11          ld       TO,#1
P 000D E6  F5  0D     12          ld       PREO,#(3 SHL 2)+1  !bit rate = 19,200;
                      13                                   continuous count mode!
P 0010 B0  E0         14          clr      R0            !init. zero byte counter!
P 0012 E6  F1  03     15          ld       TMR,#3        !load and enable TO!
                      16
                      17 !collect input bytes by counting the number of null
                      18  characters received.  Stop when non-zero byte received!
                      19 collect:
P 0015 76  FA  08     20          TM       IRQ,#%08      !character received?!
P 0018 6B  FB         21          jr       z,collect     !not yet!
P 001A 18  F0         22          ld       R1,SIO        !get the character!
P 001C 56  FA  F7     23          and      IRQ,#%F7      !clear interrupt request!
P 001F 1E             24          inc      R1            !compare to 0 ...!
P 0020 1A  05         25          djnz     R1,bitloop    !...(in 3 bytes of code)!
P 0022 06  E0  08     26          add      R0,#8         !update count of 0 bits!
P 0025 8B  EE         27          jr       collect
                      28 bitloop:                        !add in zero bits from low
                      29                                  end of 1st non-zero byte!
P 0027 E0  E1         30          RR       R1
P 0029 7B  03         31          jr       c,count_done
P 002B 0E             32          inc      R0
P 002C 8B  F9         33          jr       bitloop
                      34
                      35 !R0 has number of zero bits collected!
                      36 !translate R0 to the appropriate TO counter value!
                      37 count_done:                     !R0 has count of zero bits!
P 002E 1C  07         38          ld       R1,#7
P 0030 2C  80         39          ld       R2,#%80       !R2 will have TO counter value!
P 0032 90  E0         40          RL       R0
                      41
P 0034 90  E0         42 loop:    RL       R0
```

```
P 0036 7B  04             jr      c,done
P 0038 E0  E2             RR      R2
P 003A 1A  F8             djnz    r1,loop

P 003C 29  F4       done: ld      T0,R2      !load value for detected
                                                   bit rate!
                    !Delay long enough to clear serial line of bit stream!
P 003E D6  0000*          call    DELAY
                    !clear receive interrupt request!
P 0041 56  FA  F7         and     IRQ,#%F7

P 0044              END   main
                    END   bit_rate

   0 ERRORS
ASSEMBLY COMPLETE
```

30 instructions
68 bytes
Execution time is variable based on transmission bit rate.

8.3 Port Handshake. Each of Ports 0, 1 and 2 may be programmed to function under input or output handshake control. Table 7 defines the port bits used for the handshaking and the mode bit settings required to select handshaking. To input data under handshake control, the Z8 should read the input port when the $\overline{\text{DAV}}$ input goes Low (signifying that data is available from the attached device). To output data under handshake control, the Z8 should write the output port when the RDY input goes Low (signifying that the previously output data has been accepted by the attached device). Interrupt requests IRQ0, IRQ1, and IRQ2 are generated by the falling edge of the handshake signal input to the Z8 for Port 0, Port 1, and Port 2 respectively. Port handshake operations may therefore be processed under interrupt control.

Consider a system that requires communication of eight parallel bits of data under handshake control from the Z8 to a peripheral device and that Port 2 is selected as the output port. The following assembly code illustrates the proper sequence for initializing Port 2 for output handshake.

```
CLR  P2M   !Port 2 mode register: all Port
            2 bits are outputs!
OR   %03,#%40
           !set DAV2: data not available!
LD   P3M,#%20
           !Port 3 mode register: enable
            Port 2 handshake!
LD   %02,DATA
           !output first data byte; DAV2
            will be cleared by the Z8 to
            indicate data available to
            the peripheral device!
```

Note that following the initialization of the output sequence, the software outputs the first data byte without regard to the state of the RDY2 input; the Z8 will automatically hold $\overline{\text{DAV}}$2 High until the RDY2 input is High. The peripheral device should force the Z8 RDY2 input line Low after it has latched the data in response to a Low on $\overline{\text{DAV}}$2. The Low on RDY2 will cause the Z8 to automatically force $\overline{\text{DAV}}$2 High until the next byte is output. Subsequent bytes should be output in response to interrupt request IRQ2 (caused by the High-to-Low transition on RDY2) in either a polled or an enabled interrupt mode.

	Port 0	**Port 1**	**Port 2**
Input handshake lines	$P3_2$ = $\overline{\text{DAV}}$ $P3_5$ = RDY	$P3_3$ = $\overline{\text{DAV}}$ $P3_4$ = RDY	$P3_1$ = $\overline{\text{DAV}}$ $P3_6$ = RDY
Output handshake lines	$P3_2$ = RDY $P3_5$ = $\overline{\text{DAV}}$	$P3_3$ = RDY $P3_4$ = $\overline{\text{DAV}}$	$P3_1$ = RDY $P3_6$ = $\overline{\text{DAV}}$
To select input handshake:	set bit 6 & reset bit 7 of P01M (program high nibble as input)	set bit 3 & reset bit 4 of P01M (program byte as input)	set bit 7 of P2M (program high bit as input)
To select output handshake:	reset bits 6, 7 of P01M (program high nibble as output)	reset bits 3, 4 of P01M (program byte as output)	reset bit 7 of P2M (program high bit as output)
To enable handshake:	set bit 5 of Port 3 ($P3_5$); set bit 2 of P3M	set bit 4 of Port 3 ($P3_4$); set bits 3, 4 of P3M	set bit 6 of Port 3 ($P3_6$); set bit 5 of P3M

Table 7. Port Handshake Selection

Arithmetic Routines

This section gives examples of the arithmetic and rotate instructions for use in multiplication, division, conversion, and BCD arithmetic algorithms.

9.1 Binary to Hex ASCII. The following module illustrates the use of the ADD and SWAP arithmetic instructions in the conversion of a 16-bit binary number to its hexadecimal ASCII representation. The 16-bit number is viewed as a string of four nibbles and is processed one nibble at a time from left to right, beginning with the high-order nibble of the lower memory address. %30 is added to each nibble if it is in the range 0 to 9; otherwise %37 is added. In this way, %0 is converted to %30, %1 to %31, . . . %A to %41, . . . %F to %46. Figure 5 illustrates the conversion of RR0 (contents = %F2BE) to its hex ASCII equivalent; the destination buffer is pointed to by RR4.

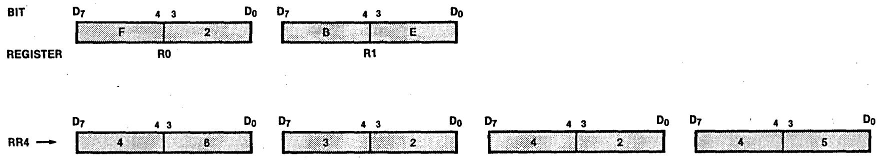


Figure 5. Conversion of (RR0) to Hex ASCII

```
Z8ASM      2.99     INTERNAL RELEASE
LOC     OBJ CODE     STMT SOURCE STATEMENT

                       1 ARITH   MODULE
                       2 GLOBAL
P 0000                 3 BINASC  PROCEDURE
                       4 !*******************************************************
                       5  Purpose =      To convert a 16-bit binary
                       6                 number to Hex ASCII
                       7
                       8  Input =        RR0 = 16-bit binary number.
                       9                 RR4 = pointer to destination
                      10                       buffer in external memory.
                      11
                      12  Output =       Resulting ASCII string (4 bytes)
                      13                 in destination buffer.
                      14                 RR4 incremented by 4 .
                      15                 R0,R2,R6 destroyed.
                      16 ******************************************************!
                      17 ENTRY
                      18
P 0000 6C  04         19         ld      R6,#%04 !nibble count!
P 0002 F0  E0         20 again:  SWAP    R0      !look at next nibble!
P 0004 28  E0         21         ld      R2,R0
P 0006 56  E2  0F     22         and     R2,#%0F !isolate 4 bits!
                      23 !convert to ASCII : R2 + #%30 if R0 in range 0 to 9
                      24                 else  R2 + #%37 (in range 0A to 0F)
                      25 !
P 0009 06  E2  30     26         ADD     R2,#%30
P 000C A6  E2  3A     27         cp      R2,#%3A
P 000F 7B  03         28         jr      ult,skip
P 0011 06  E2  07     29         ADD     R2,#%07
P 0014 92  24         30 skip:   lde     @RR4,R2          !save ASCII in buffer!
P 0016 A0  E4         31         incw    RR4              !point to next
                      32                                   buffer position!
P 0018 A6  E6  03     33         cp      R6,#%03 !time for second byte?!
P 001B EB  02         34         jr      ne,same_byte     !no.!
P 001D 08  E1         35         ld      R0,R1            !2nd byte!
                      36 same_byte:
P 001F 6A  E1         37         djnz    R6,again
P 0021 AF             38         ret
P 0022                39 END     BINASC
                      40 END     ARITH

   0 errors
Assembly complete
```

15 instructions
34 bytes
120.5 µs (average)

9.2 BCD Addition. The following module illustrates the use of the add with carry (ADC) and decimal adjust (DA) instructions for the addition of two unsigned BCD strings of equal length. Within a BCD string, each nibble represents a decimal digit (0-9). Two such digits are packed per byte with the most significant digit in bits 7-4. Bytes within a BCD string are arranged in memory with the most significant digits stored in the lowest memory location. Figure 6 illustrates the representation of 5970 in a 6-digit BCD string, starting in register %33.

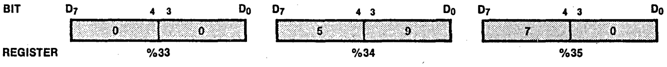


Figure 6. Unsigned BCD Representation

```
Z8ASM      2.0
LOC     OBJ CODE   STMT SOURCE STATEMENT

                      1 ARITH   MODULE
                      2 CONSTANT
                      3  BCD_SRC := R1
                      4  BCD_DST := R0
                      5  BCD_LEN := R2
                      6 GLOBAL
P 0000                7 BCDADD  PROCEDURE
                      8 !#####################################################
                      9  Purpose =       To add two packed BCD strings of
                     10                  equal length.
                     11                  dst  <-- dst + src
                     12
                     13  Input =         R0 = pointer to dst BCD string.
                     14                  R1 = pointer to src BCD string.
                     15                  R2 = byte count in BCD string
                     16                       (digit count = (R2)*2 ).
                     17
                     18  Output =        BCD string pointed to by R0 is
                     19                  the sum.
                     20                  Carry FLAG = 1 if overflow.
                     21                  R0 , R1 as on entry.
                     22                  R2 = 0
                     23 #####################################################!
                     24 ENTRY
                     25
P 0000 02   12       26         add      BCD_SRC,BCD_LEN !start at least... !
P 0002 02   02       27         add      BCD_DST,BCD_LEN !significant digits!
P 0004 CF            28         rcf                      !carry = 0!
                     29 add_again:
P 0005 00   E1       30         dec      BCD_SRC         !point to next two
                     31                                   src digits!
P 0007 00   E0       32         dec      BCD_DST         !point to next two
                     33                                   dst digits!
P 0009 E3   31       34         ld       R3,@BCD_SRC     !get src digits!
P 000B 13   30       35         ADC      R3,@BCD_DST     !add dst digits!
P 000D 40   E3       36         DA       R3              !decimal adjust!
P 000F F3   03       37         ld       @BCD_DST,R3     !move to dst!
P 0011 2A   F2       38         djnz     BCD_LEN,add_again  !loop for next
                     39                                        digits!
P 0013 AF            40         ret                      !all done!
                     41
P 0014               42 END     BCDADD
                     43 END     ARITH

   0 ERRORS
ASSEMBLY COMPLETE
```

11 instructions
20 bytes
Execution time is a function of the number of bytes (n) in input BCD string:
20 μs + 12.5 (n - 1) μs

9. Arithmetic Routines (Continued)

9.3 Multiply. The following module illustrates an efficient algorithm for the multiplication of two unsigned 8-bit values, resulting in a 16-bit product. The algorithm repetitively shifts the multiplicand right (using RRC), with the low-order bit being shifted out (into the carry flag). If a one is shifted out, the multiplier is added to the high-order byte of the partial product. As the high-order bits of the multiplicand are vacated by the shift, the resulting partial-product bits are rotated in. Thus, the multiplicand and the low byte of the product occupy the same byte, which saves register space, code, and execution time.

```
Z8ASM     2.99    INTERNAL RELEASE
LOC      OBJ CODE   STMT SOURCE STATEMENT

                       1 ARITH   MODULE
                       2 CONSTANT
                       3  MULTIPLIER      :=      R1
                       4  PRODUCT_LO      :=      R3
                       5  PRODUCT_HI      :=      R2
                       6  COUNT           :=      R0
                       7 GLOBAL
P 0000                 8 MULT    PROCEDURE
                       9 !*****************************************************
                      10  Purpose =       To perform an 8-bit by 8-bit unsigned
                      11                  binary multiplication.
                      12
                      13  Input =         R1 = multiplier
                      14                  R3 = multiplicand
                      15
                      16  Output =        RR2 = product
                      17                  R0  destroyed
                      18 *****************************************************!
                      19 ENTRY
P 0000 0C  09         20         ld      COUNT,#9        !8 BITS + 1!
P 0002 B0  E2         21         clr     PRODUCT_HI      !INIT HIGH RESULT BYTE!
P 0004 CF             22         RCF                     !CARRY = 0!
P 0005 C0  E2         23 LOOP:   RRC     PRODUCT_HI
P 0007 C0  E3         24         RRC     PRODUCT_LO
P 0009 FB  02         25         jr      NC,NEXT
P 000B 02  21         26         ADD     PRODUCT_HI,MULTIPLIER
P 000D 0A  F6         27 NEXT:   djnz    COUNT,LOOP
P 000F AF             28         ret
P 0010                29 END     MULT
                      30 END     ARITH

   0 errors
Assembly complete
```

9 instructions
16 bytes
92.5 μs (average)

9.4 Divide. The following module illustrates an efficient algorithm for the division of a 16-bit unsigned value by an 8-bit unsigned value, resulting in an 8-bit unsigned quotient. The algorithm repetitively shifts the dividend left (via RLC). If the high-order bit shifted out is a one or if the resulting high-order dividend byte is greater than or equal to the divisor, the divisor is subtracted from the high byte of the dividend. As the low-order bits of the dividend are vacated by the shift left, the resulting partial-quotient bits are rotated in. Thus, the quotient and the low byte of the dividend occupy the same byte, which saves register space, code, and execution time.

9. Arithmetic Routines (Continued)

```
Z8ASM     2.0
LOC     OBJ CODE   STMT SOURCE STATEMENT

                      1 ARITH   MODULE
                      2 CONSTANT
                      3  COUNT          :=      R0
                      4  DIVISOR        :=      R1
                      5  DIVIDEND_HI    :=      R2
                      6  DIVIDEND_LO    :=      R3
                      7 GLOBAL
P 0000                8 DIVIDE  PROCEDURE
                      9 !*****************************************************
                     10  Purpose =      To perform a 16-bit by 8-bit unsigned
                     11                 binary division.
                     12
                     13  Input =        R1 = 8-bit divisor
                     14                 RR2 = 16-bit dividend
                     15
                     16  Output =       R3  = 8-bit quotient
                     17                 R2  = 8-bit remainder
                     18                 Carry flag = 1 if overflow
                     19                            = 0 if no overflow
                     20 *****************************************************!
                     21 ENTRY
P 0000 OC  08        22         ld      COUNT,#8        !LOOP COUNTER!
                     23
                     24 !CHECK IF RESULT WILL FIT IN 8 BITS!
P 0002 A2  12        25         cp      DIVISOR,DIVIDEND_HI
P 0004 BB  02        26         jr      UGT,LOOP        !CARRY = 0 (FOR RLC)!
                     27 !WON'T FIT.  OVERFLOW!
P 0006 DF            28         SCF                     !CARRY = 1!
P 0007 AF            29         ret
                     30
                     31 LOOP:    !RESULT WILL FIT.  GO AHEAD WITH DIVISION!
P 0008 10  E3        32         RLC     DIVIDEND_LO     !DIVIDEND * 2!
P 000A 10  E2        33         RLC     DIVIDEND_HI
P 000C 7B  04        34         jr      c,subt
P 000E A2  12        35         cp      DIVISOR,DIVIDEND_HI
P 0010 BB  03        36         jr      UGT,next        !CARRY = 0!
P 0012 22  21        37 subt:   SUB     DIVIDEND_HI,DIVISOR
P 0014 DF            38         SCF                 !TO BE SHIFTED INTO RESULT!
P 0015 0A  F1        39 next:   djnz    COUNT,LOOP      !no flags affected!
                     40
                     41 !ALL    DONE!
P 0017 10  E3        42         RLC     DIVIDEND_LO
                     43                                 !CARRY = 0: no overflow!
P 0019 AF            44         ret
P 001A               45 END DIVIDE
                     46 END ARITH

   0 ERRORS
ASSEMBLY COMPLETE
```

15 instructions
26 bytes
124.5 μs (average)

SECTION 10

Conclusion

This Application Note has focused on ways in which the Z8 microcomputer can easily yet effectively solve various application problems. In particular, the many sample routines illustrated in this document should aid the reader in using the Z8 to greater advantage. The major features of the Z8 have been described so that the user can continue to expand and explore the Z8's repertoire of uses.

Z8® Subroutine Library

Zilog

Application Note

April 1982

INTRODUCTION

This application note describes a preprogrammed Z8601 MCU that contains a bootstrap to external program memory and a collection of general-purpose subroutines. Routines in this application note can be implemented with a Z8 Protopack and a 2716 EPROM programmed with the bootstrap and subroutine library.

In a system, the user's software resides in external memory beginning at hexidecimal address 0800. This software can use any of the subroutines in the library wherever appropriate for a given application. This application example makes certain assumptions about the environment; the reader should exercise caution when copying these programs for other cases.

Following RESET, software within the subroutine library is executed to initialize the control registers (Table 1). The control register selections can be subsequently modified by the user's program (for example, to use only 12 bits of Ports 0 and 1 for addressing external memory). Following control register initialization, an EI

Table 1. Control Register Initialization

Control Register			
Name	Address	Initial Value	Meaning
TMR	F1H	00H	T0 and T1 disabled
P2M	F6H	FFH	$P2_0$-$P2_7$: inputs
P3M	F7H	10H	P2 pull-ups open drain; $P3_0$-$P3_3$: inputs; $P3_5$-$P3_7$: outputs; $P3_4$: DM
P01M	F8H	D7H	$P1_0$-$P1_7$: AD_0-AD_7; $P0_0$-$P0_7$: A_8-A_{15}; normal memory timing; internal stack
IRQ	FAH	00H	no interrupt requests
IMR	FBH	00H	no interrupts enabled
RP	FDH	00H	working register file 00H-0FH
SPL	FFH	65H	1st byte of stack is register 64H

instruction is executed to enable interrupt processing, and a jump instruction is executed to transfer control to the user's program at location 0812_H. The interrupt vectors for IRQ_0 through IRQ_5 are rerouted to locations 0800_H through $080F_H$, respectively, in three-byte increments, allowing enough room for a jump instruction to the appropriate interrupt service routine. That is, IRQ_0 is routed to location 0800_H, IRQ_1 to 0803_H, IRQ_2 to 0806_H, IRQ_3 to 0809_H, IRQ_4 to $080C_H$, and IRQ_5 to $080F_H$. Figure 1 illustrates the allocation of Z8 memory as defined by this application note.

The subroutines available to the user are referenced by a jump table beginning at location 001BH. Entry to a subroutine is made via the jump table. The 32 subroutines provided in the library are grouped into six functional classifications. These classifications are described below, each with a brief overview of the functions provided by each category. Table 2 defines one set of entry addresses for each subroutine in the library.

- Binary Arithmetic: Multiplication and division of unsigned 8- and 16-bit quantities.

- BCD Arithmetic: Addition and subtraction of variable-precision floating-point BCD values.

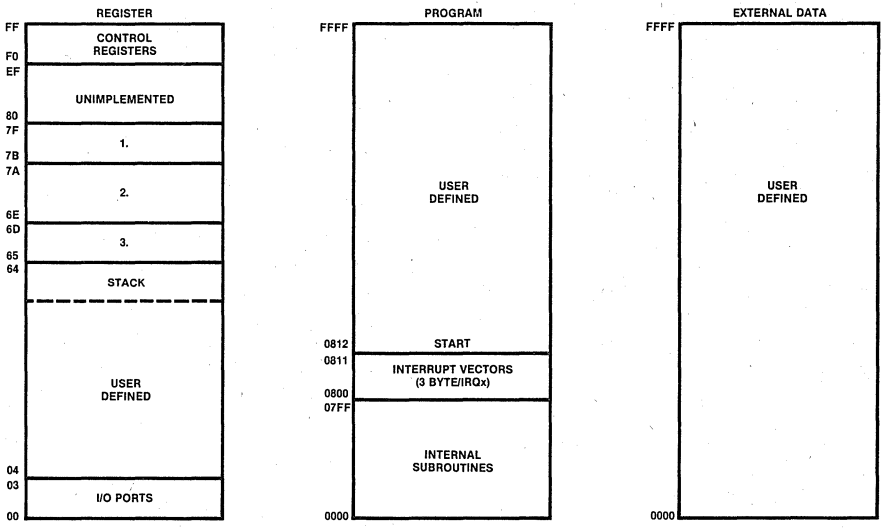
- Conversion Algorithms: BCD to and from decimal ASCII, binary to and from decimal ASCII, binary to and from hex ASCII.

- Bit Manipulations: Packs selected bits into the low-order bits of a byte, and optionally uses the result as an index into a jump table.

- Serial I/O: Inputs bytes under vectored interrupt control, outputs bytes under polled interrupt control. Options provided include:
  - odd or even parity
  - BREAK detection
  - echo
  - input editing (backspace, delete)
  - auto line feed

- Timer/Counter: Maintains a time-of-day clock with a variable number of ticks per second, generates an interrupt after a specified delay, generates variable width, variable frequency pulse output.

The listings in the "Canned Subroutine Library" provide a specification block prior to each subroutine, explain the subroutine's purpose, lists the input and output parameters, and gives pertinent notes concerning the subroutines. The following notes provide additional information on data formats and algorithms used by the subroutines.



REGISTERS USED BY SUBROUTINES:

1. USED BY MOST ROUTINES
2. USED BY SERIAL ROUTINES ONLY
3. USED BY TIMER/COUNTER ROUTINES ONLY

Figure 1. "ROMless Z8" Subroutine Library Memory Usage Map

1. Although the user is free to modify the conditions selected in the Port 3 Mode register (P3M, F7$_H$), P3M is a write-only register. This subroutine library maintains an image of P3M in its register P3M__save (7F$_H$). If software outside of the subroutine package is to modify P3M, it should reference and modify P3M__save prior to modification of P3M. For example, to select P32/P35 for handshake, the following instruction sequence could be used:

```
        OR      P3M_save, #04H
        LD      P3M, P3M_save
```

2. For many of the subroutines in this library, the location of the operands (source/destination) is flexible between register memory, external memory (code/data), and the serial channel (if enabled). The description of each parameter in the specification blocks tells what the location options are.

   - The location designation "in reg/ext memory" implies that the subroutine allows the operand to exist in register or in external data memory. The address of such an operand is contained in the designated register pair. If the high byte of that pair is 0, the operand is in register memory at the address held in the low byte of the register pair. Otherwise, the operand is in external data memory (accessed via LDE).

   - The location designation "in reg/ext/ser memory" implies the same considerations as above with one enhancement: if both bytes of the register pair are 0, the operand exists in the serial channel. In this case, the register pair is not modified (updated). For example, rather than storing a destination ASCII string in memory, it might be desirable to output the string to the serial line.

3. The BCD format supported by the following arithmetic and conversion routines allows representation of signed variable-precision BCD numbers. A BCD number of 2n digits is represented in n+1 consecutive bytes, where the byte at the lowest memory address (byte 0) represents the sign and post-decimal digit count, and the bytes in the n higher memory locations (bytes 1 through n) represent the magnitude of the BCD number. The address of byte 0 and the value n are passed to the subroutines in specified working registers. Digits are packed two per byte with the most-significant digit in the high-order nibble of byte 1 and the least-significant digit in the low-order nibble of byte n. Byte 0 is organized as two fields:

   Bit 7 represents sign:
   1 = negative;
   0 = positive.

   Bits 0-6 represent post-decimal digit count.

   For example:

   byte 0 = 05$_H$ = positive, with five post-decimal digits
   = 80$_H$ = negative, with no post-decimal digits
   = 90$_H$ = negative, with 16 post-decimal digits

4. The format of the decimal ASCII character string expected as input to the conversion routines "dascbcd" and "dascwrd" is defined as:

   (+ 1 -) (<digit>) [(<digit>)]

   in which
   ( ) Parentheses mean that the enclosed times or can be omitted.
   [ ] Brackets denote that the enclosed element is optional.

   Table 3 illustrates how various input strings are interpreted by the conversion routines.

5. The format of the decimal ASCII character string output from the conversion routine "bcddasc" operating on an input BCD string of 2n digits is

   1 sign of character (+ 1 -)
   2n-x pre-decimal digits
   1 decimal point if x does not equal 0
   x post-decimal digits

6. The format of the decimal ASCII character string output from the conversion routine "wrddassc" is

   1 sign character (determined by bit 15 of input word)
   6 pre-decimal digits
   no decimal point
   no post-decimal digits

Table 2. Subroutine Entry Points

Address	Name	Description
Binary Arithmetic Routines		
001B	divide	16/8 unsigned binary division
001E	div_16	16/16 unsigned binary division
0021	multiply	8x8 unsigned binary multiplication
0024	mult_16	16x16 unsigned binary multiplication
BCD Arithmetic Routines		
0027	bcdadd	BCD addition
002A	bcdsub	BCD subtraction
Conversion Routines		
002D	bcddasc	BCD to decimal ASCII
0030	dascbcd	Decimal ASCII to BCD
0033	bcdwrd	BCD to binary word
0036	wrdbcd	Binary word to BCD
0039	bythasc	Binary byte to hexadecimal ASCII
003C	wrdhasc	Binary word to hexadecimal ASCII
003F	hascwrd	Hexadecimal ASCII to binary word
0042	wrddasc	Binary word to decimal ASCII
0045	dascwrd	Decimal ASCII to binary word
Bit Manipulation Routines		
0048	clb	Collect bits in a byte
004B	tmj	Table jump under mask
Serial Routines		
004E	ser_init	Initialize serial I/O
0051	ser_input	IRQ$_3$ (receive) service
0054	ser_rlin	Read line
0057	ser_rabs	Read absolute
005A	ser_break	Transmit BREAK
005D	ser_flush	Flush (clear) input buffer
0060	ser_wlin	Write line
0063	ser_wabs	Write absolute
0066	ser_wbyt	Write byte
0069	ser_disable	Disable serial I/O
Timer/Counter Routines		
006C	tod_i	Initialize for time-of-day clock
006F	tod	Time-of-day IRQ service
0072	delay	Initialize for delay interval
0075	pulse_i	Initialize for pulse output
0078	pulse	Pulse IRQ service

7. Procedure name: ser_input

The conclusion of the algorithm for BREAK detection requires the Serial Receive Shift register to be cleared of the character currently being collected (if any). This requires a software wait loop of a one-character duration. The following explains the algorithm used (code lines 464 through 472, Part II):

$$1 \text{ character time} = \frac{(128\text{xPRE0xT0})}{\text{XTAL}} \frac{\text{sec}}{\text{bit}} \text{ x } 10 \frac{\text{bit}}{\text{char}}$$

$$= \frac{1280\text{xPRE0xT0}}{\text{XTAL}} \frac{\text{sec}}{\text{char}}$$

A software loop equal to one character time is needed:

$$1 \text{ character time} = \frac{2}{\text{XTAL}} \frac{\text{sec}}{\text{cycle}} \text{ x } n \frac{\text{cycle}}{\text{loop}}$$

$$= \frac{2n}{\text{XTAL}} \frac{\text{sec}}{\text{loop}}$$

Solve for n:

$$\frac{(1280 \text{ x PRE0 x T0})}{\text{XTAL}} = \frac{2n}{\text{XTAL}}$$

n = 640 x PRE0 x T0

The register pair SERhtime, SER1time was initialized during ser init to equal the product of the prescaler and the counter selected for the baud rate clock. That is,

SERhtime, SER1time = PRE0 x T0

The instruction sequence

```
inlop: ld    rSERtmpl, #53  (6 cycles)

lpl:   djnz  rSERtmpl, lpl  (12/10 cycles
                              taken/not taken)
```

executes in

6 + (52 x 12) + 10 cycles = 640 cycles

8. BREAK detection on the serial input line requires that the receive interrupt service routine be entered within a half-a-bit time, since the routine reads the input line to detect a true (=1) or false (=0) stop bit. Since the interrupt request is generated halfway through reception of the stop bit, half-a-bit time remains in which to read the stop bit level. Interrupt priorities and interrupt nesting should be established appropriately to ensure this requirement.

$$1/2 \text{ bit time} = \frac{(128 \text{ x PRE0 x T0})}{\text{XTAL x 2}} \text{ sec}$$

Table 3. Decimal ASCII Character String Interpretation

Input String	Result			Terminator
	Sign	Pre-Decimal Digits	Post-Decimal Digits	
+1234.567,	+	1234	567	,
+---+.789+	-		789	+
1234..	+	1234		.
4976-	+		4976	-

NOTE: The terminator can be any ASCII character that is not a valid ASCII string character.

```
Z8ASM     3.02
LOC     OBJ CODE    STMT SOURCE STATEMENT


PART_I  MODULE


!'ROMLESS Z8'   SUBROUTINE LIBRARY  PART I

 Initialize:    a) Port 0 & Port 1 set up to address
                    64K external memory;
                b) internal stack below allocated
                    RAM for subroutines;
                c) normal memory timing;
                d) IMR, IRQ, TMR, RP cleared;
                e) Port 2 inputs open-drain pull-ups;
                f) Data Memory select enabled;
                g) EI executed to 'unfreeze' IRQ;
                h) Jump to %0812.


Note:   The user is free to modify the initial
        conditions selected for a, b, and c above,
        via direct modification of the Port 0 & 1
        Mode register (P01M, %F8).

        The user is free to modify the conditions
        selected in the Port 3 Mode register (P3M, %F7).
        However, please note that P3M is a write-only
        register.  This subroutine library maintains
        an image of P3M in its register P3M_save (%7F).
        If software outside of the subroutine package
        is to modify P3M, it should reference and modify
        P3M_save, prior to modification of P3M.  For
        example, to select P32/P35 for handshake, use
        an instruction sequence such as:

                OR      P3M_save,#%04
                LD      P3M,P3M_save

        This is important if the serial and/or timer/
        counter subroutines are to be used, since these
        routines may modify P3M.
!
```

```
!Access to GLOBAL subroutines in this library should
be made via a CALL to the corresponding entry in the
jump table which begins at address %000F.  The jump
table should be referenced rather than a CALL to the
actual entry point of the subroutine to avoid future
conflict in the event such entry points change in
potential future revisions.

Each GLOBAL subroutine in this listing is headed by a
comment block specifying its PURPOSE and calling
sequence (INPUT and OUTPUT parameters).  For many of
the subroutines in this library, the location of the
operands (sources/destinations) is quite flexible
between register memory, external memory (code/data),
and the serial channel (if enabled).  The description
of each parameter specifies what the location choices
are:

       - The location designation 'in reg/ext memory'
implies that the subroutine allows that the operand
exist in either register or external data memory
The address of such an operand is contained
in the designated register pair.  If the high byte of
that pair is zero, the operand is in register memory
at the address given by the low byte of the register
pair.  Otherwise, the operand is in external data
memory (accessed via LDE).

       - The location designation
'in reg/ext/ser memory' implies the same
considerations as above with one enhancement: if both
bytes of the reg. pair are zero, the operand exists
in the serial channel.  In this case, the register
pair is not modified (updated).  For example, rather
than storing a destination ASCII string in memory, it
might be desirable to output such to the serial line.
!
```

```
CONSTANT
!Register Usage!

RAM_START       :=      %7F

P3M_save        :=      RAM_START
TEMP_3          :=      P3M_save-1
TEMP_2          :=      TEMP_3-1
TEMP_1          :=      TEMP_2-1
TEMP_4          :=      TEMP_1-1

!The following registers are modified/referenced
 by the Serial Routines ONLY.  They are
 available as general registers to the user
 who does not intend to make use of the
 Serial Routines!

SER_char        :=      TEMP_4-1
SER_tmp2        :=      SER_char-1
SER_tmp1        :=      SER_tmp2-1
SER_put         :=      SER_tmp1-1
SER_len         :=      SER_put-1
SER_buf         :=      SER_len-2
SER_imr         :=      SER_buf-1
SER_cfg         :=      SER_imr-1
!Serial Configuration Data
bit 7 : =1 => odd parity on
bit 6 : =1 => even parity on
 (bit 6,7 = 11 => undefined)
bit 5 : undefined
bit 4 : undefined
bit 3 : =1 => input editting on
bit 2 : =1 => auto line feed enabled
bit 1 : =1 => BREAK detection enabled
bit 0 : =1 => input echo on
!
op      :=      %80
ep      :=      %40
ie      :=      %08
al      :=      %04
be      :=      %02
ec      :=      %01
SER_get         :=      SER_cfg-1
SER_flg         :=      SER_get-1
!Serial Status Flags
bit 7 : =1 => serial I/O disabled
bit 6 : undefined
bit 5 : undefined
bit 4 : =1 => parity error
bit 3 : =1 => BREAK detected
bit 2 : =1 => input buffer overflow
bit 1 : =1 => input buffer not empty
bit 0 : =1 => input buffer full
!
sd      :=      %80
pe      :=      %10
bd      :=      %08
bo      :=      %04
bne     :=      %02
bf      :=      %01

RAM_TMR         :=      RAM_START-%10

SERltime        :=      SER_flg-1
```

```
SERhtime        :=      SERltime-1

!The following registers are modified/referenced
 by the Timer/Counter Routines ONLY.  They are
 available as general registers to the user
 who does not intend to make use of the
 Timer/Counter Routines!

TOD_tic         :=      RAM_TMR-2
TOD_imr         :=      TOD_tic-1
TOD_hr          :=      TOD_imr-1
TOD_min         :=      TOD_hr-1
TOD_sec         :=      TOD_min-1
TOD_tt          :=      TOD_sec-1
PLS_1           :=      TOD_tt-1
PLS_tmr         :=      PLS_1-1
PLS_2           :=      PLS_tmr-1

RAM_END         :=      PLS_2
STACK           :=      RAM_END

!Equivalent working register equates
 for above register layout!

!register file %70 - %7F!
RAM_STARTr      :=      %70     !for SRP!

rP3Msave        :=      R15
rTEMP_3         :=      R14
rTEMP_2         :=      R13
rTEMP_1         :=      R12
rrTEMP_1        :=      RR12
rTEMP_1h        :=      R12
rTEMP_1l        :=      R13
rTEMP_4         :=      R11
rSERchar        :=      R10
rSERtmp2        :=      R9
rSERtmp1        :=      R8
rrSERtmp        :=      RR8
rSERtmpl        :=      R9
rSERtmph        :=      R8
rSERput         :=      R7
rSERlen         :=      R6
rrSERbuf        :=      RR4
rSERbufh        :=      R4
rSERbufl        :=      R5
rSERimr         :=      R3
rSERcfg         :=      R2
rSERget         :=      R1
rSERflg         :=      R0


!register file %60 - %6F!
RAM_TMRr        :=      %60     !for SRP!
rTODtic         :=      R13
rTODimr         :=      R12
rTODhr          :=      R11
rTODmin         :=      R10
rTODsec         :=      R9
rTODtt          :=      R8
rPLS_1          :=      R7
rPLStmr         :=      R6
rPLS_2          :=      R5
```

```
                   210 EXTERNAL
                   211  ser_init        PROCEDURE
                   212  ser_input       PROCEDURE
                   213  ser_rlin        PROCEDURE
                   214  ser_rabs        PROCEDURE
                   215  ser_break       PROCEDURE
                   216  ser_flush       PROCEDURE
                   217  ser_wlin        PROCEDURE
                   218  ser_wabs        PROCEDURE
                   219  ser_wbyt        PROCEDURE
                   220  ser_disable     PROCEDURE
                   221  ser_get         PROCEDURE
                   222  ser_output      PROCEDURE
                   223  tod_i           PROCEDURE
                   224  tod_            PROCEDURE
                   225  delay           PROCEDURE
                   226  pulse_i         PROCEDURE
                   227  pulse_          PROCEDURE
                   228
                   229
                   230         $SECTION PROGRAM
                   231 GLOBAL
                   232
                   233
                   234 !Interrupt vectors!
P 0000 0800        235 IRQ_0    ARRAY    [1 word]   :=    [%0800]
P 0002 0803        236 IRQ_1    ARRAY    [1 word]   :=    [%0803]
P 0004 0806        237 IRQ_2    ARRAY    [1 word]   :=    [%0806]
P 0006 0809        238 IRQ_3    ARRAY    [1 word]   :=    [%0809]
P 0008 080C        239 IRQ_4    ARRAY    [1 word]   :=    [%080C]
P 000A 080F        240 IRQ_5    ARRAY    [1 word]   :=    [%080F]
                   241
                   242
```

```
                          GLOBAL

                          !Jump Table!
P 000C                    ENTER    PROCEDURE
                          ENTRY
P 000C 8D  007B'                   JP       INIT
P 000F                    END      ENTER


P 000F 28  43  29         copyright ARRAY [* BYTE] := '(C)1980ZILOG'
P 0012 31  39  38
P 0015 30  5A  49
P 0018 4C  4F  47

                          !Subroutine Entry Points!
P 001B                    JUMP     PROCEDURE
                          ENTRY

                          !Binary Arithmetic Routines!

P 001B 8D  0099'                   JP       divide          !16/8 unsigned binary
                                                             division!
P 001E 8D  00B7'                   JP       div_16          !16/16 unsigned binary
                                                             division!
P 0021 8D  00E2'                   JP       multiply        !8x8 unsigned binary
                                                             multiplication!
P 0024 8D  00F6'                   JP       mult_16         !16x16 unsigned binary
                                                             multiplication!

                          !BCD Arithmetic Routines!

P 0027 8D  011A'                   JP       bcdadd          !BCD addition!

P 002A 8D  0117'                   JP       bcdsub          !BCD subtraction!

                          !Conversion Routines!

P 002D 8D  0205'                   JP       bcddasc         !BCD to decimal ASCII!

P 0030 8D  0363'                   JP       dascbcd         !Decimal ASCII to BCD!

P 0033 8D  0284'                   JP       bcdwrd          !BCD to binary word!

P 0036 8D  02CD'                   JP       wrdbcd          !binary word to BCD!

P 0039 8D  025C'                   JP       bythasc         !Bin. byte to Hex ASCII!

P 003C 8D  0257'                   JP       wrdhasc         !Bin. word to hex ASCII!

P 003F 8D  0319'                   JP       hascwrd         !Hex ASCII to bin word!

P 0042 8D  03BE'                   JP       wrddasc         !Bin. word to dec ASCII!

P 0045 8D  034D'                   JP       dascwrd         !dec ASCII to bin word!

                          !Bit Manipulation Routines!

P 0048 8D  04A1'                   JP       clb             !collect bits in a byte!

P 004B 8D  04B9'                   JP       tjm             !Table Jump Under Mask!

                          !Serial Routines!

P 004E 8D  0000*                   JP       ser_init        !initialize serial I/O!
```

```

P 0051 8D   0000*                   JP      ser_input       !IRQ3 (receive) service!

P 0054 8D   0000*                   JP      ser_rlin        !read line!

P 0057 8D   0000*                   JP      ser_rabs        !read absolute!

P 005A 8D   0000*                   JP      ser_break       !transmit BREAK!

P 005D 8D   0000*                   JP      ser_flush       !flush (clear)
                                                             input buffer!
P 0060 8D   0000*                   JP      ser_wlin        !write line!

P 0063 8D   0000*                   JP      ser_wabs        !write absolute!

P 0066 8D   0000*                   JP      ser_wbyt        !write byte!

P 0069 8D   0000*                   JP      ser_disable     !disable serial I/O!

                        !Timer/Counter Routines!

P 006C 8D   0000*                   JP      tod_i           !init for time of day!

P 006F 8D   0000*                   JP      tod             !tod IRQ service!

P 0072 8D   0000*                   JP      delay           !init for delay interval

P 0075 8D   0000*                   JP      pulse_i         !init for pulse output!

P 0078 8D   0000*                   JP      pulse           !pulse IRQ service!

P 007B                  END         JUMP

                        !Initialization!
P 007B                  INIT        PROCEDURE
                        ENTRY

P 007B E6   F8   D7                 LD      P01M,#%(2)11010111
                                                            !internal stack;
                                                             AD0-A15;
                                                             normal memory
                                                             timing !
P 007E E6   7F   10                 LD      P3M_save,#%(2)00010000
                                                            !P3M is write-only,
                                                             so keep a copy in
                                                             RAM for later
                                                             reference !
P 0081 E4   7F   F7                 LD      P3M,P3M_save    !set up Port 3 !
P 0084 E6   FF   65                 LD      SPL,#STACK      !stack pointer !
P 0087 B0   F1                      CLR     TMR             !reset timers!
P 0089 E6   F6   FF                 LD      P2M,#%FF        !all inputs!
P 008C B0   FA                      CLR     IRQ             !reset int. requests!
P 008E B0   FB                      CLR     IMR             !disable interrupts !
P 0090 B0   FD                      CLR     RP              !register pointer!
P 0092 E6   70   80                 LD      SER_flg,#%80    !serial disabled!
P 0095 9F                           EI                      !globally enable
                                                             interrupts !
P 0096 8D   0812                    JP      %0812

P 0099                  END         INIT
```

```
                   CONSTANT
                    div_LEN        :=      R10
                    DIVISOR        :=      R11
                    dividend_HI    :=      R12
                    dividend_LO    :=      R13
                   GLOBAL
P 0099             divide  PROCEDURE
                   !*****************************************************
                    Purpose =       To perform a 16-bit by 8-bit unsigned
                                    binary division.

                    Input =         R11 = 8-bit divisor
                                    RR12 = 16-bit dividend

                    Output =        R13  = 8-bit quotient
                                    R12  = 8-bit remainder
                                    Carry flag = 1 if overflow
                                               = 0 if no overflow
                                    R11 unmodified
                   *****************************************************!
                   ENTRY
P 0099 A9  7C              ld       TEMP_1,div_LEN   !save caller's R10!
P 009B AC  08              ld       div_LEN,#8       !LOOP COUNTER!

                   !CHECK IF RESULT WILL FIT IN 8 BITS!
P 009D A2  BC              cp       DIVISOR,dividend_HI
P 009F BB  02              jr       UGT,LOOP         !CARRY = 0 (FOR RLC)!
                   !overflow!
P 00A1 DF                  SCF                       !CARRY = 1!
P 00A2 AF                  ret

P 00A3 10  ED      LOOP:   RLC      dividend_LO      !DIVIDEND * 2!
P 00A5 10  EC              RLC      dividend_HI
P 00A7 7B  04              jr       c,subt
P 00A9 A2  BC              cp       DIVISOR,dividend_HI
P 00AB BB  03              jr       UGT,next         !CARRY = 0!
P 00AD 22  CB      subt:   SUB      dividend_HI,DIVISOR
P 00AF DF                  SCF                   !TO BE SHIFTED INTO RESULT!
P 00B0 AA  F1      next:   djnz     div_LEN,LOOP     !no flags affected!

                   !ALL    DONE!
P 00B2 10  ED              RLC      dividend_LO
                                                     !CARRY = 0: no overflow!
P 00B4 A8  7C              ld       div_LEN,TEMP_1   !restore caller's R10!
P 00B6 AF                  ret
P 00B7             END divide
```

```
                 CONSTANT
                  d16_LEN         :=      R7
                  dvsr_hi         :=      R8
                  dvsr_lo         :=      R9
                  rem_hi          :=      R10
                  rem_lo          :=      R11
                  quot_hi         :=      R12
                  quot_lo         :=      R13
                 GLOBAL
P 00B7           div_16  PROCEDURE
                 !*****************************************************
                  Purpose =       To perform a 16-bit by 16-bit unsigned
                                  binary division.

                  Input =         RR8 = 16-bit divisor
                                  RR12 = 16-bit dividend

                  Output =        RR12  = 16-bit quotient
                                  RR10  = 16-bit remainder
                                  RR8 unmodified
                 *****************************************************!
                 ENTRY
P 00B7 79  7C              ld     TEMP_1,d16_LEN  !save caller's R10!
P 00B9 7C  10              ld     d16_LEN,#16     !LOOP COUNTER!
P 00BB CF                  rcf                    !carry = 0!
P 00BC B0  EA              clr    rem_hi
P 00BE B0  EB              clr    rem_lo
P 00C0 10  ED    dlp_16:   rlc    quot_lo
P 00C2 10  EC              rlc    quot_hi
P 00C4 10  EB              rlc    rem_lo
P 00C6 10  EA              rlc    rem_hi
P 00C8 7B  0A              jr     c,subt_16
P 00CA A2  8A              cp     dvsr_hi,rem_hi
P 00CC BB  0B              jr     ugt,skp_16
P 00CE 7B  04              jr     ult,subt_16
P 00D0 A2  9B              cp     dvsr_lo,rem_lo
P 00D2 BB  05              jr     ugt,skp_16
P 00D4 22  B9    subt_16:  sub    rem_lo,dvsr_lo
P 00D6 32  A8              sbc    rem_hi,dvsr_hi
P 00D8 DF                  scf
P 00D9 7A  E5    skp_16:   djnz   d16_LEN,dlp_16  !no flags affected!
P 00DB 10  ED              rlc    quot_lo
P 00DD 10  EC              rlc    quot_hi
P 00DF 78  7C              ld     d16_LEN,TEMP_1
P 00E1 AF                  ret
P 00E2           END div_16

                 CONSTANT
                  MULTIPLIER      :=      R11
                  PRODUCT_LO      :=      R13
                  PRODUCT_HI      :=      R12
                  mul_LEN         :=      R10
                 GLOBAL
P 00E2           multiply         PROCEDURE
                 !*****************************************************
                  Purpose =       To perform an 8-bit by 8-bit unsigned
                                  binary multiplication.

                  Input =         R11 = multiplier
                                  R13 = multiplicand

                  Output =        RR12 = product
                                  R11 unmodified
                 *****************************************************!
                 ENTRY
P 00E2 A9  7C              ld     TEMP_1,mul_LEN  !save caller's R10!
P 00E4 AC  09              ld     mul_LEN,#9      !8 BITS!
P 00E6 B0  EC              clr    PRODUCT_HI      !INIT HIGH RESULT BYTE!
P 00E8 CF                  RCF                    !CARRY = 0!
P 00E9 C0  EC    LOOP1:    RRC    PRODUCT_HI
P 00EB C0  ED              RRC    PRODUCT_LO
P 00ED FB  02              jr     NC,NEXT
P 00EF 02  CB              ADD    PRODUCT_HI,MULTIPLIER
P 00F1 AA  F6    NEXT:     djnz   mul_LEN,LOOP1
P 00F3 A8  7C              ld     mul_LEN,TEMP_1  !restore caller's R10!
P 00F5 AF                  ret
P 00F6           END       multiply
```

```
                    CONSTANT
                     m16_LEN        :=        R7
                     plier_hi       :=        R8
                     plier_lo       :=        R9
                     prod_hi        :=        R10
                     prod_lo        :=        R11
                     mult_hi        :=        R12
                     mult_lo        :=        R13
                    GLOBAL
P 00F6              mult_16 PROCEDURE
                    !*****************************************************
                     Purpose =      To perform an 16-bit by 16-bit unsigned
                                    binary multiplication.

                     Input =        RR8 = multiplier
                                    RR12 = multiplicand

                     Output =       RQ10 = product (R10, R11, R12, R13)
                                    RR8 unmodified
                                    Zero FLAG = 0 if result > 16 bits
                                              = 1 if result fits in 16
                                      (unsigned) bits (RR12 = result)
                    *****************************************************!
                    ENTRY
P 00F6 79  7C               ld      TEMP_1,m16_LEN  !save caller's R7!
P 00F8 7C  11               ld      m16_LEN,#17     !16 BITS!
P 00FA B0  EA               clr     prod_hi
P 00FC B0  EB               clr     prod_lo         !init product!
P 00FE CF                   rcf                     !CARRY = 0!
P 00FF C0  EA       loop16: rrc     prod_hi
P 0101 C0  EB               rrc     prod_lo         !bit 0 to carry!
P 0103 C0  EC               rrc     mult_hi         !multiplicand / 2!
P 0105 C0  ED               rrc     mult_lo
P 0107 FB  04               jr      nc,next16
P 0109 02  B9               add     prod_lo,plier_lo
P 010B 12  A8               adc     prod_hi,plier_hi
P 010D 7A  F0       next16: djnz    m16_LEN,loop16  !next bit!
P 010F 78  7C               ld      m16_LEN,TEMP_1  !restore caller's R7!
P 0111 A9  7C               ld      TEMP_1,prod_hi  !test product...!
P 0113 44  EB  7C           or      TEMP_1,prod_lo  !...bits 31 - 16!
P 0116 AF                   ret
P 0117              END     mult_16
```

```
                     !The BCD format supported by the following arithmetic
                      and conversion routines allows representation
                      of signed magnitude variable precision BCD
                      numbers.  A BCD number of 2n digits is
                      represented in n+1 consecutive bytes where
                      the byte at the lowest memory address
                      ('byte 0') represents the sign and post-
                      decimal digit count, and the bytes in the
                      next n higher memory locations ('byte 1'
                      through 'byte n') represent the magnitude
                      of the BCD number.  The address of 'byte 0'
                      and the value n are passed to the subroutines
                      in specified working registers.  Digits are
                      packed two per byte with the most
                      significant digit in the high order nibble
                      of 'byte 1' and the least significant digit
                      in the low order nibble of 'byte n'.  'Byte 0'
                      is organized as two fields:
                             bit 7 represents sign:
                                  = 1 => negative
                                  = 0 => positive
                             bit 6-0 represent post-decimal digit
                                     count
                     For example:
                     'byte 0'= %05 => positive, with 5 post-decimal digits
                             = %80 => negative, with no post-decimal digits
                             = %90 => negative, with 16 post-decimal digits
                     !

                     CONSTANT
                      bcd_LEN := R12
                      bcd_SRC := R14
                      bcd_DST := R15
                     GLOBAL
P 0117               bcdsub  PROCEDURE
                     !######################################################
                      Purpose =       To subtract two packed BCD strings of
                                      equal length.
                                      dst  <-- dst - src

                      Input =         R15 = address of destination BCD
                                            string (in register memory).
                                      R14 = address of source BCD
                                            string (in register memory).
                                      R12 = BCD digit count / 2

                      Output =        Destination BCD string contains the
                                      difference.
                                      Source BCD string may be modified.
                                      R12, R14, R15 unmodified if no error
                                      R13 modified.
                                      Carry FLAG = 1 if underflow or format
                                                          error.
                     ######################################################!
                     ENTRY
P 0117 B7  EE  80            xor     @bcd_SRC,#%80    !complement sign of
                                                       subtrahend!
                     !fall into bcdadd!
P 011A               END     bcdsub
```

```
                        GLOBAL
P 011A                  bcdadd  PROCEDURE
                        !*****************************************************
                         Purpose =      To add two packed BCD strings of
                                        equal length.
                                        dst  <-- dst + src

                         Input =        R15 = address of destination BCD
                                              string (in register memory).
                                        R14 = address of source BCD
                                              string (in register memory).
                                        R12 = BCD digit count / 2

                         Output =       Destination BCD string contains the sum.
                                        Source BCD string may be modified.
                                        R12, R14, R15 unmodified if no error
                                        R13 modified.
                                        Carry FLAG = 1 if overflow or format
                                                          error.
                        *****************************************************!
                        ENTRY
                        !delete all leading pre-decimal zeroes!
P 011A E6  7E  02               ld      TEMP_3,#2
P 011D D8  EE                   ld      R13,bcd_SRC
P 011F C9  7B           ba_3:   ld      TEMP_4,bcd_LEN
P 0121 04  7B  7B               add     TEMP_4,TEMP_4   !total digit count!
P 0124 E5  ED  7D               ld      TEMP_2,@R13     !get sign/post dec #!
P 0127 56  7D  7F               and     TEMP_2,#%7F     !isolate post dec #!
P 012A 24  7D  7B               sub     TEMP_4,TEMP_2   !pre-dec digit cnt!
P 012D 7D  0203'                jp      ult,ba_err      !format error!
P 0130 6B  1A                   jr      z,ba_1          !no pre-dec. digits!
P 0132 70  EC           ba_2:   push    R12             !save!
P 0134 C7  CD  01               ld      R12,1(R13)      !leading byte!
P 0137 76  EC  F0               tm      R12,#%F0        !test leading digit!
P 013A 50  EC                   pop     R12             !restore!
P 013C EB  0E                   jr      nz,ba_1         !no more leading 0's!
P 013E B0  7C                   clr     TEMP_1
P 0140 D6  0463'                call    rdl             !rotate left!
P 0143 21  ED                   inc     @R13            !update post dec #!
P 0145 4D  0203'                jp      ov,ba_err       !oops!
P 0148 00  7B                   dec     TEMP_4          !dec pre-dec #!
P 014A EB  E6                   jr      nz,ba_2         !loop!
P 014C D8  EF           ba_1:   ld      R13,bcd_DST
P 014E 00  7E                   dec     TEMP_3          !SRC and DST done?!
P 0150 EB  CD                   jr      nz,ba_3         !do DST!
                        !leading zero deletion complete!
                        !insure DST is > or = SRC; exchange if necessary!
P 0152 E3  DF                   ld      R13,@bcd_DST
P 0154 56  ED  7F               and     R13,#%7F        !isolate post dec #!
P 0157 E5  EE  7D               ld      TEMP_2,@bcd_SRC
P 015A 56  7D  7F               and     TEMP_2,#%7F     !isolate post dec #!
P 015D A4  7D  ED               cp      R13,TEMP_2
P 0160 70  ED                   push    R13             !save!
P 0162 7B  39                   jr      ult,ba_4        !DST > SRC!
P 0164 BB  18                   jr      ugt,ba_5        !DST < SRC!
                        !decimal points in same position.
                         must compare magnitude!
P 0166 D8  EC                   ld      R13,bcd_LEN
P 0168 E9  7C                   ld      TEMP_1,bcd_SRC
P 016A F9  7B                   ld      TEMP_4,bcd_DST
P 016C 20  7C           ba_6:   inc     TEMP_1
P 016E 20  7B                   inc     TEMP_4
P 0170 E5  7C  7E               ld      TEMP_3,@TEMP_1  !get SRC byte!
P 0173 A5  7B  7E               cp      TEMP_3,@TEMP_4  !compare DST byte!
```

```
P 0176 BB  06                jr      ugt,ba_5         !SRC > DST!
P 0178 7B  23                jr      ult,ba_4         !SRC < DST!
P 017A DA  F0                djnz    R13,ba_6         !loop!
P 017C 8B  1F                jr      ba_4             !DST > or = SRC!
               !swap source and destination operands!
P 017E D8  EC        ba_5:   ld      R13,bcd_LEN
P 0180 DE                    inc     R13              !include flag/size byte!
P 0181 02  ED                add     bcd_SRC,R13
P 0183 02  FD                add     bcd_DST,R13
P 0185 00  EE        ba_7:   dec     bcd_SRC
P 0187 00  EF                dec     bcd_DST
P 0189 E5  EE  7C            ld      TEMP_1,@bcd_SRC
P 018C E5  EF  7B            ld      TEMP_4,@bcd_DST
P 018F F5  7B  EE            ld      @bcd_SRC,TEMP_4
P 0192 F5  7C  EF            ld      @bcd_DST,TEMP_1 !one byte swapped!
P 0195 DA  EE                djnz    R13,ba_7
P 0197 D8  7D                ld      R13,TEMP_2
P 0199 50  7D                pop     TEMP_2
P 019B 70  ED                push    R13
               !exchange complete!
P 019D 50  ED        ba_4:   pop     R13              !restore!
               !R13 = DST post decimal digit count
                TEMP_2 = SRC post decimal digit count
                R13 =< TEMP_2                                 !
P 019F 24  ED  7D            sub     TEMP_2,R13
P 01A2 C0  7D                rrc     TEMP_2           !alignment offset!
P 01A4 FB  09                jr      nc,ba_8          !digits word aligned!
               !rotate out least significant SRC post decimal digit!
P 01A6 D8  EE                ld      R13,bcd_SRC
P 01A8 01  ED                dec     @R13             !dec post dec digit #!
P 01AA B0  7C                clr     TEMP_1
P 01AC D6  0485'             call    rdr
               !determine if addition or subtraction!
P 01AF E5  EE  7B    ba_8:   ld      TEMP_4,@bcd_SRC !sign of SRC!
P 01B2 B5  EF  7B            xor     TEMP_4,@bcd_DST !sign of DST!
               !get starting addresses!
P 01B5 D8  EC                ld      R13,bcd_LEN
P 01B7 24  7D  ED            sub     R13,TEMP_2
P 01BA 6B  45                jr      z,ba_14          !done already!
P 01BC 02  ED                add     bcd_SRC,R13
P 01BE 02  FC                add     bcd_DST,bcd_LEN
               !ready!!!
P 01C0 CF                    rcf                      !carry = 0!
P 01C1 E5  EF  7C    ba_11:  ld      TEMP_1,@bcd_DST
P 01C4 76  7B  80            tm      TEMP_4,#%80      !add or sub?!
P 01C7 6B  05                jr      z,ba_9           !add!
P 01C9 35  EE  7C            sbc     TEMP_1,@bcd_SRC
P 01CC 8B  03                jr      ba_10
P 01CE 15  EE  7C    ba_9:   adc     TEMP_1,@bcd_SRC
P 01D1 40  7C        ba_10:  da      TEMP_1
P 01D3 F5  7C  EF            ld      @bcd_DST,TEMP_1
P 01D6 00  EF                dec     bcd_DST
P 01D8 00  EE                dec     bcd_SRC
P 01DA DA  E5                djnz    R13,ba_11
               !propagate carry thru TEMP_2 bytes of DST!
P 01DC D8  7D                ld      R13,TEMP_2
P 01DE DE                    inc     R13              !may be zero!
P 01DF DA  02                djnz    R13,ba_12
P 01E1 8B  09                jr      ba_13
P 01E3 17  EF  00    ba_12:  adc     @bcd_DST,#0
P 01E6 41  EF                da      @bcd_DST
P 01E8 00  EF                dec     bcd_DST
P 01EA DA  F7                djnz    R13,ba_12
```

```
                        780 !carry propagate complete!
P 01EC FB   13          781 ba_13:  jr      nc,ba_14        !done!
                        782 !Rotate out least significant post decimal DST
                        783  digit to make room for carry at high end!
P 01EE E5   EF  7C      784         ld      TEMP_1,@bcd_DST
P 01F1 56   7C  7F      785         and     TEMP_1,#%7F
P 01F4 6D   0203'       786         jp      z,ba_err        !no post dec digits!
P 01F7 E6   7C  10      787         ld      TEMP_1,#%10
P 01FA D8   EF          788         ld      R13,bcd_DST
P 01FC D6   0485'       789         call    rdr
P 01FF 01   EF          790         dec     @bcd_DST        !dec digit cnt!
P 0201 CF               791 ba_14:  rcf
P 0202 AF               792         ret
                        793
P 0203 DF               794 ba_err: scf
P 0204 AF               795         ret
P 0205                  796 END     bcdadd
```

```
                        CONSTANT
                         bca_LEN          :=       R12
                         bca_SRC          :=       R13
                        GLOBAL
P 0205                  bcddasc PROCEDURE
                        !################################################
                         Purpose =        To convert a variable length BCD
                                          string to decimal ASCII.

                         Input =          RR14 = address of destination ASCII
                                                 string (in reg/ext/ser memory).
                                          R13 = address of source BCD
                                                 string (in register memory).
                                          R12 = BCD digit count / 2

                         Output =         ASCII string in designated
                                          destination buffer.
                                          Carry FLAG = 1 if input format error
                                                       or serial disabled,
                                                     = 0 if no error.
                                          R12, R13, R14, R15 modified.
                                          Input BCD string ummodified.
                        ################################################!
                        ENTRY
P 0205 E6  7C  2D                 ld      TEMP_1,#'-'      !minus sign!
P 0208 77  ED  80                 tm      @bca_SRC,#%80    !src negative?!
P 020B EB  03                     jr      nz,bcd_d1        !yes!
P 020D E6  7C  2B                 ld      TEMP_1,#'+'      !positive sign!
P 0210 E5  ED  7E       bcd_d1:   ld      TEMP_3,@bca_SRC
P 0213 56  7E  7F                 and     TEMP_3,#%7F      !isolate post dec cnt!
P 0216 02  CC                     add     bca_LEN,bca_LEN  !total digit count!
P 0218 70  EC                     push    bca_LEN
P 021A 24  7E  EC                 sub     bca_LEN,TEMP_3   !pre-dec digit cnt!
P 021D 50  7E                     pop     TEMP_3           !total digit count!
P 021F 7B  35                     jr      ult,bcd_d2       !format error!
P 0221 D6  03F4'                  call    put_dest         !sign to dest.!
P 0224 7B  30                     jr      c,bcd_d2         !serial error!
P 0226 A6  EC  00                 cp      bca_LEN,#0       !any pre-dec digits?!
P 0229 6B  22                     jr      z,bcd_d6         !no. start with '.'!
P 022B 76  7E  01       bcd_d4:   tm      TEMP_3,#1        !need next byte?!
P 022E EB  04                     jr      nz,bcd_d3        !not yet.!
P 0230 DE                         inc     bca_SRC          !update pointer!
P 0231 E5  ED  7D                 ld      TEMP_2,@bca_SRC  !get next byte!
P 0234 F0  7D           bcd_d3:   swap    TEMP_2
P 0236 E4  7D  7C                 ld      TEMP_1,TEMP_2
P 0239 56  7C  0F                 and     TEMP_1,#%0F      !isolate digit!
P 023C A6  7C  09                 cp      TEMP_1,#9        !verify bcd!
P 023F BB  14                     jr      ugt,bcd_d5       !no good!
P 0241 06  7C  30                 add     TEMP_1,#%30      !convert to ASCII!
P 0244 D6  03F4'                  call    put_dest         !to destination!
P 0247 00  7E                     dec     TEMP_3           !digit count!
P 0249 6B  0B                     jr      z,bcd_d2         !all done!
P 024B CA  DE                     djnz    bca_LEN,bcd_d4   !next digit!
P 024D E6  7C  2E       bcd_d6:   ld      TEMP_1,#'.'      !time for dec. pt.!
P 0250 D6  03F4'                  call    put_dest         !to destination!
P 0253 8B  D6                     jr      bcd_d4           !continue!
P 0255 DF               bcd_d5:   scf                      !set error return!
P 0256 AF               bcd_d2:   ret
P 0257                  END       bcddasc

                        GLOBAL
P 0257                  wrdhasc PROCEDURE
                        !################################################
                         Purpose =        To convert a binary word to Hex ASCII.

                         Input =          RR12 = source binary word.
                                          RR14 = address of destination ASCII
                                                 string (in reg/ext/ser memory).

                         Note =           All other details same as for bythasc.
                        ################################################!
                        ENTRY
P 0257 D6  025C'                  call    bythasc          !convert R12!
P 025A C8  ED                     ld      R12,R13
                        !fall into bythasc!
P 025C                  END       wrdhasc
```

```
                      CONSTANT
                       bna_SRC          :=      R12
                      GLOBAL
P 025C                bythasc PROCEDURE
                      !***************************************************
                       Purpose =        To convert a binary byte to Hex ASCII.

                       Input =          RR14 = address of destination ASCII
                                              string (in reg/ext/ser memory).
                                        R12 = Source binary byte.

                       Output =         ASCII string in designated
                                        destination buffer.
                                        Carry = 1 if error (serial only).
                                        R14, R15 modified.
                      ***************************************************!
                      ENTRY
P 025C B0  7E                 clr       MODE    !flag => binary to ASCII!
P 025E E6  7D  02     bca_go: ld        TEMP_2,#2
P 0261 F0  EC         bca_go1: SWAP     bna_SRC           !look at next nibble!
P 0263 C9  7C                 ld        TEMP_1,bna_SRC
P 0265 56  7C  0F             and       TEMP_1,#%0F       !isolate low nibble!
P 0268 06  7C  30             ADD       TEMP_1,#%30       !convert to ASCII!
P 026B A6  7C  3A             cp        TEMP_1,#%3A       !>9?!
P 026E 7B  09                 jr        ult,skip          !no!
P 0270 DF                     SCF                         !in case error!
P 0271 76  7E  01             TM        MODE,#1           !input is BCD?!
P 0274 EB  0D                 JR        NZ,bca_ex         !yes. error.!
P 0276 06  7C  07             ADD       TEMP_1,#%07       !input hex. adjust!
P 0279 D6  03F4'      skip:   call      put_dest          !put byte in dest!
P 027C 7B  05                 jr        c,bca_ex          !error!
P 027E 00  7D                 dec       TEMP_2
P 0280 EB  DF                 jr        nz,bca_go1        !loop till done!
P 0282 CF                     RCF                         !carry = 0: no error!
P 0283 AF             bca_ex: ret                         !done!
P 0284                END     bythasc
```

```
                     CONSTANT
                      bcd_adr          :=       R14
                      bcd_cnt          :=       R15
                     GLOBAL
P 0284               bcdwrd  PROCEDURE
                     !********************************************************
                      Purpose =        To convert a variable length BCD
                                       string to a signed binary word.  Only
                                       pre-decimal digits are converted.

                      Input =          R14 = address of source BCD
                                             string (in register memory).
                                       R15 = BCD digit count / 2

                      Output =         RR12 = binary word
                                       Carry FLAG = 1 if input format error
                                                    or dest overflow,
                                                  = 0 if no error.
                                       R14,R15 modified.
                     ********************************************************!
                     ENTRY
P 0284 B0  EC                  clr     R12              !init destination!
P 0286 B0  ED                  clr     R13
P 0288 E5  EE  7B              ld      TEMP_4,@bcd_adr  !get sign/post_length!
P 028B 56  7B  7F              and     TEMP_4,#%7F      !isolate post_length!
P 028E 02  FF                  add     bcd_cnt,bcd_cnt  !# bcd digits!
P 0290 24  7B  EF              sub     bcd_cnt,TEMP_4   !# pre-dec digits!
P 0293 7B  37                  jr      ult,bcd_w2       !format error!
P 0295 E5  EE  7B              ld      TEMP_4,@bcd_adr  !remember sign!
P 0298 E6  7E  02    bcd_w3:   ld      TEMP_3,#2        !digits per byte!
P 029B EE                      inc     bcd_adr          !src address!
P 029C E5  EE  7D              ld      TEMP_2,@bcd_adr  !get next src byte!
P 029F A6  EF  00    bcd_w1:   cp      bcd_cnt,#0       !digit count = 0?!
P 02A2 6B  12                  jr      z,bcd_w4         !conversion complete!
P 02A4 F0  7D                  swap    TEMP_2           !next digit!
P 02A6 E4  7D  7C              ld      TEMP_1,TEMP_2
P 02A9 D6  042C'               call    bcd_bin          !accumulate in binary!
P 02AC 7B  1E                  jr      c,bcd_w2         !overflow or format err!
P 02AE 00  EF                  dec     bcd_cnt          !update digit count!
P 02B0 00  7E                  dec     TEMP_3           !next byte?!
P 02B2 EB  EB                  jr      nz,bcd_w1        !no. same.!
P 02B4 8B  E2                  jr      bcd_w3           !next byte!
P 02B6 DF            bcd_w4:   scf                      !in case!
P 02B7 76  EC  80              tm      R12,#%80         !result > 15 bits?!
P 02BA EB  10                  jr      nz,bcd_w2        !overflow!
P 02BC 76  7B  80    bcd_w5:   tm      TEMP_4,#%80      !source negative?!
P 02BF 6B  0A                  jr      z,bcd_w6         !no. done.!
P 02C1 60  EC                  com     R12
P 02C3 60  ED                  com     R13
P 02C5 06  ED  01              add     R13,#1
P 02C8 16  EC  00              adc     R12,#0           !RR12 two's complement!
P 02CB CF            bcd_w6:   rcf                      !carry = 0!
P 02CC AF            bcd_w2:   ret
P 02CD               END       bcdwrd
```

```
                     GLOBAL
P 02CD               wrdbcd  PROCEDURE
                     !*****************************************************
                      Purpose =      To convert a signed binary word
                                     to a variable length BCD string.

                      Input =        R14 = address of destination BCD
                                           string (in register memory)
                                     RR12 = source binary word
                                     R15 = BCD digit count / 2

                      Output =       BCD string in destination buffer
                                     Carry FLAG = 1 if dest overflow
                                                = 0 if no error.
                                     R12,R13,R14,R15 modified.
                     *****************************************************!
                     ENTRY
P 02CD B1  EE                clr     @bcd_adr         !init sign/post_dec_cnt!
P 02CF 76  EC  80            tm      R12,#%80         !is input word negative?
P 02D2 6B  0D                jr      z,wrd_b0
P 02D4 47  EE  80            or      @bcd_adr,#%80    !set result negative!
P 02D7 60  ED                com     R13
P 02D9 60  EC                com     R12
P 02DB 06  ED  01            add     R13,#1
P 02DE 16  EC  00            adc     R12,#0           !RR12 two's complement!
P 02E1 10  ED        wrd_b0: rlc     R13
P 02E3 10  EC                rlc     R12              !bit 15 not magnitude!
P 02E5 EE                    inc     bcd_adr          !update dest pointer!
P 02E6 E9  7C                ld      TEMP_1,bcd_adr
P 02E8 F9  7D                ld      TEMP_2,bcd_cnt   !dest byte count!
P 02EA 04  EF  7C            add     TEMP_1,bcd_cnt
P 02ED 00  7C                dec     TEMP_1           != bcd end addr!
P 02EF B1  EE        wrd_b1: clr     @bcd_adr         !initialize dest!
P 02F1 EE                    inc     bcd_adr
P 02F2 FA  FB                djnz    bcd_cnt,wrd_b1
P 02F4 E6  7E  0F            ld      TEMP_3,#15       !source bit count!
P 02F7 70  7E        wrd_b3: push    TEMP_3
P 02F9 10  ED                rlc     R13
P 02FB 10  EC                rlc     R12              !bit 15 to carry!
P 02FD E8  7C                ld      bcd_adr,TEMP_1   !start at end!
P 02FF F8  7D                ld      bcd_cnt,TEMP_2   !dest byte count!
                     !(dest bcd string) <-- (dest_bcd string * 2) + carry!
P 0301 E5  EE  7E    wrd_b2: ld      TEMP_3,@bcd_adr
P 0304 15  EE  7E            adc     TEMP_3,@bcd_adr  !* 2 + carry!
P 0307 40  7E                da      TEMP_3
P 0309 F5  7E  EE            ld      @bcd_adr,TEMP_3
P 030C 00  EE                dec     bcd_adr          !next two digits!
P 030E FA  F1                djnz    bcd_cnt,wrd_b2   !loop for all digits!
P 0310 50  7E                pop     TEMP_3           !restore src bit cnt!
P 0312 7B  04                jr      c,wrd_ex         !dest. overflow!
P 0314 00  7E                dec     TEMP_3
P 0316 EB  DF                jr      nz,wrd_b3        !next bit!
P 0318 AF            wrd_ex: ret
P 0319               END     wrdbcd
```

```
                   GLOBAL
P 0319             hascwrd PROCEDURE
                   !**************************************************
                    Purpose =      To convert a variable length Hex
                                   ASCII string to binary.

                    Input =        RR14 = address of source ASCII
                                          string (in reg/ext/ser memory).

                    Output =       RR12 = binary word (any overflow
                                   high order digits are truncated
                                   without error).
                                   Carry FLAG = 1 if input error
                                                        (serial only)
                                          (SER_flg indicates cause)
                                             = 0 if no error
                                   R14, R15 modified

                    Note =         The ASCII input string processing is
                                   terminated with the occurrence of a
                                   non-hex ASCII character.
                   **************************************************!
                   ENTRY
P 0319 B0  7E                clr    TEMP_3
P 031B B0  EC                clr    R12
P 031D B0  ED                clr    R13            !init output!
P 031F D6  03DA'   has_c1:  call   get_src        !get input!
P 0322 7B  28                jr     c,has_ex1      !error!
P 0324 D6  040D'             call   ver_asc        !verify hex ASCII!
P 0327 7B  22                jr     c,has_ex       !end conversion!
P 0329 A6  7C  39            cp     TEMP_1,#%39
P 032C 3B  03                jr     ule,has_c2
P 032E 26  7C  37            sub    TEMP_1,#%37
                   !Shift left one nibble!
                   !Insert new nibble in least significant nibble!
P 0331 F0  ED      has_c2:  swap   R13
P 0333 D9  7D                ld     TEMP_2,R13
P 0335 56  ED  F0            and    R13,#%F0
P 0338 56  7C  0F            and    TEMP_1,#%0F
P 033B 44  7C  ED            or     R13,TEMP_1
P 033E F0  EC                swap   R12
P 0340 56  EC  F0            and    R12,#%F0
P 0343 56  7D  0F            and    TEMP_2,#%0F
P 0346 44  7D  EC            or     R12,TEMP_2
P 0349 8B  D4                jr     has_c1         !loop!
P 034B CF          has_ex:  rcf                   !no error!
P 034C AF          has_ex1:ret
P 034D             END      hascwrd
```

```
                        GLOBAL
P 034D                  dascwrd PROCEDURE
                        !*****************************************************
                         Purpose =      To convert a variable length decimal
                                        ASCII string to signed binary.

                         Input =        RR14 = address of source ASCII
                                               string (in reg/ext/ser memory).

                         Output =       RR12 = binary word
                                        R8,R9,R10,R11 holds the packed BCD
                                        version of the result.
                                        Carry FLAG = 1 if input error
                                                            (serial only)
                                                (SER_flg indicates cause)
                                                        or dest overflow
                                                     = 0 if no error
                                        R14, R15 modified

                         Note =         The ASCII input string processing is
                                        terminated with the occurrence of a
                                        non-decimal ASCII character.
                                        Decimal ASCII string may be no more
                                        than 6 digits in length, else Carry
                                        will be returned.
                                        Post decimal digits are not included
                                        in the binary result.
                        *****************************************************!
                        ENTRY
P 034D CC  03                   ld      R12,#3          !6 digits!
P 034F DC  08                   ld      R13,#8          !temp addr =!
P 0351 04  FD  ED               add     R13,RP          !R8 thru R11!
P 0354 D6  0363'                call    dascbcd         !convert to bcd!
P 0357 7B  F3                   jr      c,has_ex1       !error!
P 0359 EC  08                   ld      R14,#8
P 035B 04  FD  EE               add     R14,RP
P 035E FC  03                   ld      R15,#3
P 0360 8D  0284'                jp      bcdwrd          !convert to binary!
P 0363                  END     dascwrd
```

```
                         CONSTANT
                          dab_LEN        :=      R12
                          dab_DST        :=      R13
                         GLOBAL
P 0363                   dascbcd PROCEDURE
                         !##################################################
                          Purpose =      To convert a variable length decimal
                                         ASCII string to BCD.

                          Input =        R13 = address of destination BCD
                                               string (in register memory).
                                         RR14 = address of source ASCII
                                                string (in reg/ext/ser memory).
                                         R12 = BCD digit count / 2

                          Output =       BCD string in designated destination
                                         buffer (any overflow high order
                                         digits are truncated without error).
                                         Carry FLAG = 1 if input error
                                                               (serial only)
                                                 (SER_flg indicates cause)
                                                         or overflow
                                         R14, R15 modified.

                          Note =         The ASCII input string processing is
                                         terminated with the occurrence of a
                                         non-decimal ASCII character.
                         ##################################################!
                         ENTRY
P 0363 70  EC                      push   dab_LEN          !save!
P 0365 70  ED                      push   dab_DST
P 0367 B1  ED            das_g1:   clr    @dab_DST         !init. destination!
P 0369 DE                          inc    dab_DST
P 036A CA  FB                      djnz   dab_LEN,das_g1
P 036C B1  ED                      clr    @dab_DST         !init.!
P 036E 50  ED                      pop    dab_DST          !restore!
P 0370 50  EC                      pop    dab_LEN
P 0372 E6  7E  01                  ld     TEMP_3,#1        !for ver_asc!
P 0375 B0  7B                      clr    TEMP_4           !bit 0 => digit seen;
                                                            bit 1 => dec pt seen;
                                                            bit 7 => overflow!
P 0377 D6  03DA'         das_g2:   call   get_src          !get input byte!
P 037A 7B  41                      jr     c,dab_ex1        !serial error!
P 037C 56  7C  7F                  and    TEMP_1,#%7F      !7-bit ASCII!
P 037F 76  7B  03                  tm     TEMP_4,#%03      !check status!
P 0382 EB  0F                      jr     nz,das_g5        !sign char not valid!
P 0384 A6  7C  2B                  cp     TEMP_1,#'+'      !positive?!
P 0387 6B  EE                      jr     z,das_g2         !yes. no affect!
P 0389 A6  7C  2D                  cp     TEMP_1,#'-'      !negative?!
P 038C EB  07                      jr     nz,das_g4        !not sign char!
P 038E B7  ED  80                  xor    @dab_DST,#%80    !complement sign!
P 0391 8B  E4                      jr     das_g2           !get next input!
P 0393 5B  0A            das_g5:   jr     mi,das_g6        !dec pt has been seen!
P 0395 A6  7C  2E        das_g4:   cp     TEMP_1,#'.'      !is char dec pt?!
P 0398 EB  05                      jr     nz,das_g6        !nope.!
P 039A 46  7B  03                  or     TEMP_4,#%03      !dec pt and digit seen!
P 039D 8B  D8                      jr     das_g2           !get next input!
P 039F D6  040D'         das_g6:   call   ver_asc          !is bcd digit?!
P 03A2 7B  16                      jr     c,dab_ex         !end conversion.!
P 03A4 46  7B  01                  or     TEMP_4,#%01      !digit seen!
P 03A7 D6  0463'                   call   rdl              !new digit to dest!
P 03AA EB  09                      jr     nz,das_g7        !overflow!
P 03AC 76  7B  02                  tm     TEMP_4,#%02      !post dec digit?!
P 03AF 6B  C6                      jr     z,das_g2         !no. get next input!
```

```
P 03B1 21   ED              inc      @dab_DST          !inc post dec cnt!
P 03B3 8B   C2              jr       das_g2            !get next input!
P 03B5 46   7B   80  das_g7: or       TEMP_4,#%80       !set overflow!
P 03B8 8B   BD              jr       das_g2            !get next input!

P 03BA E4   7B   FC  dab_ex: ld       FLAGS,TEMP_4      !carry = 0 or 1!
P 03BD AF            dab_ex1: ret
P 03BE               END      dascbcd

                     GLOBAL
P 03BE               wrddasc PROCEDURE
                     !*****************************************************
                      Purpose =       To convert a signed binary word to
                                      decimal ASCII

                      Input =         RR12 = source binary word.
                                      RR14 = address of dest (in reg/ext/ser
                                             memory).

                      Output =        Decimal ASCII in dest buffer.
                                      R8,R9,R10,R11 holds the packed BCD
                                      version of the result.
                                      R12, R13, R14, R15 modified.
                     *****************************************************!
                     ENTRY
P 03BE 70   EE              push     R14
P 03C0 70   EF              push     R15               !save dest addr!
P 03C2 EC   08              ld       R14,#8
P 03C4 04   FD   EE         add      R14,RP            !R8,9,10 & 11 temp!
P 03C7 FC   03              ld       R15,#3            !temp byte length!
P 03C9 D6   02CD'           call     wrdbcd            !convert input word!
P 03CC 50   EF              pop      R15
P 03CE 50   EE              pop      R14               !restore dest addr!
P 03D0 CC   03              ld       R12,#3            !length of temp!
P 03D2 DC   08              ld       R13,#8
P 03D4 04   FD   ED         add      R13,RP            !addr of temp!
P 03D7 8D   0205'           jp       bcddasc           !convert to ASCII!
P 03DA               END      wrddasc
```

```
                        GLOBAL          !for PART II only!
P 03DA                  get_src PROCEDURE
                        !*****************************************************
                         Purpose =      To get source byte from
                                        reg/ext/ser memory into TEMP_1.

                         Output =       Carry FLAG = 1 if error (serial)
                                                   = 0 if all ok
                                        TEMP_1 = source byte.
                                        RR14 updated.
                        *****************************************************!
                        ENTRY
P 03DA CF                       rcf                     !set good return code!
P 03DB EE                       inc     R14             !test R14 = 0!
P 03DC EA  06                   djnz    R14,get_s1      !src in ext memory!
P 03DE FE                       inc     R15             !test R15 = 0!
P 03DF FA  0E                   djnz    R15,get_s2      !src in reg memory!
P 03E1 8D  0000*                jp      ser_get         !src in ser memory!
P 03E4 70  EB           get_s1: push    R11             !save user's!
P 03E6 82  BE                   lde     R11,@RR14       !get byte!
P 03E8 B9  7C                   ld      TEMP_1,R11      !move to common!
P 03EA 50  EB                   pop     R11             !restore user's!
P 03EC A0  EE                   incw    RR14            !update src ptr!
P 03EE AF                       ret
P 03EF E5  EF  7C       get_s2: ld      TEMP_1,@R15     !get byte!
P 03F2 FE                       inc     R15             !update src ptr!
P 03F3 AF                       ret
P 03F4                  END     get_src

                        GLOBAL          !for PART II only!
P 03F4                  put_dest        PROCEDURE
                        !*****************************************************
                         Purpose =      To store destination byte from TEMP_1
                                        into reg/ext/ser memory

                         Output =       RR14 updated.
                        *****************************************************!
                        ENTRY
P 03F4 EE                       inc     R14             !test R14 = 0!
P 03F5 EA  06                   djnz    R14,put_s1      !dest in ext memory!
P 03F7 FE                       inc     R15             !test R15 = 0!
P 03F8 FA  0E                   djnz    R15,put_s2      !dest in reg memory!
P 03FA 8D  0000*                jp      ser_output      !dest in ser memory!
P 03FD 70  EB           put_s1: push    R11             !save user's!
P 03FF B8  7C                   ld      R11,TEMP_1
P 0401 92  BE                   lde     @RR14,R11
P 0403 50  EB                   pop     R11             !restore user's!
P 0405 A0  EE                   incw    RR14
P 0407 AF                       ret
P 0408 F5  7C  EF       put_s2: ld      @R15,TEMP_1
P 040B FE                       inc     R15
P 040C AF                       ret
P 040D                  END     put_dest
```

```
                     1291 CONSTANT
                     1292  MODE            :=      TEMP_3
                     1293  char            :=      TEMP_1
                     1294 INTERNAL
P 040D               1295 ver_asc PROCEDURE
                     1296 !*****************************************************
                     1297  Purpose =       To verify input character as valid
                     1298                  hex or decimal ASCII.
                     1299
                     1300  Input =         TEMP_1 = 8-bit input
                     1301                  TEMP_3 = 0 => test for hex,
                     1302                           1 => test for decimal
                     1303
                     1304  Output =        Carry FLAG = 0 if no error
                     1305                               1 if error.
                     1306 *****************************************************!
                     1307 ENTRY
P 040D 56  7C  7F    1308         and     char,#%7F       !7-bit ASCII!
P 0410 A6  7C  30    1309         cp      char,#'0'       !range start: '0'!
P 0413 7B  16        1310         jr      ult,ver_err     !no good!
P 0415 A6  7C  3A    1311         cp      char,#'9'+1     !dec range end: '9'!
P 0418 7B  10        1312         jr      ult,ver_ok      !all's well!
P 041A 76  7E  01    1313         tm      MODE,#1         !dec or hex?!
P 041D EB  0B        1314         jr      nz,ver_erc      !no good!
P 041F 56  7C  DF    1315         and     char,#LNOT('a'-'A') !insure upper case!
P 0422 A6  7C  41    1316         cp      char,#'A'       !check A-F range!
P 0425 7B  04        1317         jr      ult,ver_err     !no good!
P 0427 A6  7C  47    1318         cp      char,#'F'+1     !end hex range!
                     1319 ver_ok:
P 042A EF            1320 ver_erc: ccf                    !complement carry!
P 042B AF            1321 ver_err: ret
P 042C               1322 END     ver_asc

                     1324 INTERNAL
P 042C               1325 bcd_bin PROCEDURE
                     1326 !*****************************************************
                     1327  Purpose =       To convert next bcd digit to binary.
                     1328
                     1329  Input =         TEMP_1 = digit
                     1330
                     1331  Output =        RR12 = RR12 * 10 + digit
                     1332 *****************************************************!
                     1333 ENTRY
P 042C 56  7C  0F    1334         and     TEMP_1,#%0F     !isolate digit!
P 042F A6  7C  09    1335         cp      TEMP_1,#9       !verify valid!
P 0432 BB  2D        1336         jr      ugt,bcd_b1      !error!
P 0434 02  DD        1337         add     R13,R13
P 0436 12  CC        1338         adc     R12,R12         !2x!
P 0438 7B  27        1339         jr      c,bcd_b1        !overflow!
P 043A 70  EC        1340         push    R12
P 043C 70  ED        1341         push    R13
P 043E 02  DD        1342         add     R13,R13
P 0440 12  CC        1343         adc     R12,R12         !4x!
P 0442 7B  19        1344         jr      c,bcd_b2        !overflow!
P 0444 02  DD        1345         add     R13,R13
P 0446 12  CC        1346         adc     R12,R12         !8x!
P 0448 7B  13        1347         jr      c,bcd_b2        !overflow!
P 044A 04  7C  ED    1348         add     R13,TEMP_1
P 044D 16  EC  00    1349         adc     R12,#0          !8x + d!
P 0450 7B  0B        1350         jr      c,bcd_b2        !overflow!
P 0452 50  7C        1351         pop     TEMP_1
P 0454 04  7C  ED    1352         add     R13,TEMP_1
P 0457 50  7C        1353         pop     TEMP_1
P 0459 14  7C  EC    1354         adc     R12,TEMP_1      !10x + d!
P 045C AF            1355         ret
                     1356
P 045D 50  7C        1357 bcd_b2: pop     TEMP_1
P 045F 50  7C        1358         pop     TEMP_1          !restore stack!
P 0461 DF            1359 bcd_b1: scf                     !error!
P 0462 AF            1360         ret
P 0463               1361 END     bcd_bin
```

```
                        CONSTANT
                         s_len          :=      R12
                         s_adr          :=      R13
                        INTERNAL
P 0463                  rdl     PROCEDURE
                        !****************************************************
                         Rotate Digit Left

                         Input =          R12 = BCD string length
                                          R13 = BCD string address
                                          TEMP_1 bit 3-0 = new digit

                         Output =         BCD string rotated left one digit;
                                          new digit inserted in units position.
                                          TEMP_1 bit 3-0 = digit rotated out
                                                 of high order digit position
                                                bit 7-4 = 0
                                          Zero FLAG = 1 if TEMP_1 <> 0
                                          R12, R13 unmodified
                        ****************************************************!
                        ENTRY
P 0463 70  EC                   push    s_len
P 0465 02  DC                   add     s_adr,s_len      !address of units place!
P 0467 F1  ED           rdl_01: swap    @s_adr
P 0469 E5  ED  7D               ld      TEMP_2,@s_adr
P 046C 57  ED  F0               and     @s_adr,#%F0      !isolate digit!
P 046F 56  7C  0F               and     TEMP_1,#%0F      !isolate new digit!
P 0472 45  ED  7C               or      TEMP_1,@s_adr
P 0475 F5  7C  ED               ld      @s_adr,TEMP_1    !save new byte!
P 0478 E4  7D  7C               ld      TEMP_1,TEMP_2
P 047B 00  ED                   dec     s_adr            !back-up pointer!
P 047D CA  E8                   djnz    s_len,rdl_01     !loop till done!
P 047F 56  7C  0F               and     TEMP_1,#%0F      !old high order digit!
P 0482 50  EC                   pop     s_len            !restore R12!
P 0484 AF                       ret
P 0485                  END     rdl

                        INTERNAL
P 0485                  rdr     PROCEDURE
                        !****************************************************
                         Rotate Digit Right

                         Input =          R12 = BCD string length
                                          R13 = BCD string address
                                          TEMP_1 bit 7-4 = new digit

                         Output =         BCD string rotated right one digit;
                                          new digit inserted in high order
                                          position.
                                          R12 unmodified
                                          R13 modified
                        ****************************************************!
                        ENTRY
P 0485 70  EC                   push    s_len
P 0487 DE               rdr_01: inc     s_adr
P 0488 F1  ED                   swap    @s_adr
P 048A E5  ED  7E               ld      TEMP_3,@s_adr
P 048D 57  ED  0F               and     @s_adr,#%0F      !isolate digit!
P 0490 56  7C  F0               and     TEMP_1,#%F0      !isolate new digit!
P 0493 45  ED  7C               or      TEMP_1,@s_adr
P 0496 F5  7C  ED               ld      @s_adr,TEMP_1    !save new byte!
P 0499 E4  7E  7C               ld      TEMP_1,TEMP_3
P 049C CA  E9                   djnz    s_len,rdr_01     !loop till done!
P 049E 50  EC                   pop     s_len            !restore R12!
P 04A0 AF                       ret
P 04A1                  END     rdr
```

```
                    CONSTANT
                     tjm_bits       :=      R12
                     tjm_mask       :=      R13
                    GLOBAL
P 04A1              clb     PROCEDURE
                    !*******************************************************
                     Purpose =      To collect selected bits in a byte
                                    into adjacent bits in the low order
                                    end of the byte.  Upper bits in byte
                                    are set to zero.

                     Input =        R12 = input byte
                                    R13 = mask. Bit = 1 => corresponding
                                            input bit is selected.

                     Output =       R12 = collected bits

                     Note =         For example:
                                    Input :  R12 = %(2)01110110
                                             R13 = %(2)10000101

                                    Output : R12 = %(2)00000010
                    *******************************************************!
                    ENTRY
P 04A1 E6  7C  08           ld      TEMP_1,#8       !bit count!
P 04A4 B0  7D               clr     TEMP_2          !bits collected here!
P 04A6 90  EC       next1:  rl      tjm_bits        !bit 7 to bit 0!
P 04A8 90  ED               rl      tjm_mask        !bit 7 to carry!
P 04AA FB  06               jr      nc,no_select    !don't use this bit!
P 04AC E0  EC               rr      tjm_bits
P 04AE 90  EC               rl      tjm_bits        !bit 7 to 0 and carry!
P 04B0 10  7D               rlc     TEMP_2          !collect source bit!
                    no_select:
P 04B2 00  7C               dec     TEMP_1
P 04B4 EB  F0               jr      nz,next1        !repeat!
P 04B6 C8  7D               ld      R12,TEMP_2
P 04B8 AF                   ret
P 04B9              END     clb
```

```
                       CONSTANT
                        tjm_tabh        :=      R14
                        tjm_tabl        :=      R15
                        tjm_tab         :=      RR14
                       GLOBAL
P 04B9                 tjm     PROCEDURE
                       !*******************************************************
                        Purpose =      To take a jump to a routine address
                                       determined by the state of selected
                                       bits in a source byte.  A bit
                                       is 'selected' by a one in the
                                       corresponding position of a mask.
                                       The 'selected' bits are packed into
                                       adjacent bits in the low order end of
                                       the byte.  This value is then doubled,
                                       and used as an index into the jump
                                       table.

                        Input =        RR14 = address of jump table in
                                               program memory.
                                       R12 = input data
                                       R13 = mask
                       *******************************************************!
                       ENTRY
P 04B9 D6   04A1'              call    clb            !collect selected bits!
P 04BC 02   CC                 add     tjm_bits,tjm_bits !collected bits * 2!
P 04BE 16   EE   00            adc     tjm_tabh,#0     !in case carry!
P 04C1 02   FC                 add     tjm_tabl,tjm_bits
P 04C3 16   EE   00            adc     tjm_tabh,#0     !tjm_tab points to...!
P 04C6 C2   DE                 ldc     tjm_mask,@tjm_tab !...table entry!
P 04C8 A0   EE                 incw    tjm_tab
P 04CA C2   FE                 ldc     tjm_tabl,@tjm_tab !get table entry...!
P 04CC E8   ED                 ld      tjm_tabh,tjm_mask !...into tjm_tab!

P 04CE 30   EE                 jp      @tjm_tab        !bye!

P 04D0                 END     tjm
                       END PART_I

   0 errors
Assembly complete
```

```
Z8ASM    3.02
LOC    OBJ CODE    STMT SOURCE STATEMENT

                      1
                      2
                      3 PART_II MODULE
                      4
                      5
                      6 !'ROMLESS Z8'   SUBROUTINE LIBRARY  PART II
                      7 !
                      9 CONSTANT
                     10 !Register Usage!
                     11
                     12 RAM_START         :=      %7F
                     13
                     14 P3M_save          :=      RAM_START
                     15 TEMP_3            :=      P3M_save-1
                     16 TEMP_2            :=      TEMP_3-1
                     17 TEMP_1            :=      TEMP_2-1
                     18 TEMP_4            :=      TEMP_1-1
                     19
                     20 !The following registers are modified/referenced
                     21  by the Serial Routines ONLY.  They are
                     22  available as general registers to the user
                     23  who does not intend to make use of the
                     24  Serial Routines!
                     25
                     26 SER_char          :=      TEMP_4-1
                     27 SER_tmp2          :=      SER_char-1
                     28 SER_tmp1          :=      SER_tmp2-1
                     29 SER_put           :=      SER_tmp1-1
                     30 SER_len           :=      SER_put-1
                     31 SER_buf           :=      SER_len-2
                     32 SER_imr           :=      SER_buf-1
                     33 SER_cfg           :=      SER_imr-1
                     34 !Serial Configuration Data
                     35 bit 7 : =1 => odd parity on
                     36 bit 6 : =1 => even parity on
                     37  (bit 6,7 = 11 => undefined)
                     38 bit 5 : undefined
                     39 bit 4 : undefined
                     40 bit 3 : =1 => input editting on
                     41 bit 2 : =1 => auto line feed enabled
                     42 bit 1 : =1 => BREAK detection enabled
                     43 bit 0 : =1 => input echo on
                     44 !
                     45 op      :=      %80
                     46 ep      :=      %40
                     47 ie      :=      %08
                     48 al      :=      %04
                     49 be      :=      %02
                     50 ec      :=      %01
                     51 SER_get           :=      SER_cfg-1
                     52 SER_flg           :=      SER_get-1
                     53 !Serial Status Flags
                     54 bit 7 : =1 => serial I/O disabled
                     55 bit 6 : undefined
                     56 bit 5 : undefined
                     57 bit 4 : =1 => parity error
                     58 bit 3 : =1 => BREAK detected
                     59 bit 2 : =1 => input buffer overflow
                     60 bit 1 : =1 => input buffer not empty
                     61 bit 0 : =1 => input buffer full
                     62 !
                     63 sd      :=      %80
                     64 pe      :=      %10
                     65 bd      :=      %08
                     66 bo      :=      %04
                     67 bne     :=      %02
                     68 bf      :=      %01
                     69
```

```
RAM_TMR           :=       RAM_START-%10

SERltime          :=       SER flg-1
SERhtime          :=       SERltime-1

!The following registers are modified/referenced
 by the Timer/Counter Routines ONLY.  They are
 available as general registers to the user
 who does not intend to make use of the
 Timer/Counter Routines!

TOD_tic           :=       RAM_TMR-2
TOD_imr           :=       TOD_tic-1
TOD_hr            :=       TOD_imr-1
TOD_min           :=       TOD_hr-1
TOD_sec           :=       TOD_min-1
TOD_tt            :=       TOD_sec-1
PLS_1             :=       TOD_tt-1
PLS_tmr           :=       PLS_1-1
PLS_2             :=       PLS_tmr-1

RAM_END           :=       PLS_2
STACK             :=       RAM_END

!Equivalent working register equates
 for above register layout!

!register file %70 - %7F!
RAM_STARTr        :=       %70       !for SRP!

rP3Msave          :=       R15
rTEMP_3           :=       R14
rTEMP_2           :=       R13
rTEMP_1           :=       R12
rrTEMP_1          :=       RR12
rTEMP_1h          :=       R12
rTEMP_1l          :=       R13
rTEMP_4           :=       R11
rSERchar          :=       R10
rSERtmp2          :=       R9
rSERtmp1          :=       R8
rrSERtmp          :=       RR8
rSERtmpl          :=       R9
rSERtmph          :=       R8
rSERput           :=       R7
rSERlen           :=       R6
rrSERbuf          :=       RR4
rSERbufh          :=       R4
rSERbufl          :=       R5
rSERimr           :=       R3
rSERcfg           :=       R2
rSERget           :=       R1
rSERflg           :=       R0


!register file %60 - %6F!
RAM_TMRr          :=       %60       !for SRP!
rTODtic           :=       R13
rTODimr           :=       R12
rTODhr            :=       R11
rTODmin           :=       R10
rTODsec           :=       R9
rTODtt            :=       R8
rPLS_1            :=       R7
rPLStmr           :=       R6
rPLS_2            :=       R5
```

```
                     CONSTANT
                      si_PTR              :=      RR14
                      si_TMP1             :=      R11
                      si_TMP2             :=      R13
                     GLOBAL
P 0000               ser_init            PROCEDURE
                     !*******************************************************
                     serial initialize

                      Purpose =          To initialize the serial channel and
                                         RAM flags for serial I/O.  Serial
                                         input occurs under interrupt control.
                                         Serial output occurs in a polled mode.

                      Input =            RR14 = address of parameter list in
                                                program memory (if R14 = 0,
                                                use defaults):
                                          1 byte = Serial Configuration Data
                                                (see definition of SER_cfg)
                                          1 byte = IMR mask for nestable
                                                interrupts
                                          1 word = address of circular input
                                                buffer (in reg/ext memory)
                                          1 byte = Length of input buffer
                                          1 byte = Baud rate counter value
                                          1 byte = Baud rate prescaler value
                                                  (unshifted)

                      Output =           Serial I/O operations initialized.
                                         R11, R12, R13, R14, R15 modified.

                      Note =             Defaults:
                                          Input echo on
                                          Input editting on
                                          BREAK detection enabled
                                          No parity
                                          Auto line feed on
                                          Input Buffer Address = SER_char
                                          Input buffer length = 1 byte
                                          Baud Rate = 9600 (assuming
                                                XTAL = 7.3728 MHz)

                                         The instruction at %0809 must result
                                         in a jump to the jump table entry for
                                         ser_input.

                                         If BREAK detection is disabled, and a
                                         BREAK occurs, it will be received as a
                                         continuous string of null characters.

                                         The parameter list is not referenced
                                         following initialization.
                     *******************************************************!
                     ENTRY
P 0000 EE                        inc     R14              !use defaults?!
P 0001 EA  04                    djnz    R14,si_1         !no. given by caller.!
P 0003 EC  00*                   ld      R14,#HI ser_def  !address of default...!
P 0005 FC  51*                   ld      R15,#LO ser_def  !... parameter list.  !
P 0007 BC  72        si_1:       ld      si_TMP1,#SER_cfg
P 0009 DC  05                    ld      si_TMP2,#5
P 000B C3  BE        si_2:       ldci    @si_TMP1,@si_PTR !get initialization...!
P 000D DA  FC                    djnz    si_TMP2,si_2     !...parameters!
P 000F 56  73   F7               and     SER_imr,#%F7     !insure no self-nesting!
```

```
                    !initialize Port 3 Mode Register for serial I/O!
P 0012 56   F1  FC             AND     TMR,#%FC           !disable T0!
P 0015 B8   72                 ld      si_TMP1,SER_cfg    !configuration data!
P 0017 56   EB  80             AND     si_TMP1,#%80       !odd parity select!
P 001A 46   EB  40             OR      si_TMP1,#%40       !P30/7 = Sin/Sout!
P 001D 56   7F  3F             AND     P3M_save,#%3F      !mask off old settings!
P 0020 44   EB  7F             OR      P3M_save,si_TMP1   !new selection!
P 0023 E4   7F  F7             LD      P3M,P3M_save       !to write-only register!

                    !initialize T0!
P 0026 BC   F4                 ld      si_TMP1,#T0
P 0028 C2   DE                 ldc     si_TMP2,@si_PTR    !save counter!
P 002A C3   BE                 ldci    @si_TMP1,@si_PTR   !init counter!
P 002C C2   BE                 ldc     si_TMP1,@si_PTR    !get prescaler!
P 002E D6   0000*              call    multiply           !T0 x PRE0!
P 0031 C9   6E                 ld      SERhtime,R12       !save for BREAK...!
P 0033 D9   6F                 ld      SERltime,R13       !...detection     !
P 0035 90   EB                 rl      si_TMP1            !SHL 1!
P 0037 DF                      scf                        !continuous mode!
P 0038 10   EB                 rlc     si_TMP1            !SHL 2!
P 003A B9   F5                 ld      PRE0,si_TMP1
                    !initialize RAM flags and pointers!
P 003C 8F                      DI                         !disable interrupts!
P 003D B0   71                 clr     SER_get            !input buffer...!
P 003F B0   77                 clr     SER_put            !...empty!
P 0041 B0   70                 clr     SER_flg            !no errors!

                    !initialize interrupts!
P 0043 56   FA  E7             AND     IRQ,#%E7           !clear IRQ3 & 4!
P 0046 56   FB  EF             and     IMR,#%EF           !disable IRQ4 (xmt)!
P 0049 46   FB  08             or      IMR,#%08           !enable IRQ3 (rcv)!
P 004C 9F                      EI
                    !go!
P 004D 46   F1  03             or      TMR,#%03           !load/enable T0!
P 0050 AF                      ret
P 0051              END        ser_init



                    !Defaults for serial initialization!

P 0051 0F   00      ser_def RECORD  [cfg_, imr_              BYTE
P 0053 007A  01
P 0056 02   03
                                        buf_                 WORD
                                        len_, ctr_, pre_     BYTE]
                               :=
                               [ec+al+ie+be, %00, SER_char, 1, %02, %03]
```

```
                        CONSTANT
                         rli_len        :=      R13
                        GLOBAL
P 0058                  ser rlin        PROCEDURE
                        !*****************************************************
                        read line

                         Purpose =      To return input from serial channel
                                        up to 'carriage return' character or
                                        maximum length requested or BREAK.

                         Input =        RR14 = address of destination buffer
                                                (in reg/ext memory)
                                        R13 = maximum length

                         Output =       Input characters is destination buffer.
                                        RR14 = unmodified
                                        R13 = length returned
                                        Carry Flag = 1 if any error,
                                                   = 0 if no error.
                                        R12 indicates read status

                         Note =         1. Return will be made to the calling
                                        program only after the requisite
                                        characters have been received from
                                        the serial line.

                                        2. If input editting is enabled, a
                                        'backspace' character will cause
                                        the previous character (if any) in the
                                        the destination buffer to be deleted;
                                        a 'delete' character will cause all
                                        previous characters (if any) in the
                                        destination buffer to be deleted.

                                        3. If parity (odd or even) is enabled,
                                        the parity error flag (R14) will be set
                                        if any character returned had a parity
                                        error. (Bit 7 of each character may
                                        then be examined if it is desirable to
                                        know which character(s) had the error).

                                        4. The status flags 'BREAK detected',
                                        'parity error', and 'input buffer
                                        overflow' will be returned
                                        as part of R12, but will be cleared in
                                        SER_stat.

                                        5. The staus flags: 'input buffer full'
                                        and 'input buffer not empty' will be
                                        updated in SER stat.
                        *****************************************************!
                        ENTRY
P 0058 B0  7E                   clr     TEMP_3          !flag => read line!
                        ser_read:
P 005A 70  EE                   push    R14             !save original...!
P 005C 70  EF                   push    R15             !...dest. pointer!
P 005E 70  ED                   push    rli_len         !...and length!
P 0060 D6  0170'        rli_4:  call    ser_get         !get input character!
P 0063 7B  48                   jr      c,rli_3         !error!
P 0065 76  72  C0               tm      SER_cfg,#op LOR ep !parity enabled?!
P 0068 6B  08                   jr      z,rli_1         !no!
P 006A 76  7C  80               tm      TEMP_1,#%80     !parity error?!
P 006D 6B  03                   jr      z,rli_1         !no!
```

```
P 006F 46   70  10   339          or      SER_flg,#pe      !yes. set error flag!
P 0072 D6   0000*    340 rli_1:   call    put_dest         !store in buffer!
P 0075 A6   7E  00   341          cp      TEMP_3,#0        !read line?!
P 0078 EB   31       342          jr      nz,rli_2         !no!
P 007A 56   7C  7F   343          and     TEMP_1,#%7F      !ignore parity bit!
P 007D 76   72  08   344          tm      SER_cfg,#ie      !input editting on?!
P 0080 6B   21       345          jr      z,rli_9          !no.!
                     346 !input editting!
P 0082 A6   7C  7F   347          cp      TEMP_1,#%7F      !char = delete?!
P 0085 6B   3E       348          jr      z,rli_6          !yes!
P 0087 A6   7C  08   349          cp      TEMP_1,#%08      !char = backspace?!
P 008A EB   17       350          jr      nz,rli_9         !no. continue!
P 008C 50   7C       351          pop     TEMP_1           !get original length!
P 008E 70   7C       352          push    TEMP_1
P 0090 A4   ED  7C   353          cp      TEMP_1,rli_len   !any characters?!
P 0093 6B   30       354          jr      eq,rli_6         !none!
P 0095 DE            355          inc     rli_len          !undo last decrement!
P 0096 26   EF  02   356          sub     R15,#2           !backspace & previous!
P 0099 EE            357          inc     R14              !reg or ext mem?!
P 009A EA   02       358          djnz    R14,rli_7        !ext!
P 009C 8B   C2       359          jr      rli_4            !reg!
P 009E 36   EE  00   360 rli_7:   sbc     R14,#0
P 00A1 8B   BD       361          jr      rli_4
                     362
P 00A3 00   ED       363 rli_9:   dec     rli_len          !in case cr!
P 00A5 A6   7C  0D   364          cp      TEMP_1,#%0D      !carriage return?!
P 00A8 6B   03       365          jr      z,rli_3          !end input!
P 00AA DE            366          inc     rli_len          !restore!
P 00AB DA   B3       367 rli_2:   djnz    rli_len,rli_4    !loop for max length!
P 00AD 50   7C       368 rli_3:   pop     TEMP_1           !original length!
P 00AF 24   ED  7C   369          sub     TEMP_1,rli_len   !# chars returned!
P 00B2 D8   7C       370          ld      rli_len,TEMP_1   !tell caller!
P 00B4 C8   70       371          ld      R12,SER_flg      !return read status!
P 00B6 56   70  E3   372          and     SER_flg,#LNOT (pe LOR bd LOR bo)
                     373                                   !reset for next time!
P 00B9 CF            374          rcf                      !good return code!
P 00BA 76   EC  9C   375          tm      R12,#pe LOR bd LOR bo LOR sd
P 00BD 6B   01       376          jr      z,rli_5          !no error!
P 00BF DF            377          scf                      !set error return!
P 00C0 50   EF       378 rli_5:   pop     R15
P 00C2 50   EE       379          pop     R14              !original buffer addr!
P 00C4 AF            380          ret
                     381
P 00C5 50   ED       382 rli_6:   pop     rli_len
P 00C7 50   EF       383          pop     R15
P 00C9 50   EE       384          pop     R14
P 00CB 8B   8D       385          jr      ser_read         !start over!
P 00CD               386 END      ser_rlin
                     388 GLOBAL
P 00CD               389 ser_rabs         PROCEDURE
                     390 !##################################################
                     391 read absolute
                     392
                     393  Purpose =       To return input from serial channel
                     394                  of maximum length requested.  (Input
                     395                  is not terminated with the receipt of
                     396                  a 'carriage return'.  BREAK will
                     397                  terminate read.)
                     398
                     399  Note =          All other details are as for 'ser_rlin'.
                     400 ##################################################!
                     401 ENTRY
P 00CD E6   7E  01   402          ld      TEMP_3,#1        !flag => read absolute!
P 00D0 8B   88       403          jr      ser_read
P 00D2               404 END      ser_rabs
```

```
                           GLOBAL
P 00D2                     ser_input       PROCEDURE
                           !*****************************************************
                           Interrupt service - Serial Input

                            Purpose =      To service IRQ3 by inputting current
                                           character into next available position
                                           in circular buffer.

                            Input =        None.

                            Output =       New character inserted in buffer.
                                           SER_stat , SER_put updated.

                            Note =         1. If even parity enabled, the software
                                           replaces the eigth data bit with a
                                           parity error flag.

                                           2. If BREAK detection is enabled, and
                                           the received character is null,
                                           the serial input line is monitored to
                                           detect a potential BREAK condition.
                                           BREAK is defined as a zero start bit
                                           followed by 8 zero data bits and a
                                           zero stop bit.

                                           3. If 'buffer full' on entry, 'input
                                           buffer overflow' is flagged.

                                           4. If input echo is on, the character is
                                           immediately sent to the output serial
                                           channel.

                                           5. IMR is modified to allow selected
                                           nested interrupts (see ser_init).
                           *****************************************************!
                           ENTRY
P 00D2 E4  03  78                  ld      SER_tmp1,%03      !read stop bit level!
P 00D5 70  FB                      push    imr               !save entry imr!
P 00D7 54  73  FB                  and     imr,SER_imr       !allow nesting!
P 00DA 9F                          ei
P 00DB 70  FD                      push    rp                !save user's!
P 00DD 31  70                      srp     #RAM_STARTr
P 00DF A8  F0                      ld      rSERchar,SIO      !capture input!
P 00E1 76  E2  02                  tm      rSERcfg,#be       !break detect enabled?!
P 00E4 6B  2F                      jr      z,ser_30          !nope.!
P 00E6 B0  E9                      clr     rSERtmp2
P 00E8 76  E2  80                  tm      rSERcfg,#op       !odd parity enabled?!
P 00EB 6B  02                      jr      z,ser_23          !no.!
P 00ED 9C  80                      ld      rSERtmp2,#%80
P 00EF A2  A9              ser_23: cp      rSERchar,rSERtmp2 !8 received bits = 0?!
P 00F1 EB  22                      jr      ne,ser_30         !no!
P 00F3 76  E8  01                  tm      rSERtmp1,#1       !test stop bit!
P 00F6 EB  1D                      jr      nz,ser_30         !not BREAK!
                           !is BREAK. Wait for marking!
P 00F8 46  E0  08                  or      rSERflg,#bd       !set BREAK flag!
P 00FB 76  03  01          ser_24: tm      %03,#1            !marking yet?!
P 00FE 6B  FB                      jr      z,ser_24          !not yet!
                           !wait 1 char time to flush receive shift register!
P 0100 70  6E                      push    SERhtime
P 0102 70  6F                      push    SERltime          !save PRE0 x T0!
P 0104 8C  35              in_loop: ld     rSERtmp1,#53
P 0106 8A  FE              lp1:    djnz    rSERtmp1,lp1      !delay 640 cycles!
P 0108 80  6E                      decw    SERhtime
```

```
P 010A EB   F8                  jr      nz,in_loop       !delay (128x10xPREOxTO)!
                                                         !      ----------------!
                                                         !             2        !
P 010C 50   6F                  pop     SERltime
P 010E 50   6E                  pop     SERhtime         !restore PREO x TO!
P 0110 56   FA   F7             and     IRQ,#LNOT %08    !clear int req!
P 0113 8B   49                  jr      ser_i5           !bye!

P 0115 76   E0   01    ser_30:  tm      rSERflg,#bf      !buffer full?!
P 0118 EB   4A                  jr      nz,ser_i1        !yes.overflow!
P 011A 76   E2   01             tm      rSERcfg,#ec      !echo on?!
P 011D 6B   0A                  jr      z,ser_i0         !no!
P 011F A9   F0                  ld      SIO,rSERchar     !echo!
P 0121 66   FA   10    ser_i6:  tcm     IRQ,#%10         !poll!
P 0124 EB   FB                  jr      nz,ser_i6        !loop!
P 0126 56   FA   EF             and     IRQ,#LNOT %10    !clear irq bit!
P 0129 76   E2   40    ser_i0:  tm      rSERcfg,#ep      !even parity?!
P 012C 6B   14                  jr      z,ser_22         !no parity!
                       !calculate parity error flag!
P 012E 8C   07                  ld      rSERtmp1,#7
P 0130 B0   E9                  clr     rSERtmp2         !count 1's here!
P 0132 C0   EA         ser_20:  rrc     rSERchar         !bit to carry!
P 0134 16   E9   00             adc     rSERtmp2,#0      !update 1's count!
P 0137 8A   F9                  djnz    rSERtmp1,ser_20  !loop till done!
P 0139 56   E9   01             and     rSERtmp2,#1      !1's count even or odd?!
P 013C B2   A9                  xor     rSERchar,rSERtmp2
P 013E C0   EA                  rrc     rSERchar         !parity error flag...!
P 0140 C0   EA                  rrc     rSERchar         !...to bit 7!
P 0142 88   E4         ser_22:  ld      rSERtmph,rSERbufh
P 0144 98   E5                  ld      rSERtmpl,rSERbufl
P 0146 02   97                  add     rSERtmpl,rSERput !next char address!
P 0148 8E                       inc     rSERtmph         !in external memory?!
P 0149 8A   1E                  djnz    rSERtmph,ser_i2  !yes.!
P 014B F3   9A                  ld      @rSERtmpl,rSERchar !store char in buf!
P 014D 46   E0   02    ser_i3:  or      rSERflg,#bne     !buffer not empty!
P 0150 7E                       inc     rSERput          !update put ptr!
P 0151 A2   76                  cp      rSERput,rSERlen  !wrap-around?!
P 0153 EB   02                  jr      ne,ser_i4        !no!
P 0155 B0   E7                  clr     rSERput          !set to start!
P 0157 A2   71         ser_i4:  cp      rSERput,rSERget  !if equal, then full!
P 0159 EB   03                  jr      ne,ser_i5
P 015B 46   E0   01             or      rSERflg,#bf
P 015E 50   FD         ser_i5:  pop     rp               !restore user's!
P 0160 8F                       di
P 0161 50   FB                  pop     imr              !restore entry imr!
P 0163 BF                       iret

P 0164 46   E0   04    ser_i1:  or      rSERflg,#bo      !buffer overflow!
P 0167 8B   F5                  jr      ser_i5

P 0169 16   E8   00    ser_i2:  adc     rSERtmph,#0
P 016C 92   A8                  lde     @rrSERtmp,rSERchar !store in buf!
P 016E 8B   DD                  jr      ser_i3
P 0170                 END      ser_input
```

```
                    GLOBAL          !for PART I!
P 0170              ser_get PROCEDURE
                    !*****************************************************
                     Purpose =      To return one serial input character.

                     Input =        None.

                     Output =       Carry FLAG = 1 if BREAK detected or
                                                 serial not enabled
                                                 or buffer overflow
                                             = 0 otherwise
                                    TEMP_1 = character

                     Note =         This routine will not return control
                                    until a character is available in the
                                    input buffer or an error is detected.
                    *****************************************************!
                    ENTRY
P 0170 70  FD                push   rp              !save caller's rp!
P 0172 31  70                srp    #RAM_STARTr     !point to subr. RAM!
P 0174 DF                    scf                    !in case error!
P 0175 76  E0  8C   ser_g1:  tm     rSERflg,#sd LOR bd LOR bo
                                                    !serial disabled or
                                                     BREAK detected or
                                                     buffer overflow?!
P 0178 EB  24                jr     nz,ser_g6       !yes.!
P 017A 76  E0  02            tm     rSERflg,#bne    !buffer not empty?!
P 017D 6B  F6                jr     z,ser_g1        !empty. wait!
P 017F D8  E5                ld     rTEMP_1l,rSERbufl
P 0181 C8  E4                ld     rTEMP_1h,rSERbufh
P 0183 8F                    di                     !prevent IRQ3 conflict!
P 0184 02  D1                add    rTEMP_1l,rSERget !next char address!
P 0186 CE                    inc    rTEMP_1h        !input buffer in...!
P 0187 CA  18                djnz   rTEMP_1h,ser_g3 !...external memory!
                                                    !...register memory!
P 0189 E3  CD                ld     rTEMP_1,@rTEMP_1l !get char!
P 018B 56  E0  FE   ser_g4:  and    rSERflg,#LNOT bf !buffer not full!
P 018E 1E                    inc    rSERget         !update get pointer!
P 018F A2  16                cp     rSERget,rSERlen !wrap-around?!
P 0191 EB  02                jr     ne,ser_g2       !no.!
P 0193 B0  E1                clr    rSERget         !yes. set to start!
P 0195 A2  17       ser_g2:  cp     rSERget,rSERput !buffer empty if get...!
P 0197 EB  03                jr     ne,ser_g5       !...and put =!
P 0199 56  E0  FD            and    rSERflg,#LNOT bne !buffer empty now!
P 019C CF           ser_g5:  rcf                    !set good return!
P 019D 9F                    ei                     !re-enable interrupts!
P 019E 50  FD       ser_g6:  pop    rp              !restore caller's rp!
P 01A0 AF                    ret

P 01A1 16  EC  00   ser_g3:  adc    rTEMP_1h,#0     !rrTEMP_1 has char addr!
P 01A4 82  CC                lde    rTEMP_1,@rrTEMP_1 !get char!
P 01A6 8B  E3                jr     ser_g4          !clean up!
P 01A8              END      ser_get
```

```
                          GLOBAL
P 01A8                    ser_break       PROCEDURE
                          !*******************************************************
                          break transmission

                           Purpose =      To transmit BREAK on the serial line.

                           Input =        RR14 = break length

                           Output =       None.

                           Note =         BREAK is defined as:
                                           serial out (P37) = 0 for
                                          2        x 28 cycles/loop x RR14 loops
                                          -----
                                          XTAL

                                          RR14 should yield at least 1 bit time
                                          so that the last 'clr SIO' will
                                          have been preceded by at least 1 bit
                                          time of spacing. Therefore, RR14 should
                                          be greater than or equal to

                                          4 x 16 x PREO x TO
                                          ------------------
                                                  28
                          *******************************************************!
                          ENTRY
                          ser_b1:
P 01A8 B0  F0                     clr     SIO
P 01AA 80  EE                     decw    RR14
P 01AC EB  FA                     jr      nz,ser_b1
                          !wait for last null to be fully transmitted!
P 01AE 8D  0238'                  jp      ser_o1
P 01B1                    END     ser_break

                          GLOBAL
P 01B1                    ser_flush       PROCEDURE
                          !*******************************************************
                          input flush

                           Purpose =      To flush (clear) the serial input
                                          buffer of characters.

                           Input =        None

                           Output =       Empty input buffer.

                           Note =         This routine might be useful to clear
                                          all past input after a BREAK has been
                                          detected on the line.
                          *******************************************************!
                          ENTRY
P 01B1 8F                         di              !disable interrupts!
                                                  !(to avoid collision with
                                                   serial input)!
P 01B2 B0  71                     clr     SER_get !buffer start!
P 01B4 B0  77                     clr     SER_put != buffer end!
P 01B6 56  70  80                 and     SER_flg,#%80    !clear status!
P 01B9 9F                         ei              !re-enable interrupts!
P 01BA AF                         ret
P 01BB                    END     ser_flush
```

```
                        CONSTANT
                         wli_len          :=       R13
                        GLOBAL
P 01BB                  ser_wlin          PROCEDURE
                        !*****************************************************
                        write line

                         Purpose =        To output a character string to serial
                                          line, ending with either a 'carriage
                                          return' character or the maximum length
                                          specified.

                         Input =          RR14 = address of source buffer
                                                 (in reg/ext memory)
                                          R13 = length

                         Output =         RR14 = updated
                                          Carry Flag = 1 if serial not enabled,
                                                     = 0 if no error.
                                          R13 = # bytes output (not including
                                                         auto line feed)

                         Note =           If auto line feed is enabled, a
                                          line feed character will be output
                                          following each carriage return
                                          (ser_wlin only).
                        *****************************************************!
                        ENTRY
P 01BB B0  7E                   clr       TEMP_3           !flag => write line!

P 01BD DF               write:  scf                        !in case error!
P 01BE 76  70  80               tm        SER_flg,#sd      !serial disabled?!
P 01C1 EB  30                   jr        nz,wli_1         !yes. error!
P 01C3 70  ED                   push      wli_len
P 01C5 D6  0000*        wli_4:  call      get_src
P 01C8 D6  020B'                call      ser_output       !write the character!
P 01CB 7B  1E                   jr        c,wli_2          !serial disabled!
P 01CD A6  7E  00               cp        TEMP_3,#0        !write line?!
P 01D0 EB  17                   jr        nz,wli_5         !no, absolute.!
P 01D2 56  7C  7F               and       TEMP_1,#%7F      !mask off parity!
P 01D5 A6  7C  0D               cp        TEMP_1,#%0D      !line done?!
P 01D8 EB  0F                   jr        nz,wli_5         !yes.!
P 01DA 00  ED                   dec       wli_len
P 01DC 76  72  04               tm        SER_cfg,#al      !auto line feed?!
P 01DF 6B  0A                   jr        z,wli_2          !disabled!
P 01E1 E6  7C  0A               ld        TEMP_1,#%0A      !output line feed!
P 01E4 D6  020B'                call      ser_output
P 01E7 8B  02                   jr        wli_2
P 01E9 DA  DA           wli_5:  djnz      wli_len,wli_4    !loop!
P 01EB 50  7C           wli_2:  pop       TEMP_1           !original length!
P 01ED 24  ED  7C               sub       TEMP_1,wli_len
P 01F0 D8  7C                   ld        wli_len,TEMP_1   !return output count!
P 01F2 CF                       rcf                        !no error!
P 01F3 AF               wli_1:  ret
P 01F4                  END     ser_wlin
```

```
                      GLOBAL
P 01F4                ser_wabs        PROCEDURE
                      !*****************************************************
                      write absolute

                       Purpose =      To output a character string to serial
                                      line for the length specified.  (Output
                                      is not terminated with the output of
                                      a 'carriage return').

                       Note =         All other details are as for 'ser_wlin'.
                      *****************************************************!
                      ENTRY
P 01F4 E6   7E  01              ld    TEMP_3,#1
P 01F7 8B   C4                  jr    write
P 01F9                END      ser_wabs
P 01F9                ser_wbyt        PROCEDURE
                      !*****************************************************
                      write byte

                       Purpose =      To output a given character to the
                                      serial line. If the character is a
                                      carriage return and auto line feed
                                      is enabled, a line feed will be output
                                      as well.

                       Input =        R12 = character to output

                       Note =         Equivalent to ser_wlin with length = 1.
                      *****************************************************!
                      ENTRY
P 01F9 C9   7C                  ld    TEMP_1,R12
P 01FB D6   020B'               call  ser_output      !output it!
P 01FE 76   72  04              tm    SER_cfg,#al     !auto line feed?!
P 0201 6B   3E                  jr    z,ser_05        !not enabled!
P 0203 A6   EC  0D              cp    R12,#%0D        !char = car. ret?!
P 0206 EB   39                  jr    nz,ser_05       !nope!
P 0208 E6   7C  0A              ld    TEMP_1,#%0A     !output line feed!
                      !fall into ser_output!
P 020B                END       ser_wbyt
```

```
                      GLOBAL          !for PART I!
P 020B                ser_output      PROCEDURE
                      !****************************************************
                       Purpose =      To output one character to the serial
                                      line.

                       Input =        TEMP_1 = character

                       Output =       Carry FLAG = 1 if serial disabled
                                                 = 0 otherwise.

                       Note =         1. If even parity is enabled, the eigth
                                      data bit is modified prior to character
                                      output to SIO.

                                      2. IRQ4 is polled to wait for completion
                                      of character transmission before control
                                      returns to the calling program.
                      ****************************************************!
                      ENTRY
P 020B DF                     scf                     !in case error!
P 020C 76  70  80             tm      SER_flg,#sd     !serial disabled?!
P 020F EB  30                 jr      nz,ser_05       !yes. error!
P 0211 76  72  40             tm      SER_cfg,#ep     !even parity enabled?!
P 0214 6B  1F                 jr      z,ser_o2        !no. just output!
                      !calculate parity!
P 0216 70  7E                 push    TEMP_3
P 0218 E6  7E  07             ld      TEMP_3,#7
P 021B B0  7D                 clr     TEMP_2
P 021D C0  7C         ser_04: rrc     TEMP_1          !character bit to carry!
P 021F 16  7D  00             adc     TEMP_2,#0       !count 1's!
P 0222 00  7E                 dec     TEMP_3
P 0224 EB  F7                 jr      nz,ser_04       !next bit!
P 0226 56  7D  01             and     TEMP_2,#01      !1's count odd/even!
P 0229 56  7C  FE             and     TEMP_1,#%FE
P 022C 44  7D  7C             or      TEMP_1,TEMP_2   !parity bit in D0!
P 022F C0  7C                 rrc     TEMP_1
P 0231 C0  7C                 rrc     TEMP_1          !parity bit in D7!
P 0233 50  7E                 pop     TEMP_3
P 0235 E4  7C  F0     ser_o2: ld      SIO,TEMP_1      !output character!
P 0238 66  FA  10     ser_o1: tcm     IRQ,#%10        !check IRQ4!
P 023B EB  FB                 jr      nz,ser_o1       !wait for complete!
P 023D 56  FA  EF             and     IRQ,#%EF        !clear IRQ4!
P 0240 CF                     rcf                     !all ok!
P 0241 AF             ser_05: ret
P 0242                END     ser_output

                      GLOBAL
P 0242                ser_disable     PROCEDURE
                      !****************************************************
                      disable

                       Purpose =      To disable serial I/O operations.

                       Input =        None.

                       Output =       Serial I/O disabled.
                      ****************************************************!
                      ENTRY
P 0242 8F                     di              !avoid IRQ3 conflict!
P 0243 46  70  80             or      SER_flg,#sd
                                              !set serial disabled!
P 0246 56  F1  FC             and     TMR,#%FC
                                              !disable T0!
P 0249 56  FB  E7             and     IMR,#%E7
                                              !disable IRQ3,4!
P 024C 56  7F  BF             and     P3M_save,#%BF
                                              !P30/7 normal i/o pins!
P 024F E4  7F  F7             ld      P3M,P3M_save
P 0252 9F                     ei              !re-enable interrupts!
P 0253 AF                     ret
P 0254                END     ser_disable
```

```
                      CONSTANT
                       TMP     :=      R13
                       PTR     :=      RR14
                       PTRh    :=      R14
                      GLOBAL
P 0254                tod i   PROCEDURE
                      !****************************************************
                      time of day : initialize

                       Purpose =       To initialize T0 or T1 to function as
                                       a time of day clock.

                       Input =         RR14 = address of parameter list in
                                               program memory:
                                         1 byte = IMR mask for nestable
                                               interrupts
                                         1 byte = # of clock ticks per second
                                         1 byte = counter # : = %F4 => T0
                                                             = %F2 => T1
                                         1 byte = Counter value
                                         1 byte = Prescaler value (unshifted)

                                       TOD_hr, TOD_min, TOD_sec, TOD_tt
                                        initialized to the starting time of
                                        hours, minutes, seconds, and ticks
                                        respectively.

                       Output =        Selected timer is loaded and
                                       enabled; corresponding interrupt
                                       is enabled.
                                       R13, R14, R15 modified.

                       Note =          The cntr and prescaler values provided
                                       are those values which will generate an
                                       interrupt (tick) the designated # of
                                       times per second.

                                       For example:
                                       for XTAL = 8 MHZ, cntr = 250 and
                                       prescaler = 40 yield a .01 sec interval;
                                       the 2nd byte of the parameter list
                                       should = 100 .

                                       For T0 the instruction at %080C or
                                       for T1 the instruction at %080F must
                                       result in a jump to the jump table entry
                                       for 'tod'.

                                       The parameter list is not referenced
                                       following initialization.
                      ****************************************************!
                      ENTRY
P 0254 DC   6C                ld      TMP,#TOD_imr
P 0256 C3   DE                ldci    @TMP,@PTR         !imr mask!
P 0258 C3   DE                ldci    @TMP,@PTR         !ticks/second!
P 025A E6   7B   6C           ld      TEMP_4,#TOD_imr
P 025D 8D   02B2'             jp      pre_ctr           !ctr & prescaler!
P 0260                END     tod_i
```

```
                    GLOBAL
P 0260              tod       PROCEDURE
                    !****************************************************
                    Interrupt service - time of day

                     Purpose =      To update the time of day clock.
                    ****************************************************!
                    ENTRY
P 0260 70 FB                  push    imr              !save entry imr!
P 0262 54 6C FB               and     imr,TOD_imr      !allow nested interrupts
P 0265 9F                     ei                       !enable interrupts!
P 0266 70 FD                  push    rp               !save rp!
P 0268 31 60                  srp     #RAM_TMRr        !point to our set!
P 026A 8E                     inc     rTODtt           !ticks/second!
P 026B A2 8D                  cp      rTODtt,rTODtic   !second complete?!
P 026D EB 13                  jr      ne,tod_ex        !nope.!
P 026F B0 E8                  clr     rTODtt
P 0271 9E                     inc     rTODsec          !seconds!
P 0272 A6 E9 3C               cp      rTODsec,#60      !minute complete?!
P 0275 EB 0B                  jr      ne,tod_ex        !nope.!
P 0277 B0 E9                  clr     rTODsec
P 0279 AE                     inc     rTODmin          !minutes!
P 027A A6 EA 3C               cp      rTODmin,#60      !hour complete?!
P 027D EB 03                  jr      ne,tod_ex        !nope.!
P 027F B0 EA                  clr     rTODmin
P 0281 BE                     inc     rTODhr           !hours!

P 0282 50 FD        tod_ex:   pop     rp               !restore rp!
P 0284 8F                     di                       !disable interrupts!
P 0285 50 FB                  pop     imr              !restore entry imr!
P 0287 BF                     iret
P 0288              END       tod
```

```
                     GLOBAL
P 0288               pulse_i PROCEDURE
                     !*****************************************************
                      Purpose =      To initialize one of the timers
                                     to generate a variable frequency/
                                     variable pulse width output.

                      Input =        RR14 = address of parameter list in
                                             program memory:
                                      1 byte = cntr value for low interval
                                      1 byte = counter # : = %F4 => T0
                                                           = %F2 => T1
                                      1 byte = cntr value for high interval
                                      1 byte = prescaler (unshifted)

                      Output =       Selected timer is loaded and
                                     enabled; corresponding interrupt
                                     is enabled.  P36 is enabled as Tout.
                                     R13, R14, R15 modified.

                      Note =         The parameter list is not referenced
                                     following initialization.

                                     The value of  Prescaler  x  Counter
                                     must be > 26 (=%1A) for proper
                                     operation.
                     *****************************************************!
                     ENTRY
P 0288 DC   65                 LD       TMP,#PLS_2
P 028A C3   DE                 ldci     @TMP,@PTR         !low interval cntr!
P 028C C3   DE                 ldci     @TMP,@PTR         !timer addr!
P 028E C3   DE                 ldci     @TMP,@PTR         !high interval cntr!
P 0290 80   EE                 decw     PTR
P 0292 80   EE                 decw     PTR               !back to flag!
P 0294 56   F1   3F            and      TMR,#%3F          !will be modifying TMR!
P 0297 56   7F   DF            and      P3M_save,#%DF     !P36 = Tout!
P 029A E4   7F   F7            ld       P3M,P3M_save
P 029D E6   7B   01            ld       TEMP_4,#%1        !flag for pre_ctr!
P 02A0 8D   02B2'              jp       pre_ctr           !set up timer!
P 02A3               END       pulse_i


                     GLOBAL
P 02A3               pulse     PROCEDURE
                     !*****************************************************
                      Purpose =      To modify the counter load value
                                     to continue the pulse output generation.

                     *****************************************************!
                     ENTRY
                     !exchange values!
P 02A3 B4   65   67            xor      PLS_1,PLS_2
P 02A6 B4   67   65            xor      PLS_2,PLS_1
P 02A9 B4   65   67            xor      PLS_1,PLS_2
                     !exchange complete!
P 02AC F5   67   66            ld       @PLS_tmr,PLS_1   !load new value!
P 02AF BF                      iret
P 02B0               END       pulse
```

```
                  GLOBAL
P 02B0            delay   PROCEDURE
                  !*****************************************************
                   Purpose =      To generate an interrupt after a
                                  designated amount of time.

                   Input =        RR14 = address of parameter list in
                                         program memory:
                                   1 byte = counter # : = %F4 => T0
                                                      = %F2 => T1
                                   1 byte = Counter value
                                   1 byte = Prescaler value and count mode
                                         (to be loaded as is into
                                          PRE0 or PRE1).

                   Output =       Selected timer is loaded and
                                  enabled; corresponding interrupt
                                  is enabled.
                                  R13, R14, R15 modified.

                   Note =         This routine will initialize the timer
                                  for single-pass or continuous mode
                                  as determined by bit 0 of byte 3 in
                                  the parameter list.
                                  The caller is responsible for provid-
                                  ing the interrupt service routine.

                                  The parameter list is not referenced
                                  following initialization.
                  *****************************************************!
                  ENTRY
P 02B0 B0  7B             clr     TEMP_4
                  !fall into pre_ctr!
P 02B2            END     delay
```

```
                    INTERNAL
P 02B2              pre_ctr PROCEDURE
                    !****************************************************
                     Purpose =       To get counter and prescaler values
                                     from parameter list and modify control
                                     registers appropriately.

                     Input   =       TEMP_4   = 0 => for 'delay'
                                              = 1 => for 'pulse'
                                              = TOD_imr => for 'tod'
                    ****************************************************!
                    ENTRY
P 02B2 C2  DE                 ldc     TMP,@PTR         !T0 or T1!
P 02B4 A0  EE                 incw    PTR
P 02B6 E6  7D  8C             ld      TEMP_2,#%8C      !for TMR!
P 02B9 E6  7E  20             ld      TEMP_3,#%20      !for IMR!
P 02BC A6  ED  F2             cp      TMP,#T1
P 02BF 6B  06                 jr      eq,pre_1         !is for T1!
P 02C1 E6  7D  43             ld      TEMP_2,#%43      !for TMR!
P 02C4 E6  7E  10             ld      TEMP_3,#%10      !for IMR!
P 02C7 C3  DE       pre_1:    ldci    @TMP,@PTR        !init counter!
P 02C9 C2  EE                 ldc     PTRh,@PTR        !prescaler!
P 02CB A6  7B  00             cp      TEMP_4,#0        !shift prescaler?!
P 02CE 6B  12                 jr      eq,pre_2         !no!
P 02D0 DF                     scf                      !internal clock!
P 02D1 10  EE                 rlc     PTRh
P 02D3 DF                     scf                      !continuous mode!
P 02D4 10  EE                 rlc     PTRh
P 02D6 A6  7B  6C             cp      TEMP_4,#TOD_imr
P 02D9 EB  0A                 jr      ne,pre_3         !for 'pulse'!
P 02DB 60  7E                 com     TEMP_3
P 02DD 54  7E  6C             and     TOD_imr,TEMP_3   !insure no self-nesting!
P 02E0 60  7E                 com     TEMP_3
P 02E2 56  7D  0F   pre_2:    and     TEMP_2,#%0F      !no Tout mode mod!
P 02E5 F3  DE       pre_3:    ld      @TMP,PTRh        !init prescaler!
P 02E7 44  7D  F1             or      TMR,TEMP_2       !init tmr mode!
P 02EA 8F                     di
P 02EB 44  7E  FB             or      imr,TEMP_3       !enable interrupt!
P 02EE 9F                     ei
P 02EF AF                     ret
P 02F0              END       pre_ctr
                    END PART_II

   0 errors
 Assembly complete
```

A Comparison of Microcomputer Units

Zilog

Benchmark Report

May 1981

INTRODUCTION

The microcomputer industry has recently developed single-chip microcomputers that incorporate on one chip functions previously performed by peripherals. These microcomputer units (MCUs) are aimed at markets requiring a dedicated computer. This report describes and compares the most powerful MCUs in today's market: the Zilog Z8611, the Intel 8051, and the Motorola MC6801. Table 1 lists facts that should be considered when comparing these MCUs.

Table 1. MCU Comparison

FEATURES	Zilog Z8611	Intel 8051	Motorola MC6801
On-Chip ROM	4Kx8	4Kx8	2Kx8
General-Purpose Registers	124	128	128
Special-Function Registers			
Status/Control	16	16	17
I/O ports	4	4	4
I/O			
Parallel lines	32	32	29
Ports	Four 8-bit	Four 8-bit	Three 8-bit,one 5-bit
Handshake	Hardware on three ports	None	Hardware on one port
Interrupts			
Source	8	5	7
External source	4	2	2
Vector	6	5	7
Priority	48 Programmable orders	2 Programmable orders	Nonprogrammable
Maskable	6	5	6
External Memory	120K bytes	124K bytes	64K bytes
Stack			
Stack pointer	16-Bit	8-Bit	16-Bit
Internal stack	Yes, uses 8-bits	Yes	Yes
External stack	Yes	No	Yes

Table 1. MCU Comparison
(Continued)

FEATURES	Zilog Z8611	Intel 8051	Motorola MC6801
Counter/ Timers			
Counters	Two 8-bit	Two 16-bit or two 8-bit	One 16-bit
Prescalers	Two 6-bit	No prescale with 16-bits; 5-bit prescale with 8-bits	None
Addressing Modes			
Register	Yes	Yes	No
Indirect Register	Yes	Yes	No
Indexed	Yes	Yes	Yes
Direct	Yes	Yes	Yes
Relative	Yes	Yes	Yes
Immediate	Yes	Yes	Yes
Implied	Yes	Yes	Yes
Index Registers	124, Any general-purpose register	1, Uses the accumulator for 8-bit offset	1, Uses 16-bit index register
Serial Communication Interface			
Full duplex UART	Yes	Yes	Yes
Interrupts for transmit and receive	One for each	One for both	One for both
Registers			
Double buffer	Receiver	Receiver	Transmitter/Receiver
Serial Data Rate	62.5K b/s @8 MHz 93.5K b/s @12 MHz	187.5K b/s @12 MHz	62.5K b/s @4 MHz
Speed			
Instruction execution average	2.2 Usec 1.5 Usec @12 MHz	1.5 Usec	3.9 Usec
Longest instruction	4.25 Usec 2.8 Usec @12 MHz	4 Usec	10 Usec
Clock Frequency	8 and 12 MHz	12 MHz	4 MHz
Power Down Mode	Saves first 124 registers	Saves first 128 registers	Saves first 64 registers
Context Switching	Saves PC and flags	Saves PC; programmer must save all registers	Saves PC, PSW, accumulators, and Index register

Table 1. MCU Comparison
(Continued)

FEATURES	Zilog Z8611	Intel 8051	Motorola MC6801
Development	40-Pin Protopack (8613) 64-Pin (8612) 40-Pin ROMless (Z8681)	40-Pin (8751)	40-Pin (68701)
Eprom	4K bytes (2732) 2K bytes (2716)	4K bytes	2K bytes
Availability	Now	TBA	Now

ARCHITECTURAL OVERVIEW

This section examines three chips: the on-chip functions and data areas manipulated by the Zilog, Intel and Motorola MCUs. The three chips have somewhat similar architectures. There are, however, fundamental differences in design criteria. The 8051 and the MC6801 were designed to maintain compatability with older products, whereas the Z8611 design was free from such restrictions and could experiment with new ideas. Because of this, the accumulator architectures of the MC6801 and the 8051 are not as flexible as that of the Z8611, which allows any register to be used as an accumulator.

Memory Spaces

The Z8611 CPU manipulates data in four memory spaces:

- 60K bytes of external data memory
- 60K bytes of external program memory
- 4K bytes of internal program memory (ROM)
- 144-byte register file

The 8051 CPU manipulates data in four memory spaces:

- 64K bytes of external data memory
- 60K bytes of external program memory
- 4K bytes of internal program memory
- 148-byte register file

The MC6801 manipulates data in three memory spaces:

- 62K bytes of external memory
- 2K bytes of internal program memory
- 149-byte register file

On-Chip ROM. All three chips have internal ROM for program memory. The Z8611 and the 8051 have 4K bytes of internal ROM, and the MC6801 has 2K bytes. In some cases, external memory may be required with the MC6801 that is not necessary with the Z8611 or the 8051.

On Chip RAM. All three chips use internal RAM as registers. These registers are divided into two catagories: general-purpose registers and special function registers (SFRs).

The 124 general-purpose registers in the Z8611 are divided into eight groups of 16 registers each. In the first group, the lowest four registers are the I/O port registers. The other registers are general purpose and can be accessed with an 8-bit address or a short 4-bit address. Using the 4-bit address saves bytes and execution time. Four-bit short addresses are discussed later. The general-purpose registers can be used as accumulators, address pointers, or Index registers.

The 128 general-purpose registers in the 8051 are grouped into two sets. The lower 32 bytes are allocated as four 8-register banks, and the upper registers are used for the stack or for general purpose. The registers cannot be used for indexing or as address pointers.

The MC6801 also has a 128-byte, general-purpose register bank, which can be used as a stack or as address pointers, but not as Index registers.

As pointed out in Table 1, any of the Z8611 general-purpose registers can be used for indexing; the MC6801 and the 8051 cannot use registers this way. The Z8611 can use any register as an accumulator; the MC6801 and the 8051 have fixed accumulators. The use of registers as memory pointers is very valuable, and only the Z8611 can use its registers in this way.

The number of general-purpose registers on each chip is comparable. However, because of its flexible design, the Z8611 clearly has a more powerful register architecture.

The Z8611 has 20 special function registers used for status, control, and I/O. These registers include:

- Two registers for a 16-bit Stack Pointer (SPH, SPL)
- One register used as Register Pointer for working registers (RP)
- One register for the status flags (FLAGS)
- One register for interrupt priority (IPR)
- One register for interrupt mask (IMR)
- One register for interrupt request (IRQ)
- Three mode registers for the four ports (P01M, P2M, P3M)
- Serial communications port used like a register (SIO)
- Two counter/timer registers (T0, T1)
- One Timer Mode Register (TMR)
- Two prescaler registers (PRE0, PRE1)
- Four I/O ports accessed as registers (PORT0, PORT1, PORT2, PORT3)

The 8051 also has 20 special function registers used for status, control, and I/O. They include:

- One register for the Stack Pointer (SP)
- Two accumulators (A,B)
- One register for the Program Status Word (PSW)
- Two registers for pointing to data memory (DPH, DPL)
- Four registers that serve as two 16-bit counter/timers (TH0, TH1, TL0, TL1)
- One mode register for the counter/timers (TMOD)
- One control register for the counter/timers (TCON)
- One register for interrupt enable (IEC)
- One register for interrupt priority (IPC)
- One register for serial communications buffer (SBUF)
- One register for serial communications control (SCON)
- Four registers used as the four I/O ports (P0, P1, P2, P3)

The MC6801 has 21 special function registers used for status, control, and I/O. These include:

- One register for RAM/EROM control
- One serial receive register
- One serial transmit register
- One register for serial control and status
- One serial rate and mode register
- One register for status and control of port 3
- One register for status and control of the timer
- Two registers for the 16-bit timer
- Two registers for 16-bit input capture used with timer
- Two registers for 16-bit output compare used with timer
- Four data direction registers associated with the four I/O ports
- Four I/O ports

The special function registers in the three chips seem comparable in number and function. However, upon closer examination, the SFRs of the MC6801 prove less efficient than those of the Z8611. The MC6801 has five registers associated with the I/O ports, whereas the Z8611 uses only three registers for the same functions. The MC6801 uses four registers to perform the serial communication function, whereas the Z8611 uses only one register and part of another.

The 8051 uses two registers for the accumulators; the Z8611 is not limited by this restriction. The 8051 also uses two registers for the serial communication interface, whereas the Z8611 accomplishes the same job with one register. Another two registers in the 8051 are used for data pointers; these are not necessary in the Z8611 since any register can be used as an address pointer.

The Z8611 uses registers more efficiently than either the MC6801 or the 8051. The registers saved by this optimal design are used to perform the functions needed for enhanced interrupt handling and for register pointing with short addresses. The Z8611 also supplies the extra register required for the external stack. These features are not available on the 8051 or the MC6801.

External Memory. All three chips can access external memory. The Z8611 and the 8051 can generate signals used for selecting either program or data memory. The Data Memory strobe (the signal used for selecting data or program memory) gives the Z8611 access to 120K bytes of external memory (60K bytes in both program and data memory). The 8051 can use 124K bytes of external memory (64K bytes of external data memory and 60K bytes of external program memory). The MC6801 can access only 62K bytes of external memory and does not distinguish between program and data memory. Thus, the Z8611 and the 8051 are clearly able to access more external memory than the MC6801.

On-Chip Peripheral Functions

In addition to the CPU and memory spaces, all chips provide an interrupt system and extensive I/O facilities including I/O pins, parallel I/O ports, a bidirectional address/ data bus, and a serial port for I/O expansion.

Interrupts. The Z8611 acknowledges interrupts from eight sources, four are external from pins IRQ_0-IRQ_3, and four are internal from serial-in, serial-out, and the two counter/timers. All interrupts are maskable, and a wide variety of priorities are realized with the Interrupt Mask Register and the Interrupt Priority Registers (see Table 1). All Z8611 interrupts are vectored, with six vectors located in the on-chip ROM. The vectors are fixed locations, two bytes long, that contain the memory address of the service routine.

The 8051 acknowledges interrupts from five sources: two external sources (from INTO and INT1) and three internal sources (one from each of the internal counters and one from the serial I/O port). All interrupts can be disabled individually or globally. Each of the five sources can be assigned one of two priorities: high or low. All 8051 interrupts are vectored. There are five fixed locations in memory, each eight bytes long, allocated to servicing the interrupt.

The MC6801 has one external interrupt, one nonmaskable interrupt, an internal interrupt request, and a software interrupt. The internal interrupts are caused by the serial I/O port, timer overflow, timer output compare, and timer input capture. The priority of each interrupt is preset and cannot be changed. The external interrupt can be masked in the Condition Code register. The MC6801 vectors the interrupts to seven fixed addresses in ROM where the 16-bit address of the service routine is located.

When an interrupt occurs in the 8051, only the Program Counter is saved; the user must save the flags, accumulator, and any registers that the interrupt service routine might affect. The MC6801 saves the Program Counter, acumulators, Index register, and the PSW; the user must save all registers that the interrupt service routine might affect. The Z8611 saves the Program Counter and the Flags register. To save the 16 working registers, only the Register Pointer register need be pushed onto the stack and another set of working registers is used for the service routine. For more detail on working registers and interrupt context switching, see the Z8 Technical Manual (03-3047-02).

With regard to interrupts, the Z8611 is clearly superior. The Z8611 requires only one command to save all the working registers, which greatly increases the efficiency of context switching.

I/O Facilities. The Z8611 has 32 lines dedicated to I/O functions. These lines are grouped into four ports with eight lines per port. The ports can be configured individually under software control to provide input, output, multiplexed address/data lines, timing, and status. Input and output can be serial or parallel, with or without handshake. One port can be configured for serial transmission and four ports can be configured for parallel transmission. With parallel transmission, ports 0, 1, and 2 can transmit data with the handshake provided by port 3.

The 8051 also has 32 I/O lines grouped together into four ports of eight lines each. The ports can be configured under program control for parallel or serial I/O. The ports can also be configured for multiplexed address/data lines, timing, and status. Handshake is provided by user software.

The MC6801 has 29 lines for I/O (three 8-bit ports and one 5-bit port). One port has two lines for handshake. The ports provide all the signals needed to control input and output either serially or in parallel, with or without multiplexed address/data lines. They can be used to interface with external memory.

The main differences in I/O facilities are the number of 8-bit ports and the hardware handshake. The Z8611 and the 8051 have four 8-bit ports, whereas the MC6801 has three 8-bit ports and an additional 5-bit port. The Z8611 has hardware handshake on three ports, the MC6801 has hardware handshake on only one port, and the 8051 has no hardware handshake.

Counter/timers. The Z8611 has two 8-bit counters and two 6-bit programmable prescalers. One prescaler can be driven internally or externally; the other prescaler is driven internally only. Both timers can interrupt the CPU when counting is completed. The counters can operate in one of two modes: they can count down until interrupted, or they can count down, reload the initial value, and start counting down again (continuously). The counters for the Z8611 can be used for measuring time intervals and pulse widths, generating variable pulse widths, counting events, or generating periodic interrupts.

The 8051 has two 16-bit counter/timers for measuring time intervals and pulse widths, generating pulse widths, counting events, and generating periodic interrupts. The counter/timers have several modes of operation. They can be used as 8-bit counters or timers with two 5-bit programmable prescalers. They can also be used as 16-bit counter/timers. Finally, they can be set as 8-bit modulo-n counters with the reload value held in the high byte of the 16-bit register. An interrupt is generated when the counter/timer has completed counting.

The MC6801 has one 16-bit counter which can be used for pulse-width measurement and generation. The counter/timer actually consists of three 16-bit registers and an 8-bit control/status register. The timer has an input capture register, an output compare register, and a free-running counter. All three 16-bit registers can generate interrupts.

Serial Communications Interface. The Z8611 has a programmable serial communication interface. The chip contains a UART for full-duplex, asynchronous, serial receiver/ transmitter operation. The bit rate is controlled by counter/timer 0 and has a maximum bit rate of 93.500 b/s. An interrupt is generated when an assembled character is transferred to the receive buffer. The transmitted character generates a separate interrupt. The receive register is double-buffered. A hardware parity generator and detector are optional.

The 8051 handles serial I/O using one of its parallel ports. The 8051 bit rate is controlled by counter/timer 1 and has a maximum bit rate of 187,500 b/s. The 8051 generates one interrupt for both transmission and receipt. The receive register is double-buffered.

The MC6801 contains a full-duplex, asynchronous, serial communication interface. The bit rate is controlled by a rate register and by the MCU's clock or an external clock. The maximum bit rate is 62,500 b/s. Both the transmit and the receive registers are double-buffered. The MC6801 generates only one interrupt for both transmit and receive operations. No hardware parity generation or detection is available, although it does have automatic detection of framing errors and overrun conditions.

The 8051 and the MC6801 generate only one interrupt for both transmit and receive, whereas the Z8611 has a separate interrupt for each. The ability to generate separate interrupts greatly enhances the use of serial communications, since separate service routines are often required for transmitting and receiving.

Other differences between the Z8611, MC6801, and the 8051 occur in the hardware parity detector, the double-buffering of registers, framing error detectors and overrun conditions. The 8051 has a faster data rate than either the Z8611 or the MC6801. The MC6801 has the advantage of a hardware framing error detector and automatic detection of overrun conditions. The MC6801 also has both its transmit and receive registers double-buffered. The Z8611 has a hardware parity detector. For detection of framing errors and overrun conditions, a simple, low-overhead software check is available that uses only two instructions. See Z8600 Software Framing Error Detection Application Brief (document #617-1881-0004).

INSTRUCTION ARCHITECTURE

The architecture of the Z8611 is designed specifically for microcomputer applications. This fact is manifest in the instruction composition. The arduous task of programming the MC6801 and the 8051 starkly contrasts that of programming the Z8611.

Addressing Modes

The Z8611 and the 8051 both have six addressing modes: Register, Indirect Register, Indexed, Direct, Relative, and Immediate. The MC6801 has five addressing modes: Accumulator, Indexed, Direct, Relative, and Immediate. A quick comparison of these addressing modes reveals the versatility of the Z8611 and the 8051. The addressing modes of the MC6801 have several restrictions, as shown in Table 1. While the 8051 has all the addressing modes of the Z8611, its use of them is restricted. The Z8611 allows many more combinations of addressing modes per instruction, because any of its registers can be used as an accumulator. For example, the instructions to clear, complement, rotate, and swap nibbles are all accumulator oriented in the 8051 and operate on the accumulator only. These same commands in the Z8611 can use any register and access it either directly, with register addressing, or with indirect register addressing.

Indexed Addressing. All three chips differ in their handling of indexing. The Z8611 can use any register for indexing. The 8051 can use only the accumulator as an Index register in conjunction with the data pointer or the Program Counter. The MC6801 has one 16-bit Index register. The address located in the second byte of an instruction is added to the lower byte of the Index register. The carry is added to the upper byte for the complete address. The MC6801 requires the index value to be an immediate value.

The MC6801 has only one 16-bit Index register and an immediate 8-bit value from the second byte of the instruction. Hence, the Indexed mode of the MC6801 is much more restrictive than that of the Z8611. The 8051 must use the accumulator as its only Index register, loading the accumulator with the register address each time a reference is made. Then, using indexing, the data is moved into the accumulator, eradicating the previous index. This forces a stream of data through the accumulator and requires a reload of the index before access can be made again. The Z8611 is clearly superior to both the MC6801 and the 8051 in the flexibility of its indexed addressing mode.

Short and Long Addressing. Short addressing helps to optimize memory space and execution speed. In sample applications of short register addressing, an eight percent decrease in the number of bytes used was recorded.

All three chips have short addressing modes, but the Z8611 has short addressing for both external memory and register memory. The 8051 has short addressing for the lowest 32 registers only.

The Z8611 has two different modes for register addressing. The full-byte address can be used to provide the address, or a 4-bit address can be used with the Register Pointer. To use the working registers, the Register Pointer is set for a particular bank of 16 registers, and then one of the 16 registers is addressed with four bits. Another feature for addressing external memory is the use of a 12-bit address in place of a full 16-bit address. To use the 12-bit address, one port supplies the eight multiplexed address/data lines and another port supplies four bits for the address. The remaining four bits of the second port can be used for I/O. This feature allows access to a maximum of 10K bytes of memory.

The 8051 uses short addresses by organizing its lowest 32 registers into four banks. The bank select is located in a 2-bit field in the PSW, with three bits addressing the register in the bank.

The MC6801 uses extended addressing for addressing external memory. With a special, nonmultiplexed expansion mode, 256 bytes of external memory can be accessed without the need for an external address latch. The MC6801 uses one 8-bit port for the address and another port for the data.

Stacks

The Z8611 and the MC6801 provide for external stacks, which require a 16-bit Stack Pointer. Internal stacks use only an 8-bit Stack Pointer. The 8051 uses only a limited internal stack requiring an 8-bit Stack Pointer. Using an external stack saves the internal RAM registers for general-purpose use.

Summary

The stack structure of the Z8611 and the MC6801 is better than that of the 8051. In most applications, the 8051 is more flexible and easier to program than the MC6801. The Z8611 is easier to use than either the 8051 or the MC6801 because of its register flexibility and its numerous combinations of addressing modes. The 8051 features a unique 4μn multiply and divide command. The MC6801 has a multiply, but it takes 10μs to perform it.

In summary, the Z8611 has the most flexible addressing modes, the most advanced indexing capabilities, and superior space- and time-saving abilities with respect to short addressing.

DEVELOPMENT SUPPORT

All three vendors provide development support for their products. This section discusses the different support features, including development chips, software, and modules.

Chips

Zilog offers an entire family of microcomputer chips for product development and final product. The Z8611 is a single-chip microcomputer with 4K bytes of mask-programmed ROM. For development, two other chips are offered. The Z8612 is a 64-pin, development version with full interface to external memory. The Z8613 is a prototype version that uses a functional, piggy-back, EPROM protopak. The Z8613 can use either a 4K EPROM (2732) or a 2K EPROM (2716). Zilog also offers a ROMless version in a 40-pin package that has all the features of the Z8611 except on-board ROM (Z8681).

Intel offers a similar line of development chips with its 8051 family. The 8031 has no internal ROM and the 8751 has 4K of internal EPROM.

Motorola offers the MC6801, MC6803, MC6803NR, and MC68701. These are all similar except the MC68701 has 2K bytes of EPROM and the MC6801 has 2K bytes of ROM. The MC6803 has no internal ROM and the MC6803NR has neither ROM nor RAM on board.

The Z8613 and the MC68701 are both available now, but the 8751 is still unavailable (as of April 1981).

Software

Development software includes assemblers, and conversion programs. All manufacturers offer some or all of these features.

Since the MC6801 is compatible with the 6800, there is no need for a new assembler. The Z8611 and the 8051 both offer assemblers for their products. The Zilog PLZ/ASM assembler generates relocatable and absolute object code. PLZ/ASM also supports high-level control and data statements, such as IF... THEN...ELSE. Intel offers an absolute macroassembler, ASM51, with their product. They also offer a program for converting 8048 code to 8051 code.

Modules

The Z8611 development module has two 64-pin development versions of the 40-pin, ROM-masked Z8611. Intel offers the EM-51 emulation board, which contains a modified 8051 and PROM or EPROM in place of memory. Motorola has the MEX6801EVM evaluation board for program development. All three development boards are available now.

ADDITIONAL FEATURES

Additional features include Power Down mode, self-testing, and family-compatibility.

Power Down Mode

All three microcomputers offer a Power Down mode. The Z8611 and the 8051 save all of their registers with an auxilary power supply. The MC6801 uses an auxiliary power supply to save only the first 64 bytes of its register file.

The Z8611 uses one of the crystal input pins for the external power supply to power the registers in Power Down mode. Since the XTAL2 input must be used, an external clock generator is necessary and is input via XTAL1. The 8051 and the MC6801 both have an input reserved for this function. The MC6801 uses the V_{cc} standby pin, and the 8051 uses the V_{pd} pin.

Family Compatibility

Another strength of the Z8611 is its expansion bus, which is completely compatible with the Zilog Z-BUS^{TM}. This means that all Z-BUS peripherals can be used directly with the Z8611.

The MC6801 is fully compatible with all MC6800 family products. The 8051 is software compatible with the older 8048 series and all others in that family.

BENCHMARKS

The following benchmark tests were used in this report to compare the Z8611, 8051, and MC6801:

- o Generate CRC check for 16-bit word.
- o Search for a character in a block of memory.
- o Execute a computed GOTO - jump to one of eight locations depending on which of the eight bits is set.
- o Shift a 16-word five places to the right.
- o Move a 64-byte block of data from external memory to the register file.
- o Toggle a single bit on a port.
- o Measure the subroutine overhead time.

These programs were selected because of their importance in microcomputer applications. Algorithms that reflect a unique function or feature were excluded for the sake of comparison. Although programs can be optimized for a particular chip and for a particular attribute (code density or speed) these programs were not.

The figures cited in this text are taken directly from the vendor's documentation. Therefore, the cycles given below for the MC6801 and the 8051 are in machine cycles and the Z8611 figures are given in clock cycles. The Z8611 clock cycles should be divided by six to give the instruction time in microseconds. The 8051 and MC6801 machine cycle is 1µs, and the Z8611 clock cycle is .166µs at 12 MHz.

Because of the lack of availability of the MC6801 and the 8051, the benchmark programs listed here have not yet been run. When these products are readily available, the programs will be run and later editions of this document will reflect any changes in the findings.

Program Listings

CRC Generation

```
8051                                Machine
                                    Cycles     Bytes
           MOV   INDEX, #8           1           2
    LOOP:  MOV   A, DATA             1           2
           XRL   A, HCHECK           1           2
           RLC   A                   1           1
           MOV   A, LCHECK           1           2
           XRL   A, LPOLY            1           2
           RLC   A                   1           1
           MOV   LCHECK, A           1           2
           MOV   A, HCHECK           1           2
           XRL   A, HPOLY            1           2
           RLC   A                   1           1
           MOV   HCHECK, A           1           2
           CLR   C                   1           1
           MOV   A, DATA             1           2
           RLC   A                   1           1
           MOV   DATA, A             1           2
           DJNZ  INDEX, LOOP         2           3
           RET                       2           1
           N = 3+17X8 = 139 cycles
             @12 MHz = 139µs
             Instructions = 18
             Bytes = 31
```

```
MC6801                              Machine
                                    Cycles     Bytes
           LDAA  #$08                2           2
    LOOP:  STAA  COUNT               3           2
           LDAA  HCHECK              3           2
           EORA  DATA                3           2
           ROLA                      2           1
           LDAD  POLY                4           2
           EORA  HCHECK              3           2
           EORB  LCHECK              3           2
           ROLB                      2           1
           ROLA                      2           1
           STAD  LCHECK              4           2
           ASL   DATA                6           3
           DEC   COUNT               6           3
           BNE   LOOP                4           2
           RTS                       5           1
           N = 45X8+7 = 367 cycles
             @4 MHz = 367µs
             Instructions = 15
             Bytes = 28
```

```
Z8611                               Clock
                                    Cycles     Bytes
           LD    INDEX, #8           6           2
    LOOP:  LD    R6, DATA            6           2
           XOR   R6, HCHECK          6           2
           RLC   R6                  6           2
           XOR   LCHECK, LPOLY       6           2
           RLC   LCHECK              6           2
           XOR   HCHECK, HPOLY       6           2
           RLC   HCHECK              6           2
           RCF                       6           1
           RLC   DATA                6           2
           DJNZ  INDEX, LOOP         12 or 10    2
           RET                       14          1
           N = 20+66X7+64 = 546 cycles
             @12 MHz = 91 µs
             Instructions = 12
             Bytes = 22
```

Character Search Through Block of 40 Bytes

```
8051                                      Machine
                                           Cycles     Bytes
        MOV    INDEX, #41                   1          2
        MOV    DPTR, #TABLE                 2          3
LOOP1:  DJNZ   INDEX, LOOP 2                2          2
        SJMP   OUT                          2          2
LOOP2:  MOV    A, INDEX                     1          2
        MOVC   A, @A+DPTR                   2          1
        CJNE   A, CHARAC, LOOP1             2          3
  OUT:
        N = 3+39X7+4 = 280 cycles
          @12 MHz = 280µs
          Instructions = 7
          Bytes = 15
```

```
MC6801                                   Machine
                                          Cycles      Bytes
       LDAB   #$40                          2          2
       LDAA   #CHARAC                       2          2
       LDX    #TABLE                        3          3
LOOP:  CMPA   $0, X                         4          2
       BEQ    OUT                           4          2
       INX                                  3          1
       DECB                                 2          1
       BNE    LOOP                          4          2
 OUT:  -
       -
       -
       N = 7+40X17 = 687 cycles
         @4 MHz = 687µs
         Instructions = 8
         Bytes = 15
```

```
Z8611                                       Clock
                                            Cycles      Bytes
       LD     INDEX, #40                     6           2
LOOP:  LD     DATA, TABLE (INDEX)           10           3
       CP     DATA, CHARAC                   6           2
       JR     Z, OUT                        12 or 10     2
       DJNZ   INDEX, LOOP                   12 or 10     2
 OUT:  -
       -
       N = 6+38X40 = 1524 cycles
         @12 MHz = 254µs
         Instructions = 5
         Bytes = 11
```

Shift 16-Bit Word to Right 5-Bits

```
8051                                     Machine
                                         Cycles      Bytes
       MOV    INDEX  #5                    1           2
LOOP:  CLR    C                            1           1
       MOV    A, WORD + 1                  1           2
       RRC    A                            1           1
       MOV    WORD + 1, A                  1           2
       MOV    A, WORK                      1           2
       RRC    A                            1           1
       MOV    WORD, A                      1           2
       DJNZ   INDEX, LOOP                  2           2
       N = 1+9X5 = 46 Cycles
         @12 MHz = 46µs
         Instructions = 9
         Bytes = 15
```

```
MC6801                                   Machine
                                         Cycles      Bytes
       LDX    #5                           6           3
       LDAD   WORK                         4           2
LOOP:  LSRD                                3           1
       DEX                                 3           1
       BNE    LOOP                         4           2
       STAD   WORD                         4           2
       N = 10X5+11 = 61 Cycles
         @4 MHz = 61 µs
         Instructions = 6
         Bytes = 11
```

```
Z8611                                    Clock
                                         Cycles      Bytes
       LD     INDEX, #5                    6           2
LOOP:  CCF                                 6           1
       RRC    WORD + 1                     6           2
       RRC    WORD                         6           2
       DJNZ   INDEX, LOOP                 12 or 10     2
       N = 6+4X30+28 = 154 Cycles
         @12 MHz = 26 µs
         Instructions = 5
         Bytes = 9
```

Computed GOTO

```
8051                                 Machine
                                      Cycles     Bytes
        MOV   INDEX, #40                1          2
  LOOP: MOV   A, DATA                   1          2
        RLC   A                         1          1
        JC    OUT                       2          2
        MOV   A, INDEX                  1          1
        ADD   A, #3                     1          2
        MOV   INDEX, A                  1          1
        SJMP  LOOP                      2          2
   OUT: MOV   DPTR, #TABLE              2          3
        MOV   A, INDEX                  1          1
        JMP   @A+DPTR                   2          1
TABLE:  LCALL ADDR1                                3
        -
        -
        LCALL ADDRN                     2
        N = 1+9X7+11 = 75 Cycles
          @12 MHz = 75μs
          Instructions = 12
          Bytes = 21
```

```
MC6801                               Machine
                                      Cycles    Bytes
       LDAB   #2                        2         2
       LDX    TABLE                     3         3
LOOP:  RORA                             2         1
       BCS    OUT                       4         2
       ABX                              3         1
       JMP    LOOP                      3         2
  OUT: LDX    0, X                      5         3
       JMP    0, X                      4         3
       N = 8X12+14 = 110 Cycles
         @4 MHz = 110μs
         Instructions = 8
         Bytes = 17
```

```
Z8611                                    Clock
                                         Cycles     Bytes
      CLR  INDEX                           6          2
LOOP: INC  INDEX                           6          1
      RLC  DATA                            6          2
      JR   NC, LOOP                        12 or 10   2
      LD   ADDR,TABLE 1, (INDEX)           10         3
      LD   ADDR+1,TABLE 2, (INDEX)         10         3
      JP   @ADDR                           12         2
      N = 6+24X7+54 = 228 Cycles
        @12 MHz = 38μs
        Instructions = 7
        Bytes = 15
```

Move 64-Byte Block

```
8051                                     Machine
                                          Cycles     Bytes
        MOV   INDEX, #COUNT                 1          2
  LOOP: MOV   DPTR, #ADDR1                  2          3
        MOVX  A, @DPTR                      2          1
        INC   #ADDR1                        1          1
        MOV   @ADDR2,A                      1          1
        INC   ADDR2                         1          1
        DJNZ  INDEX, LOOP                   2          1
        N = 1+9X64 = 577 Cycles
          @12 MHz = 577μs
          Instructions = 7
          Bytes = 10
```

```
MC6801                                   Machine
                                          Cycles     Bytes
        LDAB  #COUNT                        2          2
  LOOP: LDX   ADDR1                         4          3
        LDAA  0, X                          4          2
        INX                                 3          1
        STAA  ADDR1                         4          2
        LDX   ADDR2                         4          3
        STAA  0, X                          4          2
        INX                                 3          1
        STX   ADDR2                         4          2
        DECB                                2          1
        BNE   LOOP                          4          2
        N = 64X36+2 = 2306 Cycles
          @4 MHz =2306 μs
          Instructions = 11
          Bytes = 21
```

```
Z8611                                    Clock
                                          Cycles     Bytes
        LD    INDEX, #COUNT                 6          2
  LOOP: LDEI  @ADDR2, @ADDR1                18         2
        DJNZ  INDEX, LOOP                   12 or 10   2
        N = 6+63X30+28 = 1924 Cycles
          @12 MHz = 321μs
          Instructions = 3
          Bytes = 6
```

Toggle a Port Bit

8051

```
                               Machine
                               Cycles    Bytes
    XRL    PO, #YY               2         3
    N = 2 Cycles
      @12 MHz = 2μs
      Instructions = 1
      Bytes = 3
```

MC6801

```
                               Machine
                               Cycles    Bytes
    LDAA   PORTO                 3         2
    EORA   #YY                   2         2
    STAA   PORTO                 3         2
    N = 8 Cycles
      @4 MHz = 8μs
      Instructions = 3
      Bytes = 6
```

Z8611

```
                               Clock
                               Cycles    Bytes
    XOR    PORTO, #YY           10         2
    N = 10 Cycles
      @12-MHz = 1.7μs
      Instructions = 1
      Byte = 2
```

Subroutine Call/Return Overhead

8051

```
                                  Machine
                                  Cycles    Bytes
         LCALL  SUBR                2         3
         -
         -
         -
SUBR:    -
         -
         -
         RET                        2         1
         N = 4 Cycles
           @12 MHz = 4μs
           Instructions = 2
           Bytes = 4
```

MC6801

```
                                  Machine
                                  Cycles    Bytes
         JSR    SUBR                9         2
         -
         -
         -
SUBR:    -
         -
         -
         RTS                        5         1
         N = 14 Cycles
           @4 MHz = 14μs
           Instructions = 2
           Bytes = 3
```

Z8611

```
                                  Clock
                                  Cycles    Bytes
         CALL @SUBR                20         2
         -
         -
         -
SUBR:    -
         -
         -
         RET                       14         1
         N = 34 Cycles
           @12 MHz = 5.7μs
           Instructions = 2
           Bytes = 3
```

Results

Table 2 summarizes the results of this comparison. The relative performance column lists the speeds of the MC6801 and 8051 divided by the Z8611 speeds (12 MHz). The overall performance averages the separate relative performances. The higher the number, the faster the Z8611 as compared to the MC6801 and the 8051.

The relative performance figures show that the Z8611 runs 50 percent faster than the 8051 and 250 percent faster than the MC6801. Although speed is not necessarily the most important criterion for selecting a particular product, the Z8611 proves to be an undeniably superior product when speed is added to the advantages of programming ease, code density, and flexibility.

Table 2. Benchmark Program Results

Benchmark Test	MC6801 (4 MHz)		8051 (12 MHz)		Z8 (8 MHz)		Z8 (12 MHz)		Relative Performance	
	cycles	time	cycles	time	cycles	time	cycles	time	MC6801	8051
CRC Generation	367	367	139	139	546	137	546	91	4.03	1.53
Character Search	687	687	280	280	1524	382	1524	254	2.70	1.10
Computed GOTO	110	110	75	75	228	57	228	38	2.89	1.97
Shift Right 5 Bits	61	61	46	46	154	38	154	26	2.35	1.78
Move 64-byte block	2306	2306	577	577	1924	481	1924	321	7.18	1.80
Subroutine Overhead	14	14	4	4	34	8.5	34	5.7	2.46	0.70
Toggle a Port Bit	8	8	2	2	10	2.5	10	1.7	4.71	1.18
					Overall Performance				3.76	1.44

Note: All times are given in microseconds.

Table 3. Byte/Instruction/Time Comparison

	Bytes			Instructions			Time (microseconds)		
	MC6801	8051	Z8611	MC6801	8051	Z8611	MC6801	8051	Z8611
CRC Generation	28	31	22	15	18	12	367	139	91
Character Search	15	15	11	8	7	5	687	280	254
Shift Right 5 Bits	11	15	9	6	9	5	61	46	26
Computed GOTO	17	21	15	8	12	7	110	75	38
Move Block	21	10	6	11	7	3	2306	577	321
Toggle Port Bit	6	3	2	3	1	1	8	2	1.7
Subroutine Call	3	4	3	2	2	2	14	4	5.7

SUMMARY

The hardware of the three chips compared is very similar. The Z8611, however, has several advantages, the most important of which is its interrupt structure. It is more advanced than the interrupt structures of both the 8051 and the MC6801. Other advantages of the Z8611 over either the MC6801 or the 8051 include I/O facilities with parity detection and hardware handshake and a larger amount of internal ROM (the MC6801 has only 2K bytes).

Substantial differences are apparent with regard to software architecture. The addressing modes of the Z8611 are more flexible than those of either the MC6801 or the 8051. The Z8611 can use byte-saving addressing with working registers, and it has short external addresses for saving I/O lines. It can also provide for an external stack. The register architecture (as opposed to the accumulator architecture) of the Z8611 saves execution time and enhances programming speed by reducing the byte count.

The Z8611 microcomputer stands out as the most powerful chip of the three, and concurrently, it is the easiest to program and configure.

Z86 Interrupt Request Register

Zilog

Application Brief

October 1980

The Interrupt Request Register (IRQ, R250) stores requests from the six possible interrupt sources (IRQ^0-IRQ^5) in the Z8600 series microcomputer. In addition to other functions, a hardware reset to the Z8600 disables the IRQ register and resets its request bits. Before the IRQ will register requests, it must first be enabled by executing an Enable Interrupts (EI) instruction. Setting the Enable Interrupt bit in the Interrupt Mask Register (IMR, R251) is not an equivalent operation for this purpose; to enable the IRQ, an EI instruction is required. The function of this EI instruction is distinct from its task of globally enabling the interrupt system. Even in a polled system where IRQ bits are tested in software, it is necessary to execute the EI.

The designer must ensure that unexpected and undesirable interrupt requests will not occur after the EI is executed. One method of doing this is to reset all interrupt enable bits in the IMR for levels that are possible interrupt sources; the EI instruction may then be safely executed. Once EI is executed, the program may immediately execute a Disable Interrupts (DI) instruction. The code necessary to perform these operations is as follows:

```
RESET:  LD   IMR, #%XX   !SET INTERRUPT MASK!
        EI               !ENABLE GLOBAL INTER-
                          RUPT, ENABLE IRQ!
```

where XX has a Ø in each bit position corresponding to the interrupt level to be disabled. If all IMR bits are to be reset, a CLR IMR instruction may be used.

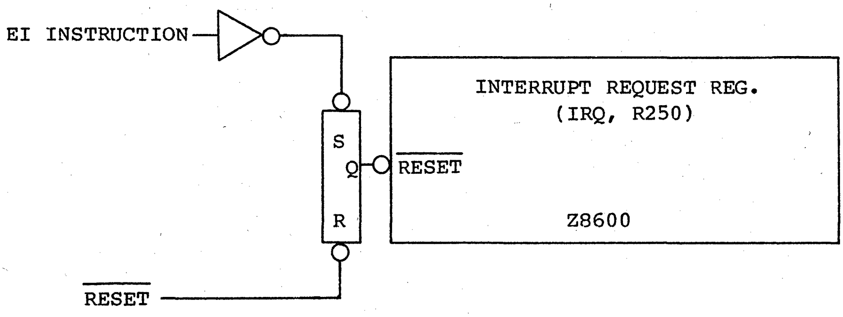


Figure 1 - IRQ Reset Functional Logic Diagram

Zilog

Z8 Family Software Framing Error Detection

Application Brief

October 1980

INTRODUCTION

The Zilog Z8600 UART microcomputer is a high-performance, single-chip device that incorporates on-chip ROM, RAM, parallel I/O, serial I/O, and a baud rate generator. The UART is capable of full-duplex, asynchronous serial communication at nine standard software-selectable baud rates from 110 to 19.2K baud; other nonstandard rates can also be obtained under software control. Odd parity generation and checking can also be selected.

Three possible error conditions can occur during reception of serial data: framing error, parity error, and overrun error. A framing error condition occurs when a stop bit is not received at the proper time (Figure 1). This can result from noise in the data channel, causing erroneous detection of the previous start bit or lack of detection of a properly transmitted stop bit. The Z8600 UART does not incorporate hardware framing error detection but does facilitate a simple, low-overhead software detection method.

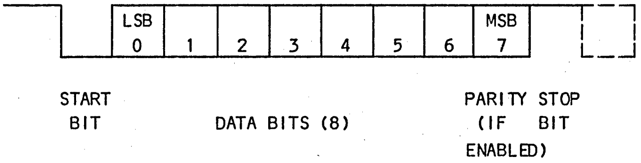


Fig. 1 - Asynchronous Data Format

METHOD

In the middle of the stop bit time, the Z8600 UART automatically posts a serial input interrupt request on IRQ$_3$. The serial input can also be tested by reading Port 3 bit 0 (P3$_0$) as shown in Figure 2. Thus, within the interrupt service routine or polling loop, it is only necessary to test P3$_0$ in order to identify a framing error. If P3$_0$ is Low when IRQ$_3$ goes High, a framing error condition exists and the following code is used to test this:

```
TM P3, #%01     ! TEST FOR P30 = 1 !
JR Z, FERR      ! ELSE FRAMING ERROR !
```

The execution time of this framing error test is only 5.5 µs at 8 MHz. In the worst case (19.2K baud), this would result in 1% overhead. Only five program bytes are required.

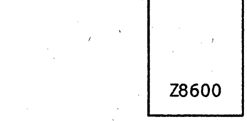


Fig. 2 - Z8600 Serial Input Connection

Z8 is a trademark of Zilog, Inc.

CONCLUSION

While the Z8600 UART does not incorporate hardware framing error detection, this feature can be implemented in software with a maximum penalty of 1% at 19.2K baud using no additional hardware and only five bytes of program memory.

Reprinted with permission of Synertek, Inc.

Zilog Technical Manual

November 1984

Z8® Microcomputer

Table Of Contents

Chapter 1. Z8 Family Overview

1

Chapter 2. Architectural Overview

2

Chapter 3. Address Spaces

3

Table Of Contents (Continued)

Chapter 4. Address Modes

4

Chapter 5. Instruction Set

5

Chapter 6. External Interface (Z8601, Z8611)

6

Chapter 7. External Interface (Z8681, Z8682)

7

Chapter 8. Reset and Clock

8

Chapter 9. I/O Ports

9

Chapter 10. Interrupts

Chapter 11. Counter/Timers

11

Chapter 12. Serial I/O

12

Appendix A. Pin Descriptions and Functions

Chapter 1
Z8 Family Overview

1.1 INTRODUCTION

This chapter provides an overview of the architecture and features of the Z8 Family of products, with particular emphasis on those features that set this microcomputer apart from earlier microcomputers. Detailed information about the architecture, address spaces and modes, instruction set, external interface, timing, input/output operations, and interrupts can be found in subsequent chapters of this manual.

1.2 FEATURES

The Z8 microcomputer introduces a new level of sophistication to single-chip architecture. Compared to earlier single-chip microcomputers, the Z8 offers faster execution; more efficient use of memory; more sophisticated interrupt, input/output and bit-manipulation capabilities; and easier system expansion.

Z8 products offer the standard on-chip functions of earlier microcomputers, including:

- 2K or 4K bytes of ROM
- 144 8-bit registers
- 32 lines of programmable I/O
- Clock oscillator
- Arithmetic logic unit
- Parallel and serial ports

Beyond these basic features, the Z8 Family offers such advanced characteristics as:

- Two counter/timers
- Six vectored interrupts
- UART for serial I/O communication
- Stack functions
- Power-down option
- TTL compatibility
- Optimized instruction set
- BASIC/Debug interpreter

All members of the Z8 Family are variations of the basic Z8 microcomputer, the Z8601/11. The Z8 Family includes a development device (Z8612), a ROMless device (Z8681/82), BASIC/Debug Interpreter (Z8671), a Protopack emulator (Z8603/13), as well as the basic microcomputer. These products offer all the parts and development tools necessary for systems development (both hardware and software prototyping), field trials (pre-production) and full production. For prototyping and preproduction, or where code flexibility is important, the Z8603/13 Protopack, 2K and 4K EPROM-based parts are the most appropriate. The ROM-based Z8601/11 microcomputers are used in high-volume production applications after the software has been perfected. For ROMless applications, two versions of the Z8 microcomputer are available: the 40-pin Z8681/82 and the 64-pin Z8612. In addition, there is a military version of the Z8611 4K ROM device, available in both 40-pin ceramic and 44-pin leadless chip carrier packages.

The Z8671 MCU is a complete microcomputer preprogrammed with a BASIC/Debug Interpreter. This device, operating with both external ROM or RAM and on-chip memory registers, is suitable for most industrial control applications, or whenever fast and efficient program development is necessary.

The Z8 microcomputer is well-suited for dedicated control applications in real-time mode. Since speed is a key consideration in such applications, the Z8 Family is available in both 8 and 12 MHz versions, supported by either of two development modules: the Development Module ($\overline{\text{DM}}$) or the Z-SCAN 8. The Z-SCAN module provides (ICE) in-circuit emulation capability.

1.2.1 Instruction Set

The Z8 instruction set, consisting of 43 basic instructions, is optimized for high-code density and reduced execution time. The 47 instruction types and six addressing modes--together with the ability to operate on bits, 4-bit words, BCD digits, 8-bit bytes, and 16-bit words--make for a code-efficient, flexible microcomputer.

1.2.2 Architecture

Z8 architecture offers more flexibility and performance than previous A/B accumulator designs. All 128 general-purpose registers, including dedicated I/O port registers, can be used as accumulators. This eliminates the bottleneck commonly found in A/B devices, particularly in high-speed applications such as disk drives, printers and terminals. In addition, the registers can be used as address pointers for indirect addressing, as index registers or for implementing an on-chip stack. Speed of execution and smooth programming are supported by a "working register area"--short 4-bit register addresses.

Table 1-1 lists the basic characteristics of the members of the Z8 Family. As shown, the major differences between the products are in their physical packaging and the manner in which address space is handled. An overall description for each Z8 type is given in the following sections. Variations within each group are specified where applicable.

1.3 MICROCOMPUTERS (Z8601/Z8611)

The Z8 can be a stand-alone microcomputer with either 2K bytes (Z8601) or 4K bytes (Z8611) of internal ROM, a traditional microprocessor that can manage up to 124K bytes (Z8601) or 120K bytes (Z8611) of external memory, or a parallel processing element in a system with other processors and peripheral controllers linked by a Z-BUS. In all configurations, a large number of device pins are available for I/O. Key features of the Z8601/11 microcomputer include:

o **ROM 2K-byte (Z8601) or 4K-byte (Z8611) Program Memory.** This ROM is mask-programmed during production with user-provided programs.

o **144-byte RAM Register File.** The internal register organization of the Z8 microcomputer centers around a 144-byte file composed of 124 general-purpose registers, 16 status and control registers, and 4 I/O port registers. Either an 8-bit or a 4-bit address mode can be used to access the register file. When the 4-bit mode is used, the register file is divided into 9 groups of 16 working registers each. A Register Pointer uses short-format instructions to quickly access any one of the nine groups. Use of the 4-bit addressing mode decreases access time and improves throughput.

o **Programmable Counter/Timers.** Two 8-bit counter/timer circuits are provided, each driven by its own prescaler. Both the counter/timers and their prescaler circuits are programmable.

o **UART (Universal Asynchronous Receiver Transmitter).** A full-duplex UART is provided to control serial data communications. One of the on-chip counter/timer circuits provides the required bit rate input to enable the UART to operate at a maximum data transfer rate of 93.75K bits per second at a crystal frequency of 12 MHz.

o **I/O Lines/Ports.** The Z8 microcomputer provides 32 input/output lines, arranged as 4 8-bit ports. Under software control, the I/O ports (Ports 0, 1, 2, 3) can be programmed as input, output, or additional address lines. The I/O ports can also be programmed to provide timing, status signals, interrupt inputs and serial or parallel I/O (with or without handshake).

o **Vectored Interrupts.** The Z8 MPU permits the use of six different interrupts from any of eight different sources. Four Port 3 lines ($P3_0$-$P3_3$), serial input pin ($P3_0$), the serial output pin ($P3_7$) and both counter/timer circuits may be interrupt sources. All interrupts are vectored and are both maskable and prioritized.

o **Oscillator Circuit.** An oscillator circuit that can be driven from an external clock or crystal is provided on the Z8 microcomputer. The oscillator will accept an input frequency of up to 12 MHz on the XTAL1 and XTAL2 pins provided.

o **Optional Power-Down Feature.** This option permits normal input power to be removed from the chip without affecting the contents of the register file. The power-down function requires an external battery backup system.

Pin functions and descriptions for the Z8601/11 microcomputer can be found in Chapter 6.

1.4 DEVELOPMENT DEVICE (Z8612)

A development device allows users to prototype a system with an actual hardware device and to develop the code that is eventually mask-programmed into the on-chip ROM of the Z8601 or Z8611 microcomputer. Development devices are also useful in applications where production volume does not justify the expense of a ROM system. The Z8612 development device is identical to its equivalent microcomputer, the Z8611, with the following exceptions:

- No internal ROM is provided, so that code is developed in an off-chip memory.
- The normally internal ROM address and data lines are buffered and brought out to external pins to interface with the external memory.
- Control lines are added to interface with external program memory.
- The device package is enlarged in order to accommodate the new control, address, and data lines.

Pin functions and descriptions for the development device can be found in the Appendix.

Table 1-1. Z8 Family of Products

Product	Part Number	ROM Capacity (Bytes)	Programmable I/O Pins	Dedicated I/O Pins	PCB Footprint	Comments
2K ROM	Z8601	2K	32, 4 ports	8 Power, Control	40 Pin	Masked ROM part, used primarily for high volume production.
2K Protopack	Z8603	0	32, 4 ports	8 Power, Control plus 24 EPROM	40 Pin	Piggyback part used where program flexibility is required (prototyping).
4K ROM	Z8611	4K	32, 4 ports	8 Power, Control	40 Pin	Masked ROM part, used primarily for high volume production.
4K Development part	Z8612	0	32, 4 ports	8 Power, Control plus 24 external memory	64 Pin	ROMless part used primarily in development systems.
4K Protopack	Z8613	0	32, 4 ports	8 Power, Control plus 24 EPROM	40 Pin	Piggyback part used where program flexibility is required (prototyping).
BASIC/ Debug	Z8671	2K	32, 4 ports	8 Power, Control	40 Pin	BASIC/Debug part used in low volume applications.
ROMless	Z8681/82	0	24, 3 ports	8 Power, Control plus 8 external memory	40 Pin	Low cost ROMless production part with reduced I/O. Program memory is external.

1.5 PROTOPACK EMULATOR (Z8603/13)

The Protopack emulator devices, Z8603 and Z8613, are ROMless versions of their equivalent microcomputers (Z8601 and Z8611, respectively). The emulators differ from development devices in two ways: they use the same pinout as the microcomputers, and an external ROM or EPROM can be plugged into the top of the package. The emulator package allows for flexibility of application, since it can be used in either prototype or final pc boards, yet still allows for program development.

When the final program is developed, it can be mask-programmed into the Z8601/11 which then replaces the emulator. The emulator is also useful in small volume applications where the cost of mask-programming is prohibitive or where program flexibility is desired.

Physical description for the Protopack emulator is found in the Appendix.

1.6 BASIC/DEBUG INTERPRETER (Z8671)

The Z8671 MCU is a complete microcomputer preprogrammed with a BASIC/Debug interpreter. BASIC/ Debug can directly address the Z8671's internal registers and all external memory. It can quickly examine and modify any external memory location or I/O port, and can call machine language subroutines to increase execution speed.

The Z8671 MCU has a combination of software and hardware that is ideal for most industrial control applications. Along with the functions mentioned above, this microcomputer has a self-contained line editor for interactive debugging which further speeds program development. In addition the BASIC/Debug Interpreter allows program execution on power-up or reset, without operator intervention.

Two kinds of memory exist in the Z8671 device: on-chip registers and external ROM or RAM. The BASIC/Debug interpreter is located in the 2K bytes of on-chip ROM. Maximum addressing capability is 62K bytes of external program memory and 62K bytes of data memory. In addition, 32 I/O lines, a 144-byte register file, on-board UART and two counter/timers are provided.

Pin descriptions and functions are the same as those for the Z8601/11 basic microcomputer (Chapter 6).

1.7 ROMLESS MICROCOMPUTER (Z8681/82)

The Z8681 and Z8682 ROMless microcomputers provide virtually all of the functions of the standard Z8 microcomputer without the need to mask-program on-chip ROM. This microcomputer is similar to the Z8601 version except that there is no on-chip program memory. Unlike the ROMless development and Protopack devices the Z8681/82 has no additional address or address control lines nor does it carry a plug-in piggyback memory module. Use of external memory rather than internal ROM enables this Z8 device to be used in low volume applications or where code flexibility is required. The use of Ports 0 and 1 to interface external memory leaves 16 to 24 lines for I/O.

Since Port 1 is dedicated as an 8-bit multiplexed Address/Data bus, and Port 0 lines can be programmed as address bits, the resulting 16-bit addresses can directly address up to 64K bytes of memory for the Z8681 and 62K bytes for the Z8682. (The Z8682 MCU cannot address the lower 2K bytes of memory).

The address capability of the Z8681/82 can be doubled by programming output $P3_4$ of Port 3 as Data Memory ($\overline{DM}$) select signal. The two states of this signal can be used with the 16-bit addresses to identify two separate external address spaces, thus increasing external address space to 128K bytes for the Z8681 and 124K bytes for the Z8682.

Pin functions and descriptions for the Z8681/82 microcomputer can be found in Chapter 7.

1.8 APPLICATIONS

Z8 microcomputers are most often used in high-performance, dedicated applications. Such specialized functions were previously accomplished with TTL logic, TTL logic plus a low-end MCU, or a microprocessor and peripherals. Some typical applications include:

o Disc drive controller
o Printer controller
o Terminals
o Modems
o Industrial controllers
o Key telephones
o Telephone switching systems
o Arcade games and intelligent home games
o Process control
o Intelligent instrumentation
o Automotive mechanisms

Following are brief descriptions for a few Z8 applications.

Printers. Input data (typically transmitted via a terminal or computer) can be sent to the Z8 on either a serial or parallel port. The Z8 then transfers the data into the external RAM buffer via another parallel port, where it can operate on the data before output to the printing mechanism.

Disk. Disk operations are read or write, with input received from either the disk or the computer. Data is transferred to the buffer memory a sector (128, 256, 512, 1024 bytes) at a time via the Z8, operated on as required, and subsequently output to the disk or computer.

Terminal. Input is received from either the keyboard or a computer. The Z8 device must maintain at least an input buffer and often the screen RAM.

Chapter 2
Architectural Overview

2.1 INTRODUCTION

The Z8 is a versatile single-chip microcomputer. Because its multiplexed address/data bus is merged with several I/O-oriented ports, the Z8 can function as either an I/O-intensive or a memory-intensive microcomputer. One key advantage to this organization is that external memory can be addressed while maintaining many of the I/O lines. Figure 2-1 shows the Z8 block diagram.

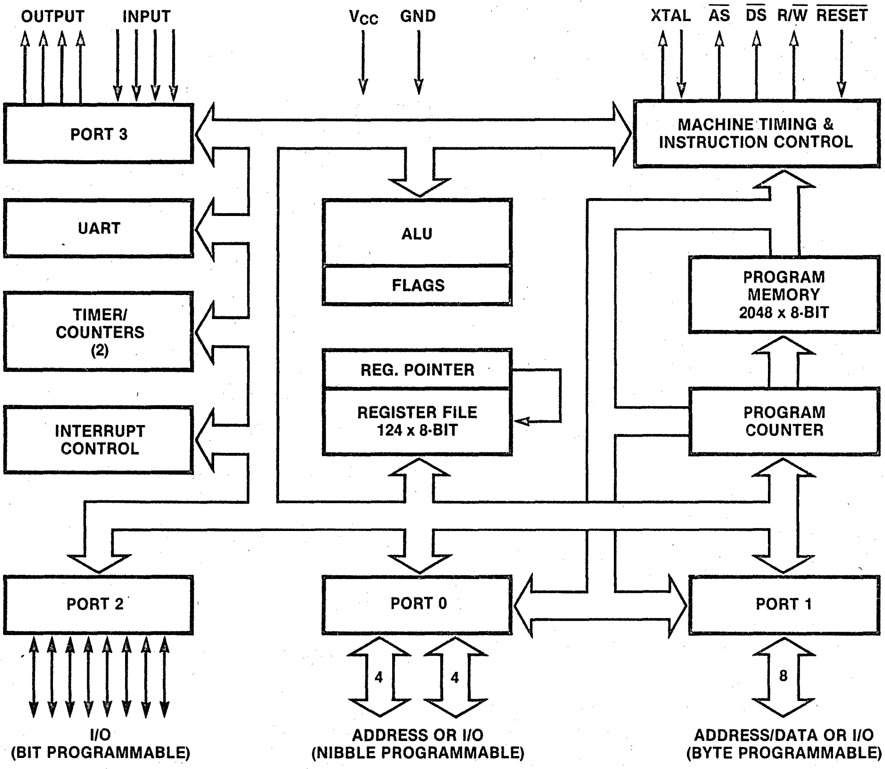
2.2 ADDRESS SPACES

To provide for both I/O-intensive and memory-intensive applications, the Z8 supports three basic address spaces:

- Program memory (internal and external)
- Data memory (external)
- Register file (internal)

A maximum of 64K bytes of program memory are directly addressable. In the Z8601 and Z8611 microcomputers, internal program memory consists of a mask-programmed ROM. The size of this internal ROM is 2K bytes for the Z8601 and 4K bytes for the Z8611. In one member of the Z8 family, the Z8681, all of the program memory is externally addressable.

Data memory space is always external to the Z8 microcomputer and is 62K bytes in size for the Z8601 and Z8682, and 60K and 64K bytes in size respectively for the Z8611 and Z8681.



OUTPUT
INPUT
Vcc
GND
XTAL
AS
DS
R/W
RESET
PORT 3
MACHINE TIMING & INSTRUCTION CONTROL
UART
ALU
FLAGS
PROGRAM MEMORY 2048 x 8-BIT
TIMER/ COUNTERS (2)
REG. POINTER
INTERRUPT CONTROL
REGISTER FILE 124 x 8-BIT
PROGRAM COUNTER
PORT 2
PORT 0
PORT 1
4
4
8
I/O (BIT PROGRAMMABLE)
ADDRESS OR I/O (NIBBLE PROGRAMMABLE)
ADDRESS/DATA OR I/O (BYTE PROGRAMMABLE)

Figure 2-1. Z8 Block Diagram

2.3 REGISTER FILE

The Z8's register-oriented architecture centers around an internal register file composed of 124 general-purpose registers, 16 CPU and peripheral control registers, and 4 I/O port registers. All registers are eight bits. Any general-purpose register can be used as an accumulator, an address pointer, or an index, data, or stack register.

2.3.1 Register Pointer

A Register Pointer logically divides the register file into 9 working register groups of 16 registers each, which allows for fast context switching and shorter instruction formats.

2.3.2 Instruction Set

The Z8 CPU has an instruction set designed for the large register file. The instruction set provides a full complement of 8-bit arithmetic and logical operations. BCD operations are supported using a decimal adjustment of binary values, and 16-bit quantities for addresses and counters can be incremented and decremented. Bit manipulation and Rotate and Shift instructions complete the data manipulation capabilities of the Z8 system. No special I/O instructions are necessary since the I/O is mapped into the register file.

2.3.3 Data Types

The Z8 CPU supports operations on bits, BCD digits, bytes, and 2-byte words.

Bits in the register file can be tested, set, cleared, and complemented. Bits within a byte are numbered from 0 to 7 with bit 0 being the least significant (right-most) bit (Figure 2-2).

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Figure 2-2. Bits in Register

Manipulation of BCD digits packed two-to-a-byte is accomplished by a Decimal Adjust instruction and a Swap instruction. Decimal Adjust is used after a binary addition or subtraction on BCD digits.

Logical, Shift, Rotate and Load instructions operate on bytes in the register file. Bytes in data memory are only affected by Load instructions.

Sixteen-bit arithmetic instructions (Increment Word and Decrement Word) operate on words in the register file.

2.3.4 Addressing Modes

The addressing modes of the Z8 CPU are:

- Register
- Indirect Register
- Immediate
- Direct Address
- Indexed (with a short 8-bit displacement)
- Program Counter Relative

Register, Indirect Register, and Immediate addressing modes are available for Load, Arithmetic, Logical, Shift, Rotate, and Stack instructions. Conditional Jumps use both Direct Address and Program Counter Relative, while Jump and Call instructions use Direct Address and Indirect Register addressing modes.

2.4 I/O OPERATIONS

The Z8 has 32 pins dedicated to input and output. These lines are grouped into four ports of eight lines each. Ports can be programmed as input, output, or bidirectional. Under software control, the ports provide timing, status signals, address outputs, and serial or parallel I/O with or without handshake. Multiprocessor system configurations are also supported.

2.4.1 Timers

To unburden the program from real-time problems such as serial data communications and counting/timing, the Z8 contains an on-chip universal asynchronous receiver/transmitter (UART) and two counter/timers with a large number of user-selectable modes. One on-chip timer provides the bit rate input to the UART during communications.

2.4.2 Interrupts

I/O operations can be interrupt-driven or polled. The Z8 supports six vectored interrupts that can be masked and prioritized.

2.5 OSCILLATOR

The Z8 offers an on-chip oscillator and an optional power-down mechanism that can be used to maintain the contents of the register file with a low-power battery.

2.6 PROTOPACK

The Z8 Protopack allows the user to prototype system hardware and develop software that is eventually to be mask-programmed into the on-chip ROM of the 2K byte (Z8601) or the 4K byte (Z8611) version of the Z8.

Chapter 3
Address Spaces

3.1 INTRODUCTION

Three address spaces are available in the Z8 microcomputer:

- The CPU Register File contains addresses for all general-purpose, peripheral, control, and I/O port registers.
- The CPU Program Memory contains addresses for all memory locations having executable code and/or data.
- The CPU Data Memory contains addresses for all memory locations that hold data only.

These address spaces are described in detail in the following sections.

3.2 CPU REGISTER FILE

The register file totals 256 consecutive bytes, of which 144 have been implemented. (Unused register space is reserved for future expansion.) The register file consists of 4 I/O ports (R0-R3), 124 general-purpose registers (R4-R127), 9 peripheral registers (R240-R248), and 7 control registers (R249-R255). Figure 3-1 shows the layout of the register file, including register names, locations, and identifiers.

Registers can be accessed as either 8- or 16-bit registers using Direct, Indirect, or Indexed addressing. All 144 registers can be referenced or modified by any instruction that accesses an 8-bit register, without the need for special instructions. Registers accessed as 16-bits are treated as even-odd register pairs (there are 72 valid pairs). In this case, the data's MSB is stored in the even-numbered register, while the LSB goes into the next higher, odd-numbered register (Figure 3-2).

DEC		HEX	IDENTIFIERS
255	STACK POINTER (BITS 7-0)	FF	SPL
254	STACK POINTER (BITS 15-8)	FE	SPH
253	REGISTER POINTER	FD	RP
252	PROGRAM CONTROL FLAGS	FC	FLAGS
251	INTERRUPT MASK REGISTER	FB	IMR
250	INTERRUPT REQUEST REGISTER	FA	IRQ
249	INTERRUPT PRIORITY REGISTER	F9	IPR
248	PORTS 0-1 MODE	F8	P01M
247	PORT 3 MODE	F7	P3M
246	PORT 2 MODE	F6	P2M
245	T0 PRESCALER	F5	PRE0
244	TIMER/COUNTER 0	F4	T0
243	T1 PRESCALER	F3	PRE1
242	TIMER/COUNTER 1	F2	T1
241	TIMER MODE	F1	TMR
240	SERIAL I/O	F0	SIO
	NOT IMPLEMENTED		
127	GENERAL-PURPOSE REGISTERS	7F	
4		04	
3	PORT 3	03	P3
2	PORT 2	02	P2
1	PORT 1	01	P1
0	PORT 0	00	P0

Figure 3-1. Register File

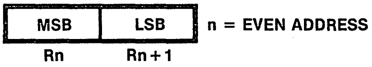


Figure 3-2. 16-Bit Register Addressing

By using logical instructions and a mask, individual bits within registers can be accessed for bit set, bit clear, bit complement, or bit test operations. For example, the instruction AND R, MASK performs a bit clear operation.

When instructions are executed, registers are read when defined as sources and written when defined as destinations. All general-purpose registers function as accumulators, address pointers, index registers, stack areas, or scratchpad memory.

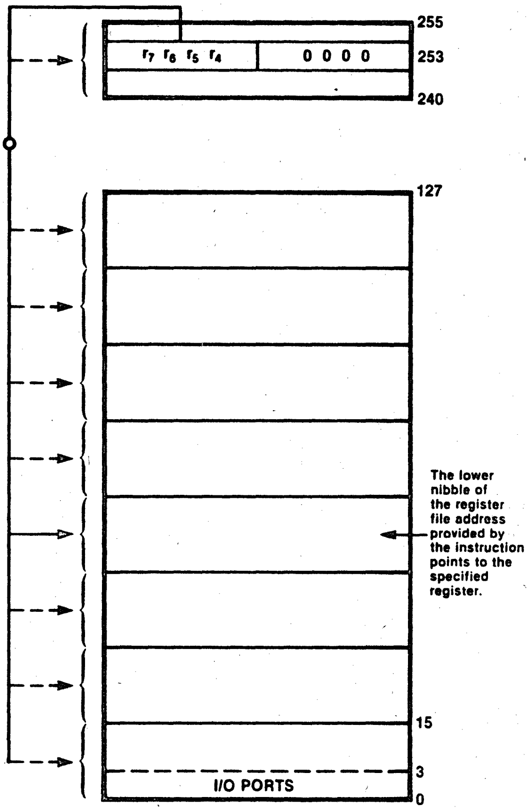
Z8 instructions can access 8-bit registers and register pairs (16-bit) using either 4-bit or 8-bit address fields. With 4-bit addressing, the register file is logically divided into 9 groups of 16 working registers as shown in Figure 3-3. A Register Pointer (one of the control registers) contains the base address of the active working register group.

When accessing one of the working registers, the 4-bit address is concatenated with the upper four bits of the Register Pointer, thus forming an 8-bit address. Figure 3-4 illustrates this operation. Since working registers are typically specified by short format instructions, there are fewer bytes of code needed, which reduces execution time. In addition, when processing interrupts or changing tasks, the Register Pointer speeds context switching. A special Set Register Pointer (SRP) instruction sets the contents of the Register Pointer.

3.2.1 Error Conditions

Registers must be correctly used because certain conditions produce inconsistent results and should be avoided:

- Registers R243 and R245-R249 are write-only registers. If an attempt is made to read these registers, %FF is returned (% is a prefix that indicates hexadecimal notation).
- When register R253 (Register Pointer) is read, all 0s are returned in the least significant four bits.



r_7 r_6 r_5 r_4
0 0 0 0
255
253
240
127
15
3
0
I/O PORTS
The lower nibble of the register file address provided by the instruction points to the specified register.

Figure 3-3. Working Register Groups

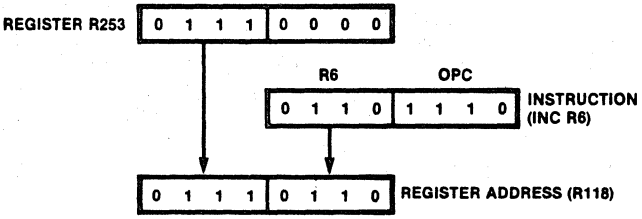


Figure 3-4. Working Register Addressing

- When registers R0 and R1 (Ports 0 and 1) are defined as address outputs, they will return 1s in each address bit location when read.

- Writing to bits which are defined as address output, timer output, serial output, or handshake output will have no effect.

- Instruction DJNZ uses a general register as a counter. Only registers R4-R127 can be used with this instruction.

3.3 CPU CONTROL AND PERIPHERAL REGISTERS

The Z8 control registers govern the operation of the CPU. Any instruction that references the register file can access these control registers. Available control registers are:

- Interrupt Priority register (IPR)
- Interrupt Mask register (IMR)
- Interrupt Request register (IRQ)
- Program Control flags (FLAGS)
- Register Pointer (RP)
- Stack Pointer - high-byte (SPH)
- Stack Pointer - low-byte (SPL)

The Z8 uses a 16-bit Program Counter (PC) to determine the sequence of current program instructions. The PC is not an addressable register.

Peripheral registers are used to transfer data, configure the operating mode, and control the operation of the on-chip peripherals. Any instruction that references the register file can access peripheral registers. The peripheral registers are:

- Serial I/O (SIO)
- Timer Mode (TMR)
- Timer/Counter 0 (T0)
- T0 Prescaler (PRE0)
- Timer/Counter 1 (T1)
- T1 Prescaler (PRE1)
- Port 0-1 Mode (P01M)
- Port 2 Mode (P2M)
- Port 3 Mode (P3M)

In addition, the four port registers (P0-P3) are considered to be peripheral registers.

The functions and applications of control and peripheral registers are described in subsequent sections of this manual.

3.4 CPU PROGRAM MEMORY

The Z8 can access 64K bytes of program memory with the 16-bit Program Counter. In the Z8601, the lower 2K bytes of the program memory address space are internal ROM, while in the Z8611 the lower 4K bytes are internal ROM. In the Z8682 the lower 2K bytes are not accessible.

To access program memory outside the on-board ROM space, Port 0 and Port 1 can be configured as a memory interface. For example, Port 1 as a multiplexed Address/Data port (AD_0-AD_7) provides Address lines A_0-A_7 and Data lines D_0-D_7. Port 0 can be configured for an additional four or eight address lines (A_8-A_{11} or A_8-A_{15}). This memory interface is supported by the control lines $\overline{AS}$ (Address Strobe), $\overline{DS}$ (Data Strobe) and R/$\overline{W}$ (Read/Write).

In the ROMless Z8681 version, Port 1 is automatically a multiplexed Address/Data port. Port 0 must be configured for additional address lines as needed.

The first 12 bytes of program memory are reserved for the interrupt vectors. Addresses 0-11 contain six 16-bit vectors that correspond to the six available interrupts. Figure 3-5 illustrates the order of 16-bit data stored in program memory.

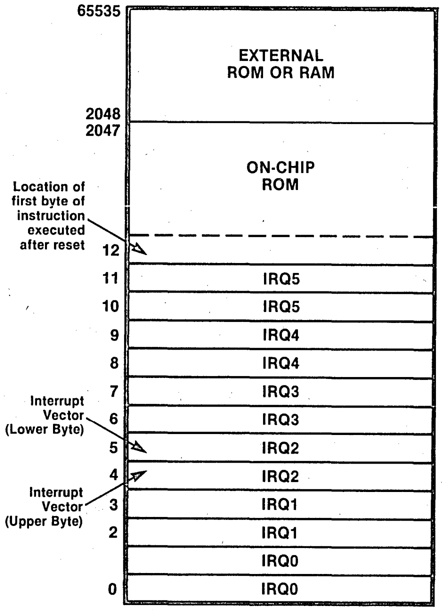

65535
EXTERNAL ROM OR RAM
2048
2047
ON-CHIP ROM
Location of first byte of instruction executed after reset
12
11
IRQ5
10
IRQ5
9
IRQ4
8
IRQ4
7
IRQ3
Interrupt Vector (Lower Byte)
6
IRQ3
5
IRQ2
4
IRQ2
Interrupt Vector (Upper Byte)
3
IRQ1
2
IRQ1
IRQ0
0
IRQ0

Figure 3-5a. Z8601 Program Memory Map

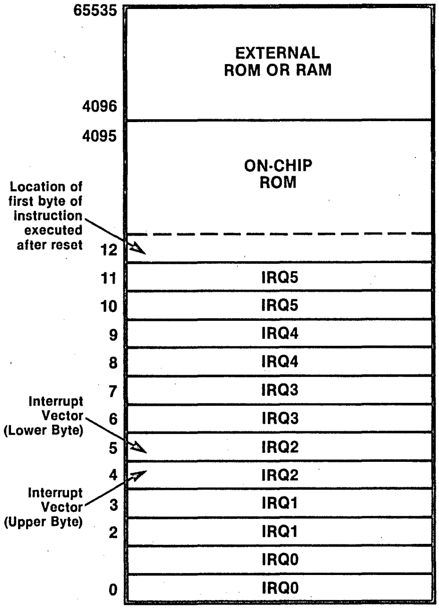

65535
EXTERNAL ROM OR RAM
4096
4095
ON-CHIP ROM
Location of first byte of instruction executed after reset
12
11
IRQ5
10
IRQ5
9
IRQ4
8
IRQ4
7
IRQ3
Interrupt Vector (Lower Byte)
6
IRQ3
5
IRQ2
4
IRQ2
Interrupt Vector (Upper Byte)
3
IRQ1
2
IRQ1
IRQ0
0
IRQ0

Figure 3-5b. Z8611 Program Memory Map

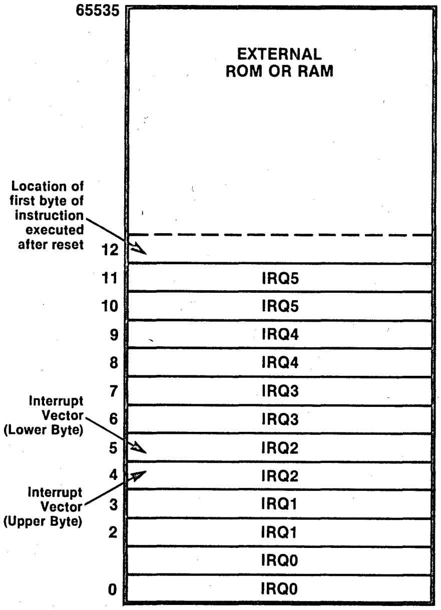

65535
EXTERNAL ROM OR RAM
Location of first byte of instruction executed after reset
12
11
IRQ5
10
IRQ5
9
IRQ4
8
IRQ4
7
IRQ3
Interrupt Vector (Lower Byte)
6
IRQ3
5
IRQ2
4
IRQ2
Interrupt Vector (Upper Byte)
3
IRQ1
2
IRQ1
IRQ0
0
IRQ0

Figure 3-5c. Z8681 Program Memory Map

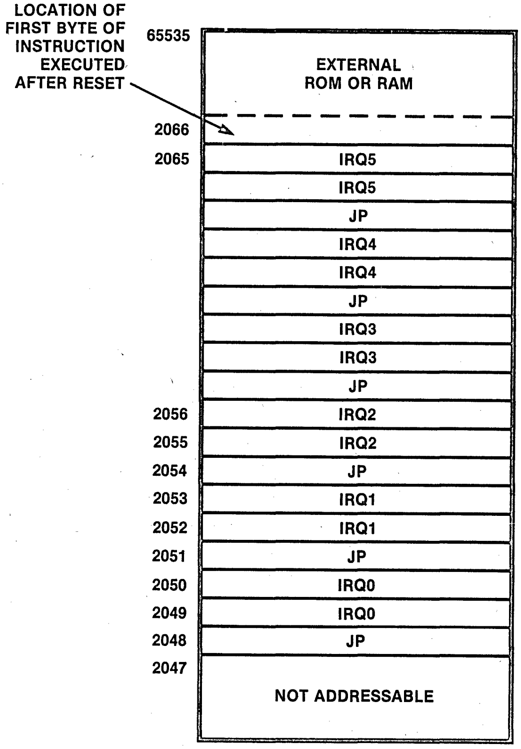


Figure 3-5d. Z8682 Program Memory Map

When an interrupt occurs, the address stored in the interrupt's vector location points to a service routine. This routine assumes program control.

The first 2K bytes of program memory are not addressable in the Z8682 ROMless version. Beginning at address 2048 the first 18 bytes contain interrupt vectors which are Jump Direct instructions. When an interrupt occurs, the Z8682 executes the corresponding Jump to interrupt.

The first address available for a user program is location 12. This address is loaded into the Program Counter after a hardware reset.

The first address available for a user program in the Z8682 is location 2066 (Hexadecimal %812). This address is loaded into the Program Counter after a hardware reset.

3.5 CPU DATA MEMORY

Up to 64K bytes of external data memory can be accessed in the Z8 microcomputer. As shown in Figure 3-6, the origin, and hence, the actual size of data memory is device-dependent. The origin of data memory is the same as the starting address of external program memory.

Like external program memory, external data memory Address/Data lines are provided by Port 1 for 8-bit addresses, and by Ports 0 and 1 for 12-bit and 16-bit addresses.

External data memory can be included with or separated from the external program memory addressing space. When data memory is separated from program memory, the Data Memory output ($\overline{DM}$) is used to select between data and program memories.

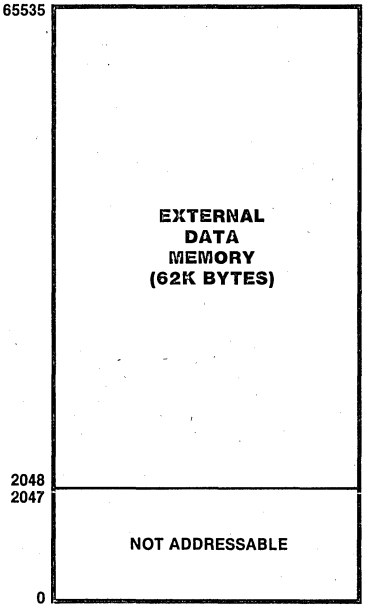


Figure 3-6a. Z8601 or Z8682 Data Memory Map

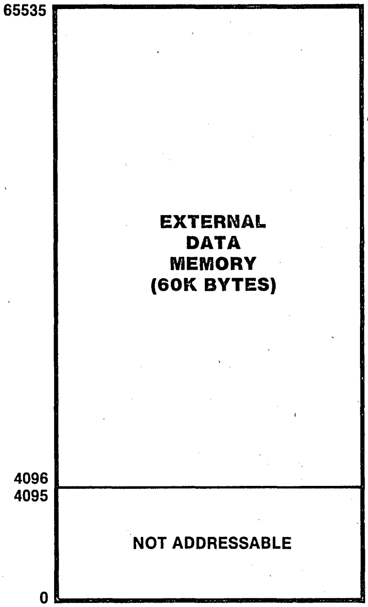


Figure 3-6b. Z8611 Data Memory Map

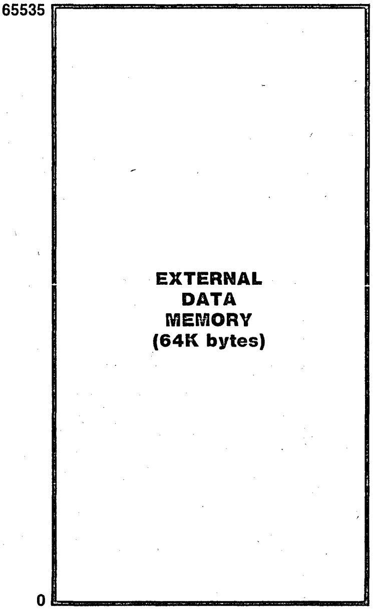


Figure 3-6c. Z8681 Data Memory Map

3.6 CPU STACKS

Stack operations can occur in either the register file or data memory. Under software control, Port 0 and 1 Mode register (R258) selects stack location.

The register pair R254 and R255 forms the 16-bit Stack Pointer (SP) which is used for all stack operations. The stack address is stored with the MSB in R254 and LSB in R255 (Figure 3-7).

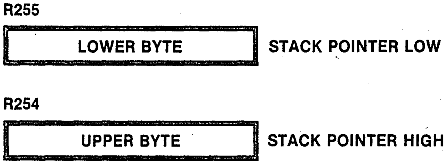


Figure 3-7. Stack Pointer

The stack address is decremented prior to a Push operation and incremented after a Pop operation. The stack address always points to the data stored on the top-of-stack. The Z8 stack is a return stack for Call instructions and interrupts as well as a data stack. During a Call instruction, the contents of the PC are saved on the stack. The PC is restored during a Return instruction. Interrupts cause the contents of the PC and Flag register to be saved on the stack. The IRET instruction restores them (Figure 3-8).

When the Z8 is configured for an internal stack (i.e., using the register file), register R255 serves as the Stack Pointer. The value in R254 is ignored and can be used as a general-purpose register. However, an overflow or underflow can occur when stack address is incremented or decremented during normal stack operations.

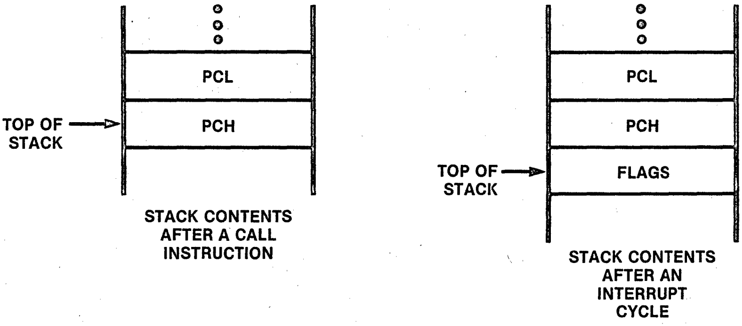

PCL
PCH
TOP OF STACK
STACK CONTENTS AFTER A CALL INSTRUCTION
PCL
PCH
FLAGS
TOP OF STACK
STACK CONTENTS AFTER AN INTERRUPT CYCLE

Figure 3-8. Stack Operations

Chapter 4
Address Modes

4.1 INTRODUCTION

The Z8 microcomputer provides six addressing modes:

- Register (R)
- Indirect Register (IR)
- Indexed (X)
- Direct (D)
- Relative (RA)
- Immediate (IM)

With the exception of immediate data and condition codes, all operands are expressed as register file, program memory, or data memory addresses. Registers are accessed using 8-bit addresses in the range 0-127 and 240-255.

Working registers are accessed using 4-bit addresses in the range 0-15. The address of the register being accessed is formed by the concatenation of the upper four bits in the Register Pointer (R253) with the 4-bit working register address supplied by the instruction.

Registers can be used in pairs to designate 16-bit values or memory addresses. A register pair must be specified as an even-numbered address in the range 0, 2,...., 14.

Addressing modes are instruction-specific. Section 5.4 discusses each addressing mode as it corresponds to particular instructions.

In the following definitions, the use of "register" also implies register pair, working register, or working register pair.

4.2 REGISTER ADDRESSING (R)

In the Register addressing mode, the operand value is the contents of the specified register or register pair (Figures 4-1 and 4-2).

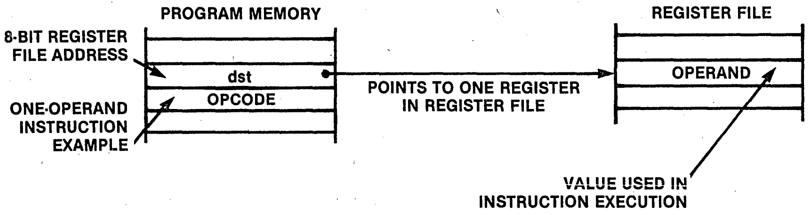

PROGRAM MEMORY
8-BIT REGISTER FILE ADDRESS
dst
OPCODE
ONE-OPERAND INSTRUCTION EXAMPLE
POINTS TO ONE REGISTER IN REGISTER FILE
REGISTER FILE
OPERAND
VALUE USED IN INSTRUCTION EXECUTION

Figure 4-1. Register Addressing

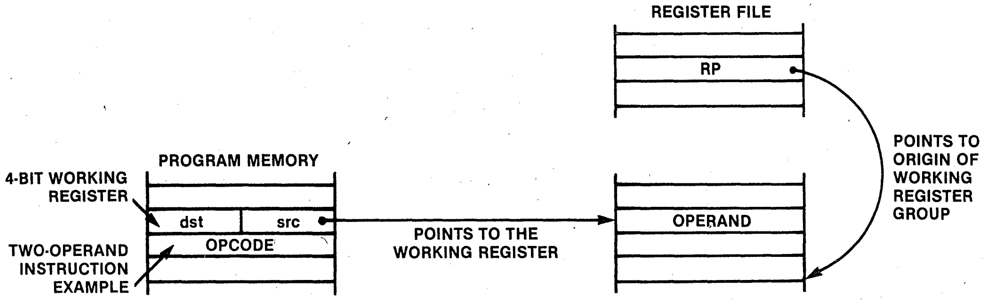

REGISTER FILE
RP
POINTS TO ORIGIN OF WORKING REGISTER GROUP
PROGRAM MEMORY
4-BIT WORKING REGISTER
dst
src
OPCODE
TWO-OPERAND INSTRUCTION EXAMPLE
POINTS TO THE WORKING REGISTER
OPERAND

Figure 4-2. Working-Register Addressing

4.3 INDIRECT REGISTER ADDRESSING (IR)

In the Indirect Register addressing mode, the contents of the specified register is the address of the operand (Figures 4-3 and 4-4).

Depending upon the instruction selected, the address points to a register, program memory, or an external data memory location.

When accessing program memory or external data memory, register pairs or working register pairs are used to hold the 16-bit addresses.

4.4 INDEXED ADDRESSING (X)

The Indexed addressing mode is used only by the Load (LD) instruction. An indexed address consists of a register address offset by the contents of a designated working register (the Index). This offset is added to the register address to obtain the address of the operand. Figure 4-5 illustrates this addressing convention.

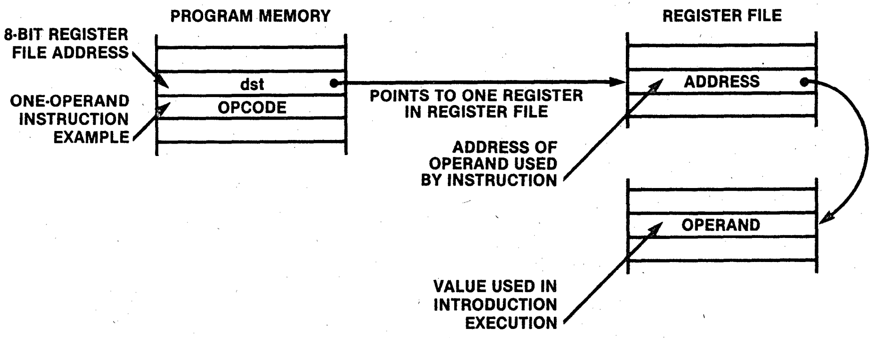

PROGRAM MEMORY
8-BIT REGISTER FILE ADDRESS
dst
OPCODE
ONE-OPERAND INSTRUCTION EXAMPLE
POINTS TO ONE REGISTER IN REGISTER FILE
REGISTER FILE
ADDRESS
ADDRESS OF OPERAND USED BY INSTRUCTION
OPERAND
VALUE USED IN INTRODUCTION EXECUTION

Figure 4-3. Indirect Register Addressing to Register File

Figure 4-4. Indirect Register Addressing to Program or Data Memory

4.5 DIRECT ADDRESSING (DA)

The Direct addressing mode, as shown in Figure 4-6, specifies the address of the next instruction to be executed. Only the Conditional Jump (JP) and Call (CALL) instructions use this addressing mode.

4.6 RELATIVE ADDRESSING (RA)

In the Relative addressing mode, illustrated in Figure 4-7, the instruction specifies a two's-complement signed displacement in the range of -128 to +127. This is added to the contents of the PC to obtain the address of the next instruction to be executed. The PC (prior to the add) consists of the address of the instruction following the Jump Relative (JR) or Decrement and Jump if Nonzero (DJNZ) instruction. JR and DJNZ are the only instructions that use this addressing mode.

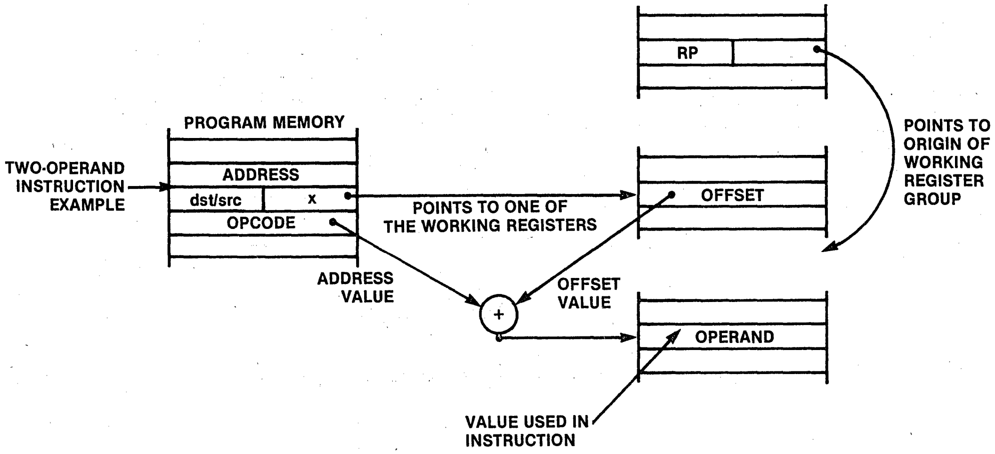

RP
POINTS TO ORIGIN OF WORKING REGISTER GROUP
PROGRAM MEMORY
TWO-OPERAND INSTRUCTION EXAMPLE
ADDRESS
dst/src
x
OPCODE
POINTS TO ONE OF THE WORKING REGISTERS
OFFSET
ADDRESS VALUE
+
OFFSET VALUE
OPERAND
VALUE USED IN INSTRUCTION

Figure 4-5. Indexed Addressing

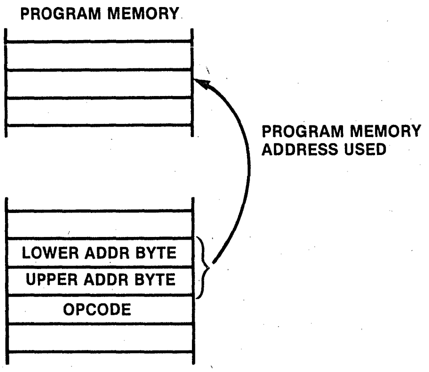


Figure 4-6. Direct Addressing

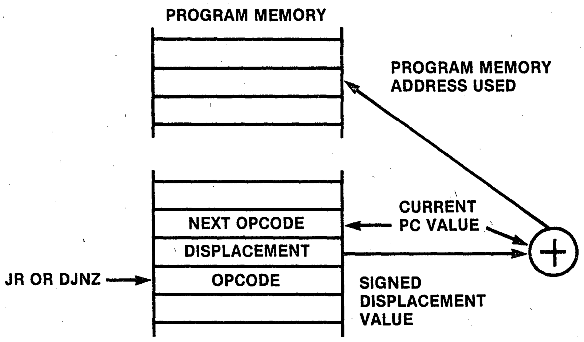


Figure 4-7. Relative Addressing

4.7 IMMEDIATE DATA ADDRESSING (IM)

Immediate data is considered an "addressing mode" for the purposes of this discussion. It is the only addressing mode that does not indicate a register or memory address as the source operand; the operand value used by the instruction is the value supplied in the operand field itself. Because an immediate operand is part of the instruction, it is always located in the program memory address space.

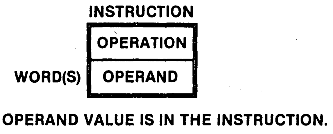


Figure 4-8. Immediate Data Addressing

Chapter 5
Instruction Set

5.1 FUNCTIONAL SUMMARY

Z8 instructions can be divided functionally into the following eight groups:

- Load
- Arithmetic
- Logical
- Program Control
- Bit Manipulation
- Block Transfer
- Rotate and Shift
- CPU Control

The following summary shows the instructions belonging to each group and the number of operands required for each. The source operand is "src", "dst" is the destination operand, and "cc" is a condition code.

Load Instructions

Mnemonic	Operands	Instruction
CLR	dst	Clear
LD	dst,src	Load
LDC	dst,src	Load Constant
LDE	dst,src	Load External
POP	dst	Pop
PUSH	src	Push

Arithmetic Instructions

Mnemonic	Operands	Instruction
ADC	dst,src	Add With Carry
ADD	dst,src	Add
CP	dst,src	Compare
DA	dst	Decimal Adjust
DEC	dst	Decrement
DECW	dst	Decrement Word
INC	dst	Increment
INCW	dst	Increment Word
SBC	dst,src	Subtract With Carry
SUB	dst,src	Subtract

Logical Instructions

Mnemonic	Operands	Instruction
AND	dst,src	Logical And
COM	dst	Complement
OR	dst,src	Logical Or
XOR	dst,src	Logical Exclusive Or

Program-Control Instructions

Mnemonic	Operands	Instruction
CALL	dst	Call Procedure
DJNZ	r,dst	Decrement and Jump NonO
IRET		Interrupt Return
JP	cc,dst	Jump
JR	cc,dst	Jump Relative
RET		Return

Bit-Manipulation Instructions

Mnemonic	Operands	Instruction
TCM	dst,src	Test Complement Under Mask
TM	dst,src	Test Under Mask
AND	dst,src	Bit Clear
OR	dst,src	Bit Set
XOR	dst,src	Bit Complement

Block-Transfer Instructions

Mnemonic	Operands	Instruction
LDCI	dst,src	Load Constant Autoincrement
LDEI	dst,src	Load External Autoincrement

Rotate and Shift Instructions

Mnemonic	Operands	Instruction
RL	dst	Rotate Left
RLC	dst	Rotate Left Through Carry
RR	dst	Rotate Right
RRC	dst	Rotate Right Through Carry
SRA	dst	Shift Right Arithmetic
SWAP	dst	Swap Nibbles

CPU Control Instructions

Mnemonic	Operand	Instruction
CCF		Complement Carry Flag
DI		Disable Interrupts
EI		Enable Interrupts
NOP		No Operation
RCF		Reset Carry Flag
SCF		Set Carry Flag
SRP	src	Set Register Pointer

5.2 PROCESSOR FLAGS

The Flag register (R252) informs the user about the current status of the Z8. The flags and their bit positions in the Flag register are shown in Figure 5-1.

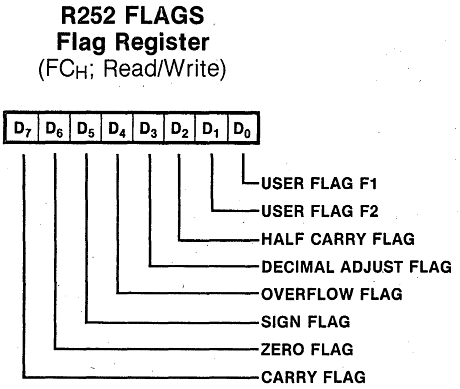


Figure 5-1. Flag Register

The Z8 Flag register contains six bits of status information which are set or cleared by CPU operations. Four of the bits (C, V, Z and S) can be tested for use with conditional Jump instructions. Two flags (H, D) cannot be tested and are used for BCD arithmetic.

The two remaining bits in the Flag register (F1, F2) are available to the user, but they must be set or cleared by instruction and are not usable with conditional Jumps.

As with bits in the other control registers, Flag register bits can be set or reset by instructions; however, only those instructions that do not affect the flags as an outcome of the execution should be used (e.g., Load Immediate).

5.2.1 Carry Flag (C)

The Carry flag is set to 1 whenever the result of an arithmetic operation generates a carry out of or a borrow into the high order bit 7; otherwise, the Carry flag is cleared to 0.

Following Rotate and Shift instructions, the Carry flag contains the last value shifted out of the specified register.

An instruction can set, reset, or complement the Carry flag.

RETI changes the value of the Carry flag when the saved Flag register is restored.

5.2.2 Zero Flag (Z)

For arithmetic and logical operations, the Zero flag is set to 1 if the result is zero; otherwise, the Zero flag is cleared.

If the result of testing bits in a register is 0, the Zero flag is set to 1; otherwise the flag is cleared.

If the result of a Rotate or Shift operation is 0, the Zero flag is set to 1; otherwise, the flag is cleared.

RETI changes the value of the Zero flag when the saved Flag register is restored.

5.2.3 Sign Flag (S)

The Sign flag stores the value of the most significant bit of a result following arithmetic, logical, Rotate, or Shift operations.

When performing arithmetic operations on signed numbers, binary two's complement notation is used to represent and process information. A positive number is identified by a 0 in the most significant bit position, and therefore, the Sign flag is also 0.

A negative number is identified by a 1 in the most significant bit position, and therefore, the Sign flag is also 1.

RETI changes the value of the Zero flag when the saved Flag register is restored.

5.2.4 Overflow Flag (V)

For signed arithmetic, Rotate, and Shift operations, the Overflow flag is set to 1 when the result is greater than the maximum possible number (> 127) or less than the minimum possible number (< -128) that can be represented in two's complement form. The flag is set to 0 if no overflow occurs.

Following logical operations, the Overflow flag is set to 0.

RETI changes the value of the Overflow flag when the saved Flag register is restored.

5.2.5 Decimal-Adjust Flag (D)

The Decimal-adjust flag is used for BCD arithmetic. Since the algorithm for correcting BCD operations is different for addition and subtraction, this flag specifies what type of instruction was last executed so that the subsequent Decimal Adjust (DA) operation can function properly. Normally, the Decimal-adjust flag cannot be used as a test condition.

After a subtraction, the Decimal-adjust flag is set to 1; following an addition it is cleared to 0.

RETI changes the value of the Decimal-adjust flag when the saved Flag register is restored.

5.2.6 Half-Carry Flag (H)

The Half-carry flag is set to 1 whenever an addition generates a carry out of bit 3 (Overflow), or a subtraction generates a borrow into bit 3. The Half-carry flag is used by the Decimal Adjust (DA) instruction to convert the binary result of a previous addition or subtraction into the correct decimal (BCD) result. As in the case of the Decimal-adjust flag, the user does not normally access this flag.

RETI changes the value of the Half-carry flag when the saved Flag register is restored.

5.3 CONDITION CODES

Flags C, Z, S, and V control the operation of the "conditional" Jump instructions. Sixteen frequently useful functions of the flag settings are encoded in a 4-bit field called the condition code (CC), which forms bits 4-7 of the conditional instructions.

Section 5.4.2 lists the condition codes and the flag settings they represent.

5.4 NOTATION AND BINARY ENCODING

In the detailed instruction descriptions that make up the rest of this chapter, operands and status flags are represented by a notational shorthand. Operands (condition codes and address modes) and their notations are as follows:

Notation	Address Mode	Actual Operand/Range
cc	Condition Code	See condition code list below
r	Working register only	Rn: where n = 0-15
R	Register or working register	reg: where reg represents a number in the range 0-127, 240-255 Rn: where n = 0-15
RR	Register pair or working register pair	reg: where reg represents an even number in the range 0-126, 240-254 RRp: where p = 0, 2,...,14
Ir	Indirect working register only	@ Rn: where n = 0-15
IR	Indirect register or working register	@ reg: where reg represents a number in the range 0-127, 240-255 @ Rn: where n = 0-15
Irr	Indirect working register pair only	@ RRp: where p = 0, 2,...,14
IRR	Indirect register pair or working register pair	@ reg: where reg resents an even number in the range 0-126, 240-254 @ RRp: where p = 0, 2,...,14

Notation	Address Mode	Actual Operand/Range
X	Indexed	reg (Rn): where reg represent a number in the range 0-127, 240-255 and n = 0-15
DA	Direct Address	addrs: where addrs represents a number in the range 0-65,535
RA	Relative Address	addrs: where addrs represents a number in the range +127, -128 which is an offset relative to the address of the next instruction
IM	Immediate	#data: where data is a number between 0 and 255

Additional symbols used are:

Symbol	Meaning
dst	Destination operand
src	Source operand
@	Indirect address prefix
SP	Stack Pointer
PC	Program Counter
FLAGS	Flag register (R252)
RP	Register Pointer (R253)
IMR	Interrupt mask register (251)
#	Immediate operand prefix
%	Hexadecimal number prefix
OPC	Opcode

Assignment of a value is indicated by the symbol "<-". For example,

dst <- dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

5.4.1 Assembly Language Syntax

For proper instruction execution, Z8 PLZ/ASM assembly language syntax requires that "dst, src" be specified, in that order. The following instruction descriptions show the format of the object code produced by the assembler. This binary format should be followed by users who prefer manual program coding or who intend to implement their own assembler.

Example: If the contents of registers %43 and %08 are added and the result stored in %43, the assembly syntax and resulting object code are:

```
ASM:  ADD   %43, %08   (ADD dst, src)
OBJ:  04    08   43    (OPC src, dst)
```

In general, whenever an instruction format requires an 8-bit register address, that address can specify any register location in the range 0-127, 240-255 or a working register R0-R15. If, in the above example, register %08 is a working register, the assembly syntax and resulting object code would be:

```
ASM:  ADD  %43, R8    (ADD dst src)
OBJ:  04   E8   43    (OPC src dst)
```

For a more complete description of assembler syntax refer to the Z8 PLZ/ASM Assembly Language Manual (publication no. 03-3023-03) and ZSCAN 8 User's Tutorial (publication no. 03-8200-01).

5.4.2 Condition Codes and Flag Settings

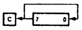
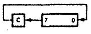
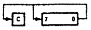
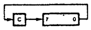
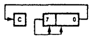
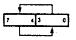
The condition codes and flag settings are summarized in the following tables. Notation for the flags and how they are affected are as follows:

C	Carry flag	0	Cleared to 0
Z	Zero flag	1	Set to 1
S	Sign flag	*	Set or cleared according to operation
V	Overflow flag		
D	Decimal-adjust flag	-	Unaffected
H	Half-carry flag	X	Undefined

Condition Codes

Binary	Mnemonic	Meaning	Flags Settings
0000	F	Always false	-
1000	(blank)	Always true	-
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not 0	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater Than	(Z OR (S XOR V))=0
0010	LE	Less than or equal	(Z OR (S XOR V))=1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C=0 AND Z=0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
ADC dst,src dst ← dst + src + C	(Note 1)		1□	*	*	*	*	0	*
ADD dst,src dst ← dst + src	(Note 1)		0□	*	*	*	*	0	*
AND dst,src dst ← dst AND src	(Note 1)		5□	–	*	*	0	–	–
CALL dst SP ← SP - 2 @SP ← PC; PC ← dst	DA IRR		D6 D4	–	–	–	–	–	–
CCF C ← NOT C			EF	*	–	–	–	–	–
CLR dst dst ← 0	R IR		B0 B1	–	–	–	–	–	–
COM dst dst ← NOT dst	R IR		60 61	–	*	*	0	–	–
CP dst,src dst - src	(Note 1)		A□	*	*	*	*	–	–
DA dst dst ← DA dst	R IR		40 41	*	*	*	X	–	–
DEC dst dst ← dst - 1	R IR		00 01	–	*	*	*	–	–
DECW dst dst ← dst - 1	RR IR		80 81	–	*	*	*	–	–
DI IMR (7) ← 0			8F	–	–	–	–	–	–
DJNZ r,dst r ← r - 1 if r ≠ 0 PC ← PC + dst Range: +127, -128	RA		rA r=0-F	–	–	–	–	–	–
EI IMR (7) ← 1			9F	–	–	–	–	–	–
INC dst dst ← dst + 1	r R IR		rE r=0-F 20 21	–	*	*	*	–	–
INCW dst dst ← dst + 1	RR IR		A0 A1	–	*	*	*	–	–
IRET FLAGS ← @SP; SP ← SP + 1 PC ← @SP; SP ← SP + 2; IMR (7) ← 1			BF	*	*	*	*	*	*
JP cc,dst if cc is true PC ← dst	DA IRR		cD c=0-F 30	–	–	–	–	–	–
JR cc,dst if cc is true, PC ← PC + dst Range: +127, -128	RA		cB c=0-F	–	–	–	–	–	–
LD dst,src dst ← src	r r R r X r Ir R R R IR IR	IM R r X r Ir r R IR IM IM R	rC r8 r9 r=0-F C7 D7 E3 F3 E4 E5 E6 E7 F5	–	–	–	–	–	–
LDC dst,src dst ← src	r Irr	Irr r	C2 D2	–	–	–	–	–	–
LDCI dst,src dst ← src r ← r + 1; rr ← rr + 1	Ir Irr	Irr Ir	C3 D3	–	–	–	–	–	–

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
LDE dst,src dst ← src	r Irr	Irr r	82 92	–	–	–	–	–	–
LDEI dst,src dst ← src r ← r + 1; rr ← rr + 1	Ir Irr	Irr Ir	83 93	–	–	–	–	–	–
NOP			FF	–	–	–	–	–	–
OR dst,src dst ← dst OR src	(Note 1)		4□	–	*	*	0	–	–
POP dst dst ← @SP SP ← SP + 1	R IR		50 51	–	–	–	–	–	–
PUSH src SP ← SP - 1; @SP ← src		R IR	70 71	–	–	–	–	–	–
RCF C ← 0			CF	0	–	–	–	–	–
RET PC ← @SP; SP ← SP + 2			AF	–	–	–	–	–	–
RL dst 	R IR		90 91	*	*	*	*	–	–
RLC dst 	R IR		10 11	*	*	*	*	–	–
RR dst 	R IR		E0 E1	*	*	*	*	–	–
RRC dst 	R IR		C0 C1	*	*	*	*	–	–
SBC dst,src dst ← dst - src - C	(Note 1)		3□	*	*	*	*	1	*
SCF C ← 1			DF	1	–	–	–	–	–
SRA dst 	R IR		D0 D1	*	*	*	0	–	–
SRP src RP ← src		Im	31	–	–	–	–	–	–
SUB dst,src dst ← dst - src	(Note 1)		2□	*	*	*	*	1	*
SWAP dst 	R IR		F0 F1	X	*	*	X	–	–
TCM dst,src (NOT dst) AND src	(Note 1)		6□	–	*	*	0	–	–
TM dst,src dst AND src	(Note 1)		7□	–	*	*	0	–	–
XOR dst,src dst ← dst XOR src	(Note 1)		B□	–	*	*	0	–	–

Note 1

These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the left of the applicable addressing mode pair.

For example, to determine the opcode of an ADC instruction using the addressing modes r (destination) and Ir (source) is 13.

Addr Mode dst	Addr Mode src	Lower Opcode Nibble
r	r	$\boxed{2}$
r	Ir	$\boxed{3}$
R	R	$\boxed{4}$
R	IR	$\boxed{5}$
R	IM	$\boxed{6}$
IR	IM	$\boxed{7}$

ADC
Add With Carry

ADC dst,src

Instruction Format:

					OPC	Address Mode	
				Cycles	(Hex)	dst	src
OPC	dst	src		6	12	r	r
					13	r	Ir
OPC	src		dst	10	14	R	R
					15	R	IR
OPC	dst		src	10	16	R	IM
					17	IR	IM

Operation: dst <-- dst + src + c

The source operand, along with the setting of the C flag, is added to the destination operand and the sum is stored in the destination. The contents of the source are not affected. Two's complement addition is performed. In multiple precision arithmetic, this instruction permits the carry from the addition of low-order operands to be carried into the addition of high-order operands.

Flags:

C: Set if there is a carry from the most-significant bit of the result; cleared otherwise
Z: Set if the result is zero; cleared otherwise
S: Set if the result is negative; cleared otherwise
V: Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise
D: Always cleared
H: Set if there is a carry from the most-significant bit of the low-order four bits of the result; cleared otherwise

Example: If the register named SUM contains %16, the C flag is set to 1, working register 10 contains %20 (32 decimal), and register 32 contains %10, the statement

```
ADC SUM,@R10
```

leaves the value %27 in Register SUM. The C, Z, S, V, D, and H flags are all cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src/dst

ADD
Add

ADD dst,src

Instruction Format:

				OPC	Address Mode	
			Cycles	**(Hex)**	**dst**	**src**
OPC	dst | src		6	02	r	r
				03	r	Ir
OPC	src	dst	10	04	R	R
				05	R	IR
OPC	dst	src	10	06	R	IM
				07	IR	IM

Operation: dst <-- dst + src

The source operand is added to the destination operand and the sum is stored in the destination. The contents of the source are not affected. Two's complement addition is performed.

Flags:

C: Set if there was a carry from the most-significant bit of the result; cleared otherwise
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Set if a carry from the low-order nibble occurs
D: Always reset to 0

Example: If the register named SUM contains %44 and the register named AUGEND contains %11, the statement

ADD SUM,AUGEND

leaves the value %55 in register SUM and leaves all flags cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src/dst

AND dst,src

Instruction Format:

				OPC	Address Mode	
			Cycles	(Hex)	dst	src
OPC	dst | src		6	52	r	r
				53	r	IR
OPC	src	dst	10	54	R	R
				55	R	IR
OPC	dst	src	10	56	R	IM
				57	IR	IM

Operation: dst <-- dst AND src

The source operand is logically ANDed with the destination operand. The result is stored in the destination. The AND operation results in a 1 bit being stored whenever the corresponding bits in the two operands are both 1s; otherwise a 0 bit is stored. The contents of the source bit are not affected.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the source operand is the immediate value %7B (01111011) and the register named TARGET contains %C3 (11000011), the statement

AND TARGET, #%7B

leaves the value %43 (01000011) in register TARGET. The Z, V, and S flags are cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src/dst

CALL
Call Procedure

CALL dst

Instruction Format:

		Cycles	OPC (Hex)	Address Mode dst
OPC	dst	20	D6	DA
OPC	dst	20	D4	IRR

Operation:

```
SP <-- SP - 2
@SP <-- PC
PC <-- dst
```

The current contents of the PC are pushed onto the top of the stack. The PC value is the address of the first instruction following the CALL instruction. The specified destination address is then loaded into the PC and points to the first instruction of a procedure.

At the end of the procedure a RETurn instruction can be used to return to the original program flow. RET pops the top of the stack back into the PC.

Flags: No flags affected.

Example: If the contents of the PC are %1A47 and the contents of the SP (control registers 254-5) are %3002, the statement

CALL %3521

causes the SP to be decremented to %3000, %1A4A (the address following the instruction) is stored in external data memory %3000-%3001, and the PC is loaded with %3521. The PC now points to the address of the first statement in the procedure to be executed.

Note: When used to specify a 4-bit working-register pair address, address mode IRR uses the format:

E	dst

CCF
Complement Carry Flag

CCF

Instruction Format:

OPC

Cycles	OPC (Hex)
6	EF

Operation: C <-- NOT C

The C flag is complemented; if C = 1, it is changed to C = 0, and vice-versa.

Flags: C: Complemented
No other flags affected

Example: If the C flag contains a 0, the statement

```
CCF
```

will change the 0 to 1.

CLR
Clear

CLR dst

Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	B0	R
	B1	IR

Operation: dst <-- 0

The destination location is cleared to 0.

Flags: No flags affected.

Example: If working register 6 contains %AF, the statement

CLR R6

will leave the value 0 in that register

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

COM
Complement

COM dst

Instruction Format:

OPC	dst

	OPC	Address Mode
Cycles	(Hex)	dst
6	60	R
	61	IR

Operation: dst <-- NOT dst

The contents of the destination location are complemented (one's complement); all 1 bits are changed to 0, and vice-versa.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If working register 8 contains %24 (00100100), the statement

COM R8

leaves the value %DB (11011011) in that register. The Z and V flags are cleared and the S flag is set.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

CP
Compare

CP dst,src

Instruction Format:

			Cycles	OPC (Hex)	Address Mode dst	Address Mode src
OPC	dst | src		6	A2	r	r
				A3	r	Ir
OPC	src	dst	10	A4	R	R
				A5	R	IR
OPC	dst	src	10	A6	R	IM
				A7	IR	IM

Operation: dst - src

The source operand is compared to (subtracted from) the destination operand, and the appropriate flags set accordingly. The contents of both operands are unaffected by the comparison.

Flags:

C: Cleared if there is a carry from the most significant bit of the result; set otherwise, indicating a "borrow"
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurs; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the register named TEST contains %63, working register 0 contains %30 (48 decimal), and register 48 contains %63, the statement

CP TEST, @R0

sets (only) the Z flag. If this statement is followed by "JP EQ, true_routine", the jump is taken.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src/dst

DA
Decimal Adjust

DA dst

Instruction Format:

		Cycles	OPC (Hex)	Address Mode dst
OPC	dst	8	40	R
			41	IR

Operation: dst <-- DA dst

The destination operand is adjusted to form two 4-bit BCD digits following a binary addition or subtraction operation on BCD encoded bytes. For addition (ADD, ADC), or subtraction (SUB, SBC), the following table indicates the operation performed:

Instruction	Carry Before DA	Bits 4-7 Value (Hex)	H Flag Before DA	Bits 0-3 Value (Hex)	Number Added To Byte	Carry After DA
ADD ADC	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
	0	A-F	0	0-9	60	1
	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
SUB SBC	0	0-9	0	0-9	00	0
	0	0-8	1	6-F	FA	0
	1	7-F	0	0-9	A0	1
	1	6-F	1	6-F	9A	1

If the destination operand is not the result of a valid addition or subtraction of BCD digits, the operation is undefined.

Flags:

C: Set if there is a carry from the most significant bit; cleared otherwise (see table above)
Z: Set if the result is 0; cleared otherwise
V: Undefined
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If addition is performed using the BCD values 15 and 27, the result should be 42. The sum is incorrect, however, when the binary representations are added in the destination location using standard binary arithmetic.

```
    0001  0101
  + 0010  0111
    0011  1100  = %3C
```

The DA statement adjusts this result so that the correct BCD representation is obtained.

```
    0011  1100
  + 0000  0110
    0100  0010  =  42
```

The C, Z, and S flags are cleared and V is undefined.

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

DEC
Decrement

DEC dst

Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	00	R
	01	IR

Operation: dst <-- dst - 1

The destination operand's contents are decremented by one.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example: If working register 10 contains %2A, the statement

```
DEC R10
```

leaves the value %29 in that register. The Z, V, and S flags are cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

DECW
Decrement Word

DECW dst

Instruction Format:

		Cycles	OPC (Hex)	Address Mode dst
OPC	dst	10	80	RR
			81	IR

Operation: dst <-- dst - 1

The contents of the destination location (which must be an even address) and the operand following that location are treated as a single 16-bit value which is decremented by one.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example: If working register 0 contains %30 (48 decimal) and registers 48-49 contain the value %FAF3, the statement

```
DECW @R0
```

leaves the value %FAF2 in registers 48 and 49. The Z and V flags are cleared and S is set.

DI
Disable Interrupts

DI

Instruction Format:

OPC

Cycles	OPC (Hex)
6	8F

Operation: IMR (7) <-- 0

Bit 7 of control register 251 (the Interrupt Mask Register) is reset to 0. All interrupts are disabled, although they remain potentially enabled (i.e., the Global Interrupt Enable is cleared--not the individual interrupt level enables.)

Flags: No flags affected

Example: If control register 251 contains %8A (10001010, that is, interrupts IRQ1 and IRQ3 are enabled), the statement

```
DI
```

sets control register 251 to %0A and disables these interrupts.

DJNZ
Decrement and Jump if Nonzero

DJNZ r,dst

Instruction Format:

			Cycles	OPC (Hex)	Address Mode dst
r	OPC	dst	12 if jump taken 10 if jump not taken	rA r=0 to F	RA

Operation:

r <-- r - 1
If r $\neq$ 0, PC <-- PC + dst

The working register being used as a counter is decremented. If the contents of the register are not zero after decrementing, the relative address is added to the Program Counter (PC) and control passes to the statement whose address is now in the PC. The range of the relative address is +127, -128, and the original value of the PC is the address of the instruction byte following the DJNZ statement. When the working register counter reaches zero, control falls through to the statement following DJNZ.

Flags:

No flags affected

Example:

DJNZ is typically used to control a "loop" of instructions. In this example, 12 bytes are moved from one buffer area in the register file to another. The steps involved are:

o Load 12 into the counter (working register 6)
o Set up the loop to perform the moves
o End the loop with DJNZ

```
      LD R6, #12          !Load Counter!
LOOP: LD R9,OLDBUF (R6)   !Move one byte to!
      LD NEWBUF (R6),R9   !New location!
      DJNZ R6,LOOP        !Decrement and !
                          !Loop until counter = 0!
```

Note:

The working register being used as a counter must be one of the registers 04-7F. Use of one of the I/O ports, control or peripheral registers will have undefined results.

EI
Enable Interrupts

EI

Instruction Format:

OPC

Cycles	OPC (Hex)
6	9F

Operation: IMR (7) <-- 1

Bit 7 of control register 251 (the Interrupt Mask Register) is set 10 to 1. This allows any potentially enabled interrupts to become enabled.

Flags: No flaqs affected

Example: If control register 251 contains %0A (00001010, that is, interrupts IRQ1 and IRQ3 potentially enabled), the statement

EI

sets control register 251 to %8A (10001010) and enables these interrupts.

INC
Increment

INC dst

Instruction Format:

dst	OPC

OPC		dst

Cycles	OPC (Hex)	Address Mode dst
6	rE r=0 to F	r
6	20 21	R IR

Operation: dst <-- dst + 1

The destination operand's contents are incremented by one.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example: If working register 10 contains %2A, the statement

INC R10

leaves the value %2B in that register. The Z, V, and S flags are cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

INCW
Increment Word

INCW dst

Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
10	A0	RR
	A1	IR

Operation: dst <-- dst + 1

The contents of the destination (which must be an even address) and the byte following that location are treated as a single 16-bit value which is incremented by one.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example: If working-register pair 0-1 contains the value %FAF3, the statement

```
INCW RR0
```

leaves the value %FAF4 in working-register pair 0-1. The Z and V flags are cleared and S is set.

IRET
Interrupt Return

IRET

Instruction Format:

	Cycles	OPC (Hex)
OPC	16	BF

Operation:

```
FLAGS <-- @SP
SP <-- SP + 1
PC <-- @SP
SP <-- SP + 2
IMR (7) <-- 1
```

This instruction is issued at the end of an interrupt service routine. It restores the Flag register (control register 252) and the PC. It also reenables any interrupts that are potentially enabled.

Flags: All flags are restored to original settings (before interrupt occurred).

JP
Jump

JP cc,dst

Instruction Format:

		Cycles	OPC (Hex)	Address Mode dst
Conditional				
cc | OPC	dst	12 if jump taken 10 if jump not taken	ccD cc=0 to F	DA
Unconditional				
OPC	dst	8	30	IRR

Operation: If cc is true, PC <-- dst

A conditional jump transfers Program Control to the destination address if the condition specified by "cc" is true; otherwise, the instruction following the JP instruction is executed. See Section 6.4 for a list of condition codes.

The unconditional jump simply replaces the contents of the Program Counter with the contents of the specified register pair. Control then passes to the statement addressed by the PC, decremented by one.

Flags: No flags affected

Example: If the carry flag is set, the statement

```
JP C,%1520
```

replaces the contents of the Program Counter with %1520 and transfers control to that location. Had the carry flag not been set, control would have fallen through to the statement following the JP.

Note: When used to specify a 4-bit working-register pair address, address mode IRR uses the format:

E	dst

JR
Jump Relative

JR   cc,dst

Instruction Format:

cc	OPC		dst

	Cycles	OPC (Hex)	Address Mode dst
	12 If jump taken 10 If jump not taken	ccB cc=0 to F	RA

Operation: If cc is true, PC <-- PC + dst

If the condition specified by "cc" is true, the relative address is added to the PC and control passes to the statement whose address in now in the PC; otherwise, the instruction following the JR instruction is executed. (See Section 5.3 for a list of condition codes). The range of the relative address is +127, -128, and the original value of the PC is taken to be the address of the first instruction byte following the JR statement.

Flags: No flags affected

Example: If the result of the last arithmetic operation executed is negative, the following four statements (which occupy a total of seven bytes) are skipped with the statement

```
JR MI,$+9
```

If the result is not negative, execution continues with the statement following the JR. A short form of a jump to label LO is

```
JR LO
```

where LO must be within the allowed range. The condition code is "blank" in this case, and is assumed to be "always true."

LD dst,src

Instruction Format:

				OPC	Address	Mode
			Cycles	**(Hex)**	**dst**	**src**
dst OPC	src		6	rC	r	IM
			6	r8	r	R
src OPC	dst		6	r9 r=0 to F	R*	r
OPC	dst src		6	E3	r	Ir
			6	F3	Ir	r
OPC	src	dst	10	E4	R	R
			10	E5	R	IR
OPC	dst	src	10	E6	R	IM
			10	E7	IR	IM
OPC	src	dst	10	F5	IR	R
OPC	dst x	src	10	C7	r	X
OPC	src x	dst	10	D7	X	r

*In this instance only a full 8-bit register address can be used.

Operation: dst <-- src

The contents of the source are loaded into the destination. The contents of the source are not affected.

Flags: No flags affected

Example: If working register 0 contains %0B (11 decimal) and working register 10 contains %83, the statement

LD 240(R0),R10

will load the value %83 into register 251 (240 + 11). Since this is the Interrupt Mask register, the Load statement has the effect of enabling IRQ0 and IRQ1. The contents of working register 10 are unaffected by the load.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src/dst

LDC
Load Constant

LDC dst,src

Instruction Format:

				OPC	Address Mode	
			Cycles	**(Hex)**	**dst**	**src**
OPC	dst	src	12	C2	r	Irr
OPC	src	dst	12	D2	Irr	r

Operation: dst <-- src

This instruction is used to load a byte constant from program memory into a working register, or vice-versa. The address of the program memory location is specified by a working register pair. The contents of the source are not affected.

Flags: No flags affected

Example: If the working-register pair 6-7 contains %30A2 and program-memory location %30A2 contains the value %22, the statement

```
LDC R2, @RR6
```

loads the value %22 into working register 2. The value of location %30A2 is unchanged by the load.

LDCI
Load Constant Autoincrement

LDCI dst,src

Instruction Format:

				OPC	Address Mode	
			Cycles	(Hex)	dst	src
OPC	dst	src	18	C3	Ir	Irr
OPC	src	dst	18	D3	Irr	Ir

Operation:

```
dst <-- src
r <-- r + 1
rr <-- rr + 1
```

This instruction is used for block transfers of data between program memory and the register file. The address of the program-memory location is specified by a working-register pair, and the address of the register-file location is specified by a working register. The contents of the source location are loaded into the destination location. Both addresses are then incremented automatically. The contents of the source are not affected.

Flags:

No flags affected

Example:

If the working-register pair 6-7 contains %30A2 and program-memory locations %30A2 and %30A3 contain %22BC, and if working register R2 contains %20 (32 decimal), the statement

```
LDCI @R2, @RR6
```

loads the value %22 into register 32. A second

```
LDCI @R2, @RR6
```

loads the value %BC into register 33.

LDE
Load External Data

LDE dst,src

Instruction Format:

				OPC	Address Mode	
			Cycles	**(Hex)**	**dst**	**src**
OPC	dst	src	12	82	r	Irr
OPC	src	dst	12	92	Irr	r

Operation: dst <-- src

This instruction is used to load a byte from external data memory into a working register or vice-versa. The address of the external data-memory location is specified by a working-register pair. The contents of the source are not affected.

Flags: No flags affected

Example: If the working-register pair 6-7 contains %404A and working register 2 contains %22, the statement

```
LDE @RR6,R2
```

loads the value %22 into external data-memory location %404A.

LDEI
Load External Data Autoincrement

LDEI dst,src

Instruction Format:

			Cycles	OPC (Hex)	Address Mode	
					dst	src
OPC	dst	src	18	83	Ir	Irr
OPC	src	dst	18	93	Irr	Ir

Operation:

```
dst <-- src
r <-- r + 1
rr <-- rr + 1
```

This instruction is used for block transfers of data between external data memory and the register file. The address of the external data-memory location is specified by a working-register pair, and the address of the register file location is specified by a working register. The contents of the source location are loaded into the destination location. Both addresses are then incremented automatically. The contents of the source are not affected.

Flags: No flags affected

Example: If the working-register pair 6-7 contains %404A, working register 2 contains %22 (34 decimal), and registers 34-35 contain %ABC3, the statement

LDEI @RR6,@R2

loads the value %AB into external location %404A. A second

LDEI @RR6,@R2

loads the value %C3 into external location %404B.

Note: When used to specify a 4-bit working-register pair address, address modes RR or IR use the format:

E	dst

NOP
No Operation

NOP

Instruction Format:

OPC

Cycles	OPC (Hex)
6	FF

Operation: No action is performed by this instruction. It is typically used for timing delays.

Flags: No flags affected

OR
Logical Or

OR dst,src

Instruction Format:

			Cycles	OPC (Hex)	Address Mode dst	Address Mode src
OPC	dst | src		6	42	r	r
			6	43	r	Ir
OPC	src	dst	10	44	R	R
			10	45	R	IR
OPC	dst	src	10	46	R	IM
			10	47	IR	IM

Operation: dst <-- dst OR src

The source operand is logically ORed with the destination operand and the result is stored in the destination. The contents of the source are not affected. The OR operation results in a one bit being stored whenever either of the corresponding bits in the two operands is 1; otherwise a 0 bit is stored.

Flags:

C: Unaffected
Z: Set if result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the source operand is the immediate value %7B (01111011) and the register named TARGET contains %C3 (11000011), the statement

```
OR TARGET,#%7B
```

leaves the value %FB (11111011) in register TARGET. The Z and V flags are cleared and S is set.

Note: When used to specify a 4-bit working-register address, address modes R and IR use the format:

E	src/dst

POP
Pop

POP dst

Instruction Format:

OPC	dst

Cycles	OPC (Hex)	Address Mode dst
10	50	R
10	51	IR

Operation:

dst <-- @SP
SP <-- SP + 1

The contents of the location addressed by the SP are loaded into the destination. The SP is then incremented automatically.

Flags:

No flags affected

Example:

If the SP (control registers 254-255) contains %1000, external data-memory location %1000 contains %55, and working register 6 contains %22 (34 decimal), the statement

```
POP @R6
```

loads the value %55 into register 34. After the POP operation, the SP contains %1001.

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

PUSH
Push

PUSH src

Instruction Format:

OPC	src

	Cycles	OPC (Hex)	Address Mode src
10	Internal stack	70	R
12	External stack		
12	Internal stack	71	IR
14	External stack		

Operation:

```
SP <-- SP - 1
@SP <-- src
```

The contents of the SP are decremented, then the contents of the source are loaded into the location addressed by the decremented SP, thus adding a new element to the top of the stack.

Flags: No flags affected

Example: If the SP contains %1001, the statement

PUSH FLAGS

stores the contents of the register named FLAGS in location %1000. After the PUSH operation, the SP contains %1000.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src

RCF
Reset Carry Flag

RCF

Instruction Format:

		OPC
	Cycles	(Hex)
OPC	6	CF

Operation: C <-- 0

The C flag is reset to 0, regardless of its previous value.

Flags: **C:** Reset to 0
No other flags affected

RET
Return

RET

Instruction Format:

OPC

Cycles	OPC (Hex)
14	AF

Operation:

```
PC <-- @SP
SP <-- SP + 2
```

This instruction is normally used to return to the previously executed procedure at the end of a procedure entered by a CALL instruction. The contents of the location addressed by the SP are popped into the PC. The next statement executed is that addressed by the new contents of the PC.

Flags: No flags affected

Example: If the PC contains %35B4, the SP contains %2000, external data-memory location %2000 contains %18, and location %2001 contains %B5, then the statement

```
RET
```

leaves the value %2002 in the SP and the PC contains %18B5, the address of the next instruction.

RL
Rotate Left

RL dst

Instruction Format:

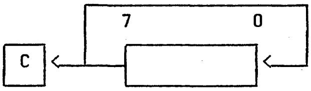
OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	90	R
6	91	IR

Operation:

```
C <-- dst(7)
dst(0) <-- dst(7)
dst(n + 1) <-- dst(n) n = 0 - 6
```

The contents of the destination operand are rotated left one bit position. The initial value of bit 7 is moved to the bit 0 position and also replaces the carry flag.



Flags:

C: Set if the bit rotated from the most significant bit position was 1; i.e., bit 7 was 1
Z: Set if the result is zero; cleared otherwise.
V: Set if arithmetic overflow occurred; that is, if the sign of the destination changed during rotation; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the contents of the register named SHIFTER are %88 (10001000), the statement

RL SHIFTER

leaves the value %11 (00010001) in that register. The C flag and V flags are set to 1 and the Z flag is cleared.

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

RLC
Rotate Left Through Carry

RLC dst

Instruction Format:

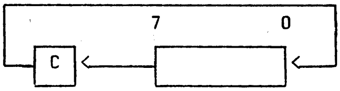
OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	10	R
6	11	IR

Operation:

dst (0) <-- C
C <-- dst (7)
dst(n + 1) <-- dst(n) n = 0 - 6

The contents of the destination operand with the C flag are rotated left one bit position. The initial value of bit 7 replaces the C flag; the initial value of the C flag replaces bit 0.



Flags:

C: Set if the bit rotated from the most significant bit position was 1; i.e., bit 7 was 1
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurs, that is, if the sign of the destination changed during rotation; cleared otherwise
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the C flag is reset (to 0) and the register named SHIFTER contains %8F (10001111), the statement

RLC SHIFTER

sets the C flag and the V flag to 1 and SHIFTER contains %1E (00011110).

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

RR
Rotate Right

RR dst

Instruction Format:

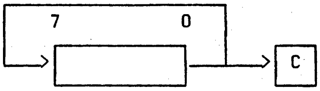
OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	E0	R
6	E1	IR

Operation:

```
C <-- dst(0)
dst(7) <-- dst(0)
dst(n) <-- dst(n + 1) n = 0 - 6
```

The contents of the destination operand are rotated right one bit position. The initial value of bit 0 is moved to bit 7 and also replaces the C flag.



Flags:

C: Set if the bit rotated from the least significant bit position was 1; i.e., bit 0 was 1
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the contents of working register 6 are %31 (00110001), the statement

```
RR R6
```

sets the C flag to 1 and leaves the value %98 (10011000) in working register 6. Since bit 7 now equals 1, the S flag and the V flag are also set.

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

RRC
Rotate Right Through Carry

RRC dst

Instruction Format:

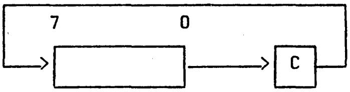
OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	C0	R
6	C1	IR

Operation:

```
dst(7) <-- C
C <-- dst(0)
dst(n) <-- dst(n + 1)   n = 0 - 6
```

The contents of the destination operand with the C flag are rotated right one bit position. The initial value of bit 0 replaces the C flag; the initial value of the C flag replaces bit 7.



Flags:

C: Set if the bit rotated from the least significant bit position was 1; i.e., bit 0 was 1
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred, that is, the sign of the destination changed during rotation; cleared otherwise
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the contents of the register named SHIFTER are %DD (11011101) and the Carry flag is reset to 0, the statement

RRC SHIFTER

sets the C flag and the V flag and leaves the value %6E (01101110) in the register.

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

SBC
Subtract With Carry

SBC dst,src

Instruction Format:

				OPC	Address Mode	
			Cycles	(Hex)	dst	src
OPC	dst | src		6	32	r	r
			6	33	r	Ir
OPC	src	dst	10	34	R	R
			10	35	R	IR
OPC	dst	src	10	36	R	IM
			10	37	IR	IM

Operation: dst <-- dst - src - C

The source operand, along with the setting of the C flag, is subtracted from the destination operand and the result is stored in the destination. The contents of the source are not affected. Subtraction is performed by adding the two's complement of the source operand to the destination operand. In multiple precision arithmetic, this instruction permits the carry ("borrow") from the subtraction of low-order operands to be subtracted from the subtraction of high-order operands.

Flags:

C: Cleared if there is a carry from the most significant bit of the result; set otherwise, indicating a "borrow"
Z: Set if the result is 0; cleared otherwise
V: Set if arithmetic overflow occurred, that is, if the operands were of opposite sign and the sign of the result is the same as the sign of the source; reset otherwise
S: Set if the result is negative; cleared otherwise
H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a "borrow."
D: Always set to 1

Example: If the register named MINUEND contains %16, the Carry flag is set to 1, working register 10 contains %20 (32 decimal), and register 32 contains %05, the statement

SBC MINUEND, @R10

leaves the value %10 in register MINUEND. The C, Z, V, S and H flags are cleared and D is set.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src/dst

SCF
Set Carry Flag

SCF

Instruction Format:

Instruction Format	Cycles	OPC (Hex)
OPC	6	DF

Operation: C <-- 1

The C flag is set to 1, regardless of its previous value.

Flags: **C:** Set to 1
No other flags affected

SRA
Shift Right Arithmetic

SRA dst

Instruction Format:

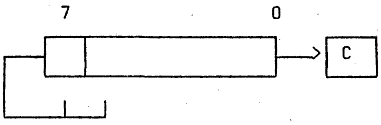
OPC	dst

Cycles	OPC (Hex)	Address Mode dst
6	D0	R
6	D1	IR

Operation:

dst(7) <-- dst(7)
C <-- dst(0)
dst(n) <-- dst(n + 1)   n = 0 - 6

An arithmetic shift right one bit position is performed on the destination operand. Bit 0 replaces the C flag. Bit 7 (the Sign bit) is unchanged, and its value is also shifted into bit position 6.



Flags:

C: Set if the bit shifted from the least significant bit position was 1; i.e., bit 0 was 1
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result is negative; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the register named SHIFTER contains %B8 (10111000), the statement

SRA SHIFTER

resets the C flag to 0 and leaves the value %DC (11011100) in register SHIFTER. The S flag is set to 1.

Note:

When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	dst

SRP
Set Register Pointer

SRP src

Instruction Format:

OPC	src

Cycles	OPC (Hex)	Address Mode src
6	31	IM

Operation: RP <-- src

The specified value is loaded into bits 4-7 of the Register Pointer (RP) (control register 253). Bits 0-3 of the RP are always set to 0. The source data (with bits 0-3 forced to 0) is the starting address of a working-register group. The working-register group starting addresses are:

Hex	Decimal
%00	0
%10	16
%20	32
%30	48
%40	64
%50	80
%60	96
%70	112
%F0	240 (control and peripheral registers)

Values in the range %80-E0 are invalid.

Flags: No flags affected

Example: Assume the RP currently addresses the control and peripheral register group and the program has just entered an interrupt service routine. The statement

```
SRP #%70
```

saves the contents of the control and peripheral registers by setting the RP to %70 (01110000), or 112 decimal. Any reference to working registers in the interrupt routine will point to registers 112-127.

SUB
Subtract

SUB dst,src

Instruction Format:				**OPC**	**Address Mode**	
			Cycles	**(Hex)**	**dst**	**src**
OPC	dst | src		6	22	r	r
			6	23	r	Ir
OPC	src	dst	10	24	R	R
			10	25	R	IR
OPC	dst	src	10	26	R	IM
			10	27	IR	IM

Operation: dst <-- dst - src

The source operand is subtracted from the destination operand and the result is stored in the destination. The contents of the source are not affected. Subtraction is performed by adding the two's complement of the source operand to the destination operand.

Flags:

C: Cleared if there is a carry from the most significant bit of the result; set otherwise, indicating a "borrow"
Z: Set if the result is zero; cleared otherwise
V: Set if arithmetic overflow occurred, that is, if the operands were of opposite signs and the sign of the result is the same as the sign of the source operand; cleared otherwise
S: Set if the result is negative; cleared otherwise
H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a "borrow."
D: Always set to 1

Example: If the register named MINUEND contains %29, the statement

SUB MINUEND, #%11

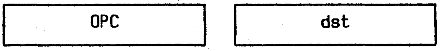
will leave the value %18 in the register. The C, Z, V, S and H flags are cleared and D is set.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src/dst

SWAP dst

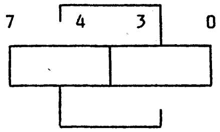
Instruction Format:



Cycles	OPC (Hex)	Address Mode dst
8	F0	R
8	F1	IR

Operation: dst(0 - 3) <--> dst(4 - 7)

The contents of the lower four bits and upper four bits of the destination operand are swapped.



Flags:

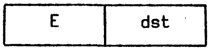
C: Undefined
Z: Set if the result is zero; cleared otherwise
V: Undefined
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: Suppose the register named BCD_Operands contains %B3 (10110011). The statement

SWAP BCD_Operands

will leave the value %3B (00111011) in the register. The Z and S flags are cleared.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



TCM
Test Complement Under Mask

TCM dst,src

Instruction Format:

			Cycles	OPC (Hex)	Address Mode dst	Address Mode src
OPC	dst | src		6 6	62 63	r r	r Ir
OPC	src	dst	10 10	64 65	R R	R IR
OPC	dst	src	10 10	66 67	R IR	IM IM

Operation: (NOT dst) AND src

This instruction tests selected bits in the destination operand for a logical "1" value. The bits to be tested are specified by setting a 1 bit in the corresponding position of the source operand (mask). The TCM statement complements the destination operand, which is then ANDed with the source mask. The Zero (Z) flag can then be checked to determine the result. When the TCM operation is complete, the destination location still contains its original value.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the register named TESTER contains %F6 (11110110) and the register named MASK contains %06 (00000110), that is, bits 1 and 2 are being tested for a 1 value, the statement

```
TCM TESTER, MASK
```

complements TESTER (to 00001001) and then do a logical AND with register MASK, resulting in %00. A subsequent test of the Z flag,

```
JP Z,plabel
```

causes a transfer of program control. At the end of this sequence, TESTER still contains %F6.

Note: When used to specify a 4-bit working-reqister address, address modes R or IR use the format:

E	src/dst

TM
Test Under Mask

TM dst,src

Instruction Format:

			Cycles	OPC (Hex)	Address Mode dst	src
OPC	dst src		6 6	72 73	r r	r Ir
OPC	src	dst	10 10	74 75	R R	R IR
OPC	dst	src	10 10	76 77	R IR	IM IM

Operation: dst AND src

This instruction tests selected bits in the destination operand for a logical "0" value. The bits to be tested are specified by setting a 1 bit in the corresponding position of the source operand (mask), which is ANDed with the destination operand. The Z flag can be checked to determine the result. When the TM operation is complete, the destination location still contains its original value.

Flags:

C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example:

If the register named TESTER contains %F6 (11110110) and the register named MASK contains %06 (00000110), that is, bits 1 and 2 are being tested for a 0 value, the statement

```
TM TESTER, MASK
```

results in the value %06 (00000110). A subsequent test for nonzero

```
JP NZ, plabel
```

causes a transfer of program control. At the end of this sequence, TESTER still contains %F6. The Z and S flags are cleared.

Note:

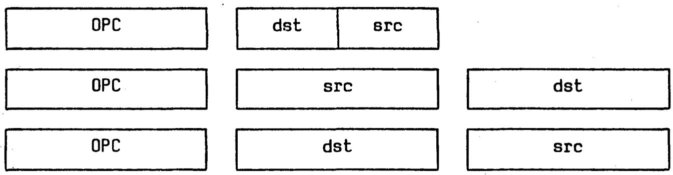
When used to specify a 4-bit working-register address, address modes R or IR use the format:

E	src/dst

XOR
Logical Exclusive OR

XOR dst,src

Instruction Format:



Cycles	OPC (Hex)	Address Mode dst	Address Mode src
6	B2	r	r
6	B3	r	Ir
10	B4	R	R
10	B5	R	IR
10	B6	R	IM
10	B7	IR	IM

Operation: dst <-- dst XOR src

The source operand is logically EXCLUSIVE ORed with the destination operand and the result stored in the destination. The EXCLUSIVE OR operation results in a one bit being stored whenever the corresponding bits in the operands are different; otherwise, a 0 bit is stored.

Flags:

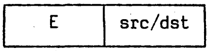
C: Unaffected
Z: Set if the result is zero; cleared otherwise
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise
H: Unaffected
D: Unaffected

Example: If the source operand is the immediate value %7B (011111011) and the register named TARGET contains %C3 (11000011), the statement

```
OR TARGET, #%7B
```

leaves the value %B8 (10111000) in the register.

Note: When used to specify a 4-bit working-register address, address modes R or IR use the format:



Chapter 6
External Interface
(Z8601, Z8611)

6.1 INTRODUCTION

The ROM versions of the Z8 microcomputer have 40 external pins, of which 32 are programmable I/O pins. The remaining 8 pins are used for power and control. Up to 16 I/O pins can be configured as an external memory interface. This interface function is the subject of this chapter. The I/O mode of these pins is described in Chapter 9.

6.2 PIN DESCRIPTIONS

$\overline{AS}$. Address Strobe (output, active Low, 3-state, pin 9). Address Strobe is pulsed Low once at the beginning of each machine cycle. The rising edge of $\overline{AS}$ indicates that addresses, Read/Write (R/$\overline{W}$), and Data Memory ($\overline{DM}$) signals, are valid when output for external program or data memory transfers. Under program control, $\overline{AS}$ can be placed in a high-impedance state along with Ports 0 and 1, Data Strobe ($\overline{DS}$), and R/$\overline{W}$.

$\overline{DS}$. Data Strobe (output, active Low, 3-state, pin 8). Data Strobe provides the timing for data movement to or from Port 1 for each external memory transfer. During a Write cycle, data out is valid at the leading edge of $\overline{DS}$. During a Read cycle, data in must be valid prior to the trailing edge of $\overline{DS}$. $\overline{DS}$ can be placed in a high-impedance state along with Ports 0 and 1, $\overline{AS}$, and R/$\overline{W}$.

R/$\overline{W}$. Read/Write. (output, 3-state, pin 7). Read/Write determines the direction of data transfer for external memory transactions. R/$\overline{W}$ is Low when writing to external program or data memory, and High for all other transactions. R/$\overline{W}$ can be placed in a high-impedance state along with Ports 0 and 1, $\overline{AS}$, and $\overline{DS}$.

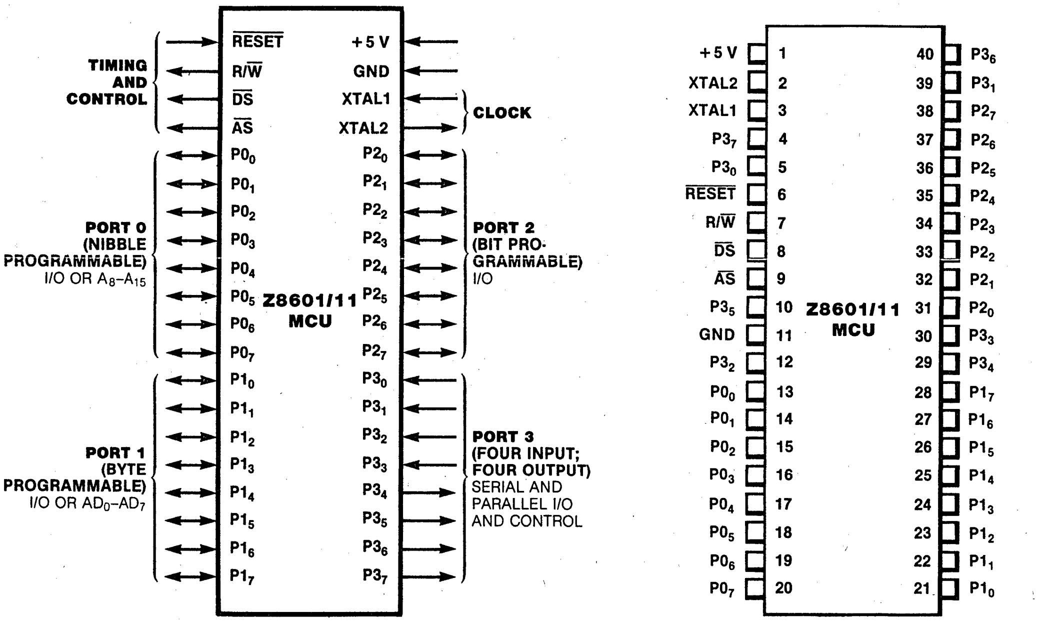


Figure 6-2. Z8601/11 Pin Assignments

$P0_0$-$P0_7$, $P1_0$-$P1_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. I/O port lines (inputs/outputs, TTL-compatible, pins 12-40). These 32 I/O lines are divided into four 8-bit I/O ports that can be configured under program control for I/O or external memory interface. Individual lines of a port are denoted by the second digit of the port number. For example, $P3_0$ refers to bit 0 of Port 3. Ports 0 and 1 can be placed in a high-impedance state along with $\overline{AS}$, $\overline{DS}$, and $R/\overline{W}$.

$\overline{RESET}$. Reset (input, active Low, pin 6). $\overline{RESET}$ initializes the Z8. When $\overline{RESET}$ is deactivated, program execution begins from internal program location %C. If held Low, $\overline{RESET}$ acts as a register file protect during power-down and power-up sequences. $\overline{RESET}$ also enables the Z8 Test mode.

XTAL1, XTAL2. Crystal 1, Crystal 2 (oscillator input and output, pins 3 and 2). These pins connect a parallel-resonant crystal (12 MHz maximum) or an external source (12 MHz maximum) to the on-board clock oscillator and buffer.

6.3 CONFIGURING FOR EXTERNAL MEMORY

Before interfacing with external memory, the user must configure Ports 0 and 1 appropriately. The minimum bus configuration uses Port 1 as a multiplexed Address/Data port (AD_0-AD_7), allowing access to 256 bytes of external memory. In this configuration, the eight lower order address bits (A_0-A_7) are multiplexed with the data (D_0-D_7).

Port 0 can be programmed to provide four additional address lines (A_8-A_{11}), which increases the externally addressable program memory to 4K bytes. Port 0 can also be programmed to provide eight additional address lines (A_8-A_{15}), which increases the externally addressable memory to 62K bytes for the Z8601 or 60K bytes for the Z8611. Refer to Chapter 3, Figures 3-5 and 3-6, for external memory maps.

Ports 0 and 1 are configured for external memory operation by writing the appropriate bits in the Port 0-1 Mode register (Figure 6-3).

For example, Port 1 can be defined as a multiplexed Address/Data port (AD_0-AD_7) by setting D_4 to 1 and D_3 to 0. The lower nibble of Port 0 can be defined as address lines A_8-A_{11}, by setting D1 to 1. Similarly, setting D7 to 1 defines the upper nibble of Port 0 as address lines A_{12}-A_{15}. Whenever Port 0 is configured to output address lines A_{12}-A_{15}, A_8-A_{11} must also be selected as address lines.

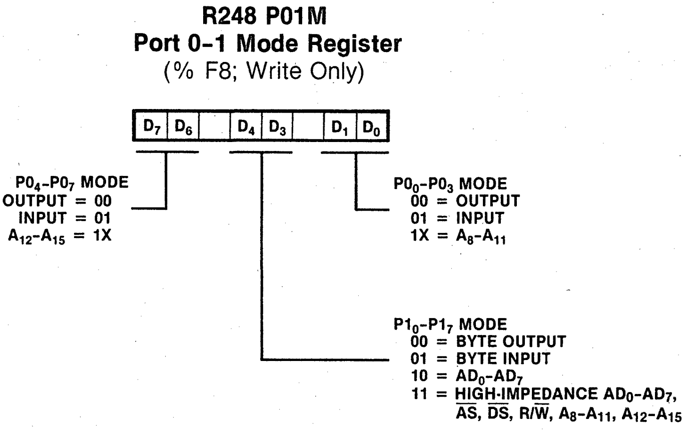

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)

Figure 6-3. Ports 0 and 1 External Memory Operation

Once Port 1 is configured as an Address/Data port, it can no longer be used as a register. Attempting to read Port 1 returns FF; writing has no effect. Similarly, if Port 0 is configured for address lines A_8-A_{15}, it can no longer be used as a register. However, if only the lower nibble is defined as address lines A_8-A_{11}, the upper nibble is still addressable as an I/O register. Reading Port 0 with only the lower nibble defined as address outputs returns XF, where X equals the data in bits D_4-D_7. Writing to Port 0 transfers data to the I/O nibble only.

An instruction to change the modes of Ports 0 or 1 should not be immediately followed by an instruction that performs a stack operation, because this may cause indeterminate program flow. In addition, after setting the modes of Ports 0 and 1 for external memory, the next three bytes must be fetched from internal program memory.

6.4 EXTERNAL STACKS

Z8 architecture supports stack operations in either the register file or data memory. A stack's location is determined by bit D_2 in the Port 0-1 Mode register. For example, if D_2 is set to 1, the stack is in internal data memory (Figure 6-4).

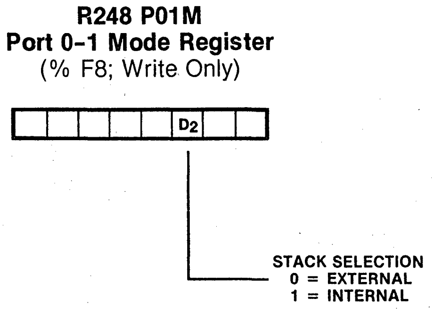


Figure 6-4. Ports 0 and 1 Stack Selection

The instruction used to change the stack selection bit should not be immediately followed by the instructions RET or IRET, because this will cause indeterminate program flow.

6.5 DATA MEMORY

The two external memory spaces, data and program, can be addressed as a single memory space or as two separate spaces of equal size; i.e., 62K bytes each for the Z8601 and 60K bytes each for the Z8611. If the memory spaces are separated, program memory and data memory are logically selected by the Data Memory select output ($\overline{\text{DM}}$). DM is available on Port 3, line 4 ($P3_4$) by setting bits D_4 and D_3 in the Port 3 Mode register to 10 or 01 (Figure 6-5). $\overline{\text{DM}}$ is active Low during the execution of the LDE, LDEI instructions. $\overline{\text{DM}}$ is also active during the execution of CALL, POP, PUSH, RET and IRET instructions if the stack resides in external memory.

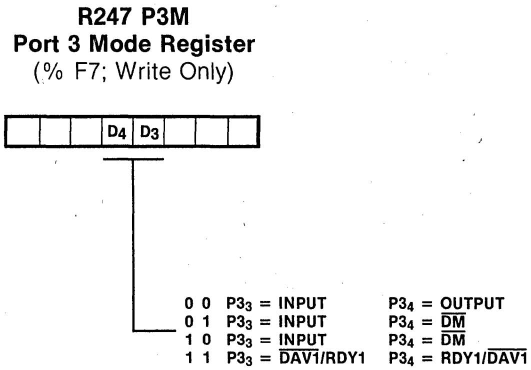


Figure 6-5. Data Memory Operation

6.6 BUS OPERATION

The timing for typical data transfers between the Z8 and external memory is illustrated in Figure 6-6. Machine cycles can vary from six to twelve clock periods depending on the operation being performed. The notations used to describe the basic timing periods of the Z8 are: machine cycles (Mn), timing states (Tn), and clock periods. All timing references are made with respect to the output signals $\overline{\text{AS}}$ and $\overline{\text{DS}}$. The clock is shown for clarity only and does not have a specific timing relationship with other signals.

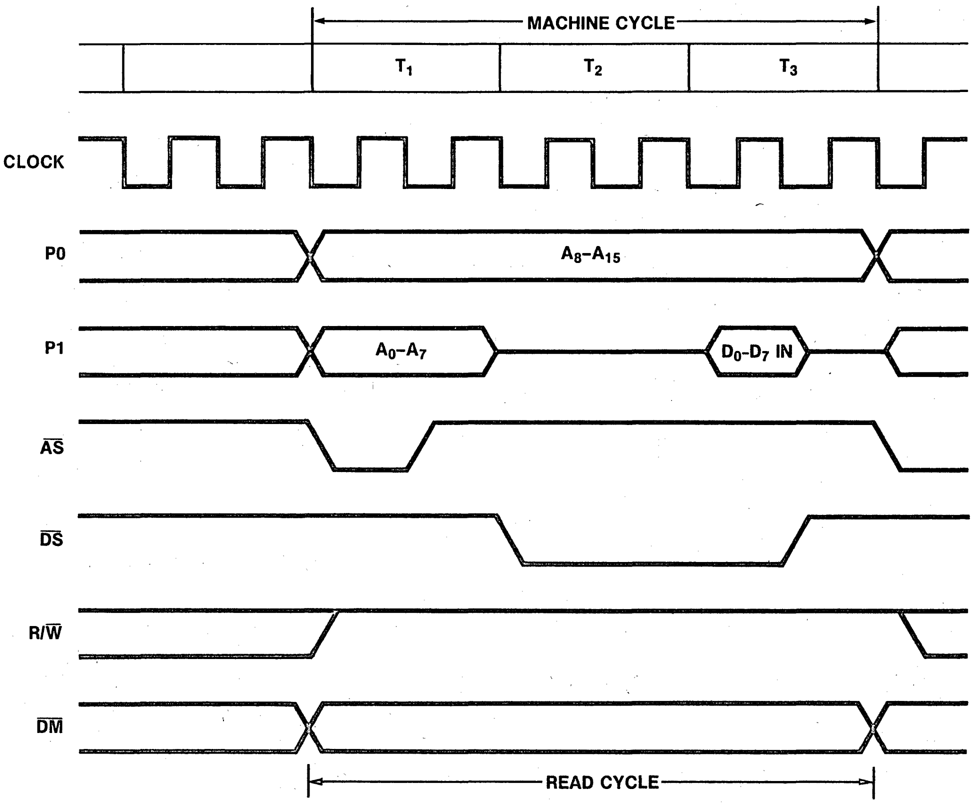


Figure 6-6a. External Instruction Fetch, or Memory Read Cycle

6.6.1 Address Strobe ($\overline{AS}$)

All transactions start with $\overline{AS}$ driven Low and then raised High by the Z8. The rising edge of $\overline{AS}$ indicates that R/$\overline{W}$, $\overline{DM}$, and the addresses output from Ports 0 and 1 are valid. The addresses output via Port 1 remain valid only during MnT1 and typically need to be latched using $\overline{AS}$, whereas Port 0 address outputs remain stable throughout the machine cycle.

6.6.2 Data Strobe

The Z8 uses $\overline{DS}$ to time the actual data transfer. For Write operations (R/$\overline{W}$ = Low), a Low on $\overline{DS}$ indicates that valid data is on the Port 1 AD_0-AD_7 lines. For Read operations, (R/$\overline{W}$ = High), the Address/Data bus is placed in a high-impedance state before driving $\overline{DS}$ Low so that the addressed device can put its data on the bus. The Z8 samples this data prior to raising $\overline{DS}$ High.

6.6.3 External Memory Operations

Whenever the Z8 is configured for external memory operation, the addresses of all internal program memory references appear on the external bus. This should have no effect on the external system since the bus control lines, $\overline{DS}$ and R/$\overline{W}$, remain in their inactive High state. $\overline{DS}$ and R/$\overline{W}$ become active only during external memory references.

CAUTION

Do not use LDC, LDCI, LDE or LDEI to write to internal program memory. The execution of these instructions causes the Z8 to assume that an external write operation is being performed and this will activate control signals $\overline{DS}$ and R/$\overline{W}$.

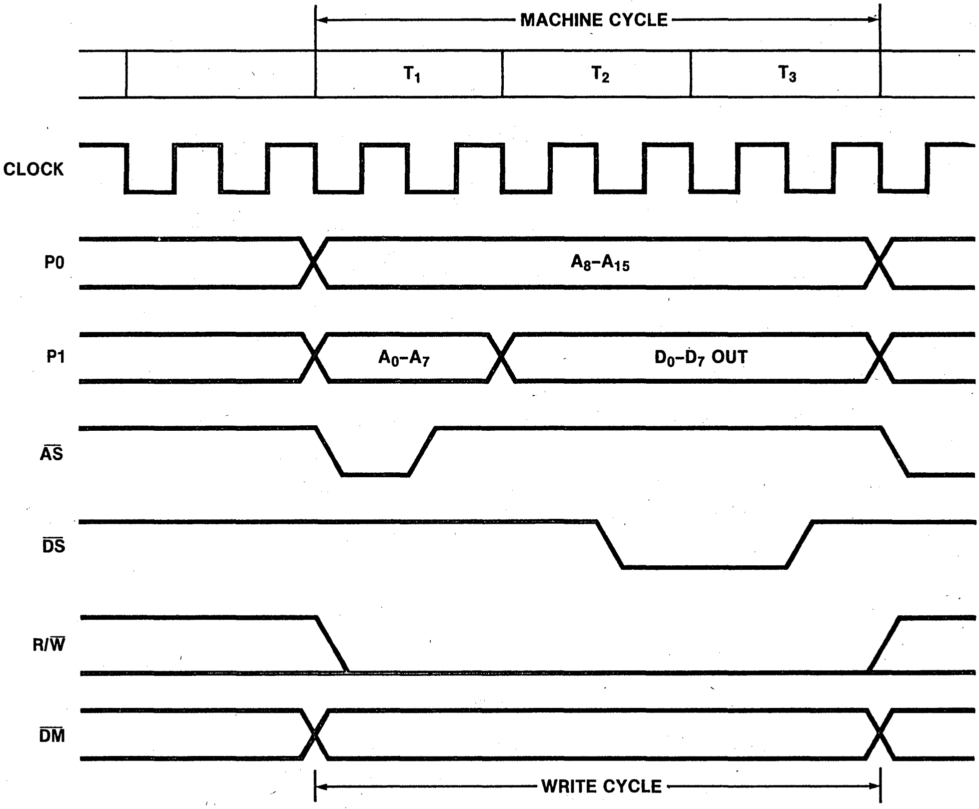


Figure 6-6b. External Memory Write Cycle

6.7 SHARED BUS

Port 1, along with $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, and Port 0 nibbles configured as address lines, can be placed in a high-impedance state, allowing the Z8601 or the Z8611 to share common resources with other bus masters. This shared bus mode is under software control and is programmed by setting Port 0-1 Mode register bits D_4 and D_3 both to 1 (Figure 6-7).

Data transfers can be controlled by assigning, for example, $P3_3$ as a Bus Acknowledge input and $P3_4$ as a Bus Request output. Bus Request/Acknowledge control sequences must be software driven.

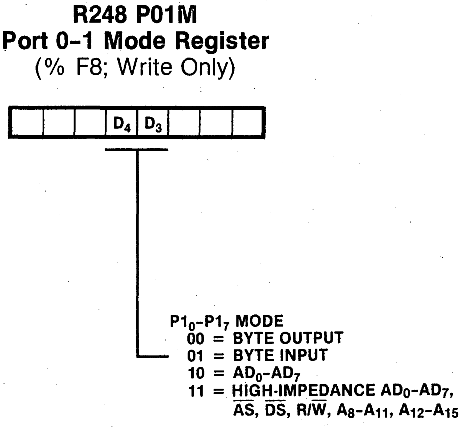


Figure 6-7. Shared Bus Operation

6.8 EXTENDED BUS TIMING

The Z8601 and Z8611 can accommodate slow memory access times by automatically inserting an additional state time (Tx) into the bus cycle. This stretches the $\overline{DS}$ timing by two clock periods, though internal memory access time is not affected. Timing is extended by setting bit D_5 in the Port 0-1 Mode register to 1 (Figure 6-8).

Figures 6-9a and 6-9b illustrate extended memory Read and Write cycles.

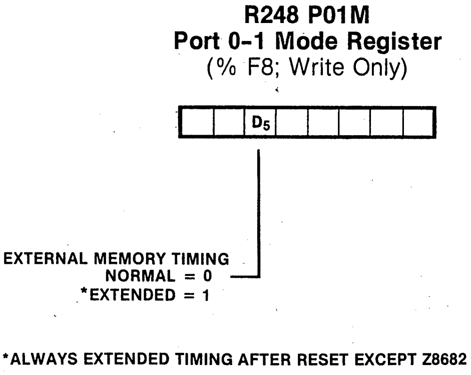


Figure 6-8. Extended Bus Timing

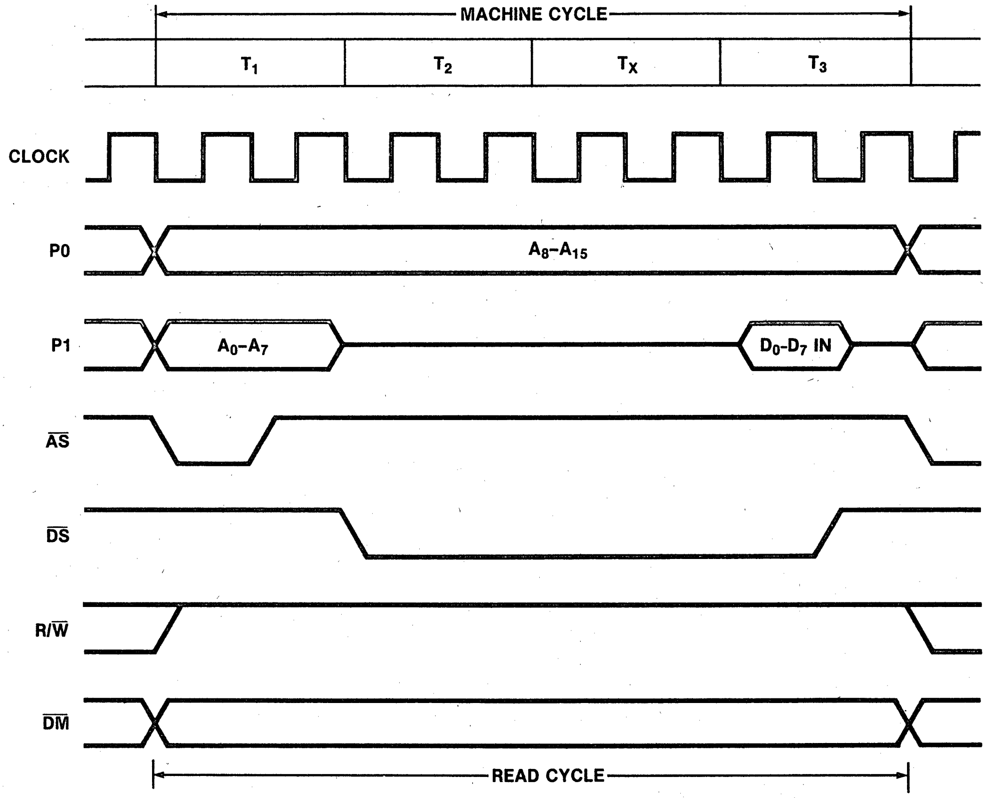


Figure 6-9a. Extended External Instruction Fetch, or Memory Read Cycle

6.9 INSTRUCTION TIMING

The high throughput of the Z8 is due, in part, to the use of instruction pipelining, in which the instruction fetch and execution cycles are overlapped. During the execution of an instruction the opcode of the next instruction is fetched. This is illustrated in Figure 6-10.

Figures 6-11 and 6-12 show typical instruction cycle timing for instructions fetched from external memory. (It should be noted that all instruction fetch cycles have the same machine timing regardless of whether memory is internal or external.) For those instructions that require execution time longer then that of the overlapped fetch, or instructions that reference program or data memory as part of their execution, the pipe must be flushed. In order to calculate the execution time of a program, the internal clock periods shown in the cycles column of the instruction formats in Section 5.4 should be added together. The cycles are equal to one-half the crystal or input clock rate.

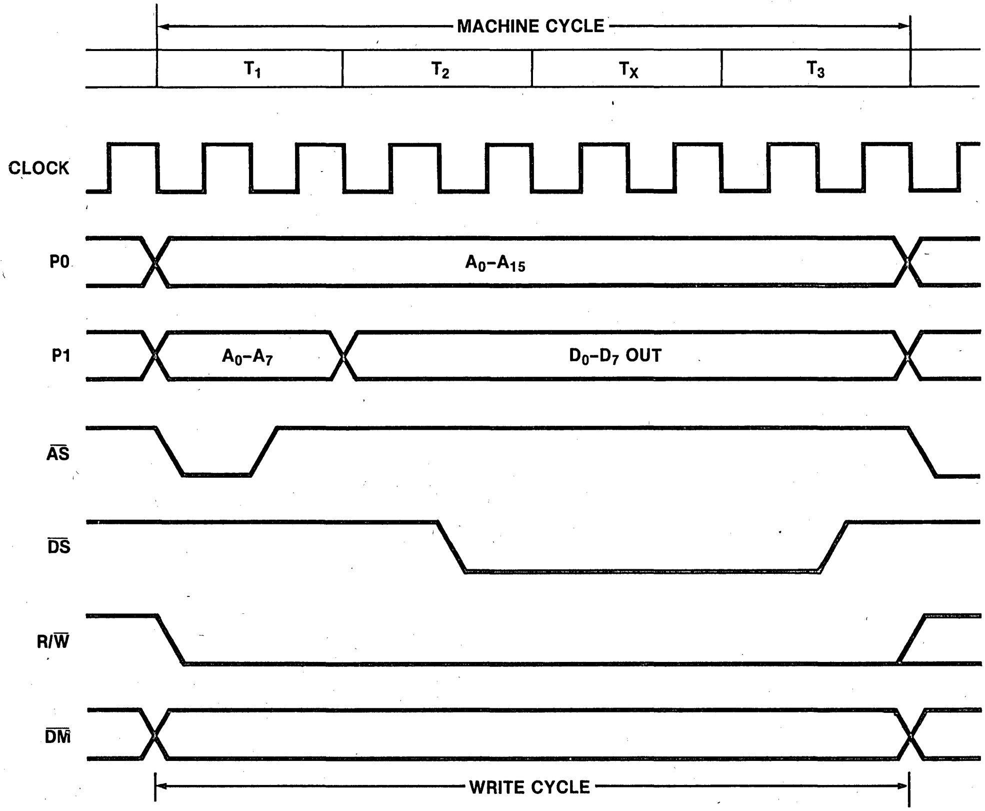


Figure 6-9b. Extended External Memory Write Cycle

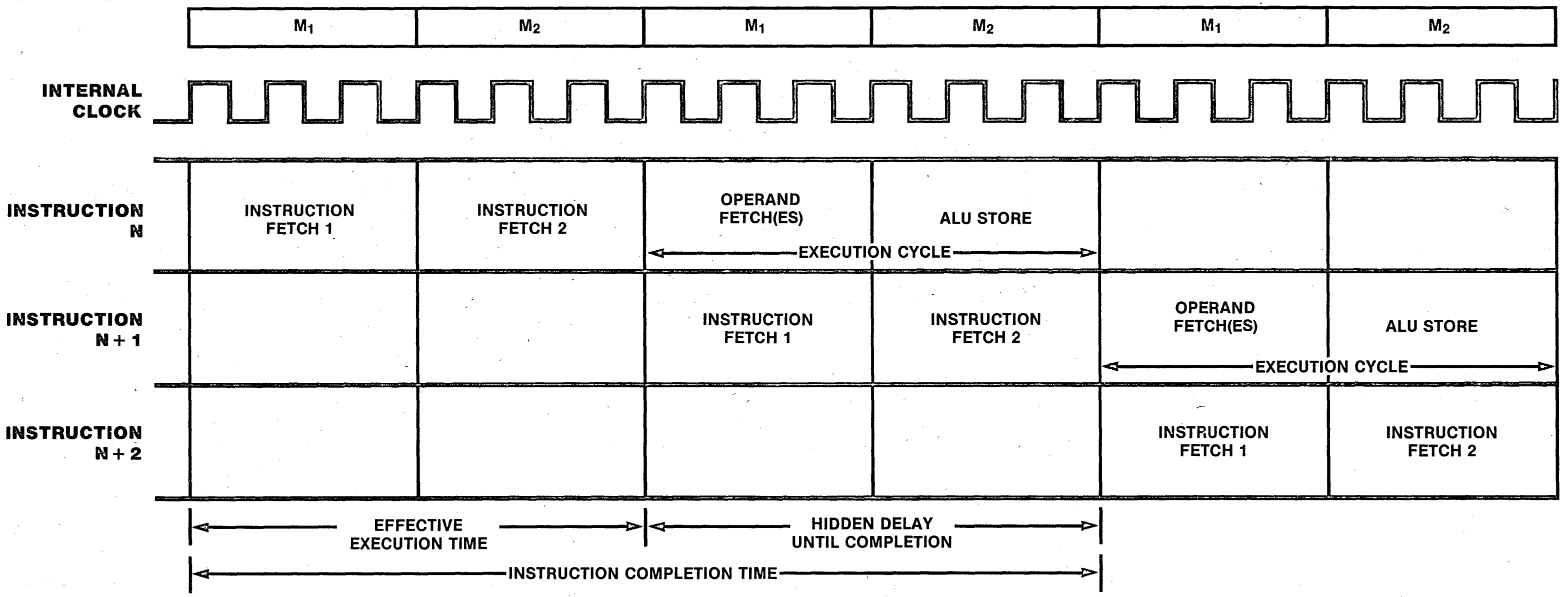


Figure 6-10. Instruction Pipelining

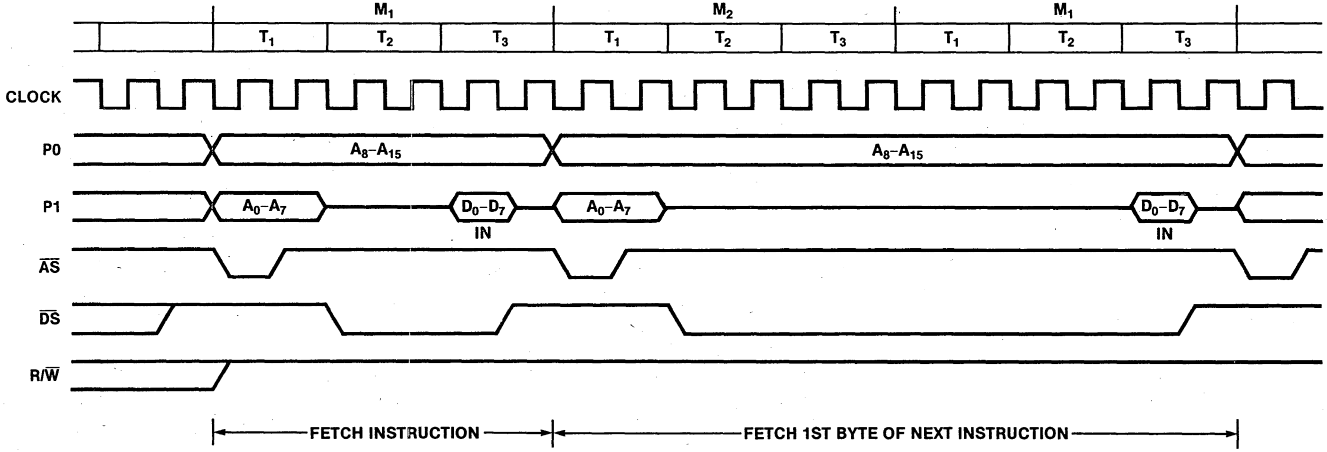


Figure 6-11. Instruction Cycle Timing (One Byte Instructions)

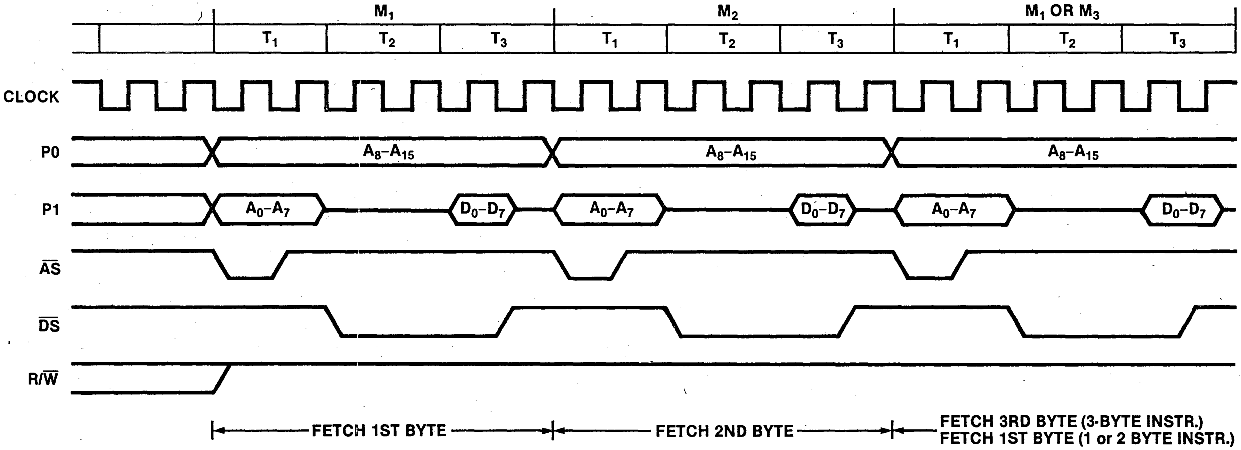
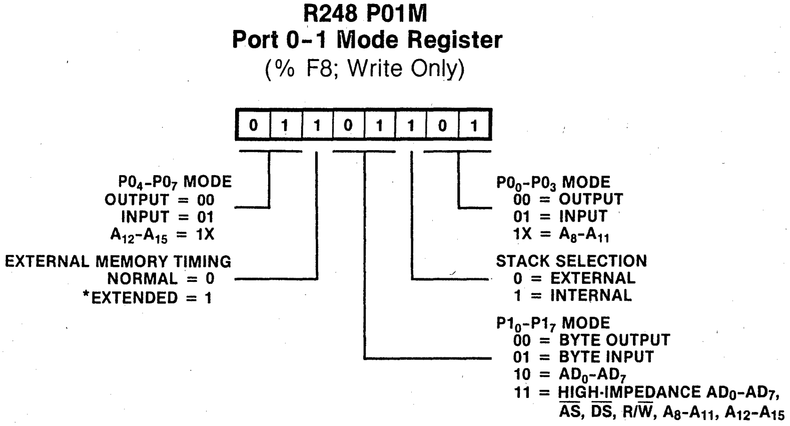


Figure 6-12. Instruction Cycle Timing (Two and Three Byte Instructions)

6.10 RESET CONDITIONS

After a hardware reset, Ports 0 and 1 are configured as input ports, memory and stack are internal, extended timing is set and DM is inactive. Figure 6-13 shows the binary values reset into P01M.



*ALWAYS EXTENDED TIMING AFTER RESET EXCEPT Z8682

Figure 6-13. Ports 0 and 1 Reset

Chapter 7
External Interface
(Z8681, Z8682)

7.1 INTRODUCTION

The ROMless versions of the Z8 microcomputer have 40 external pins, of which 24 are programmable I/O pins. Of the remaining 16 pins, 8 form an Address/Data bus and the others are used for power and control. Up to 8 I/O pins can be programmed as additional address lines to be used for external memory interface.

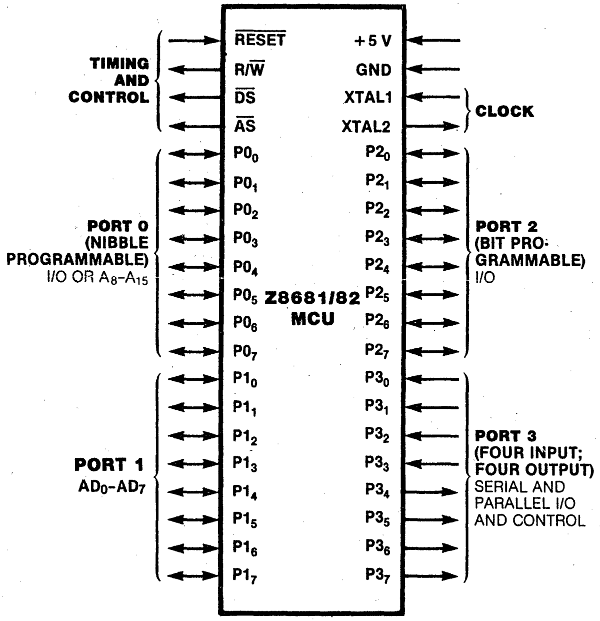
7.2 PIN DESCRIPTIONS

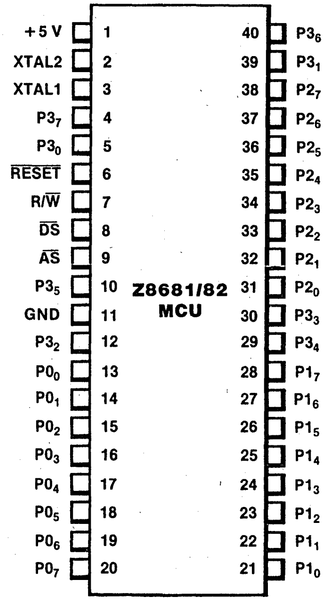
$\overline{\text{AS}}$. Address Strobe (output, active Low, pin 9). Address Strobe is pulsed Low once at the beginning of each machine cycle. The rising edge of $\overline{\text{AS}}$ indicates that addresses, Read/Write (R/$\overline{\text{W}}$), and Data Memory ($\overline{\text{DM}}$) signals are valid when output for program or data memory transfers.

$\overline{\text{DS}}$. Data Strobe (output, active Low, pin 8). Data Strobe provides the timing for data movement to or from Port 1 for each memory transfer. During a Write cycle, data out is valid at the leading edge of $\overline{\text{DS}}$. During a Read cycle, data in must be valid prior to the trailing edge of $\overline{\text{DS}}$.

R/$\overline{\text{W}}$. Read/Write. (output, pin 7). Read/Write determines the direction of data transfer for memory transactions. R/$\overline{\text{W}}$ is Low when writing to program or data memory, and High for all other transactions.

$P0_1$-$P0_7$. Address/Data Port (inputs/outputs, TTL-compatible, pins 13-20). Port 1 is permanently configured as a multiplexed Address/Data memory interface. The lower eight address lines (A_0-A_7) are multiplexed with data (D_0-D_7).





$P0_0$-$P0_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. I/O Port Lines (inputs/outputs, TTL-compatible). These 24 I/O lines are divided into 3 8-bit I/O ports that can be configured under program control for I/O or memory interface. Individual lines of a port are denoted by the second digit of the port number. For example, $P3_0$ refers to bit 0 of Port 3.

$\overline{\text{RESET}}$. Reset (input, active Low, pin 6). $\overline{\text{RESET}}$ initializes the Z8681/82. When $\overline{\text{RESET}}$ is deactivated, program execution begins from external program location %C for the Z8681 and location %812 for the Z8682. If held Low, $\overline{\text{RESET}}$ acts as a register file protect during power-down and power-up sequences.

XTAL1, XTAL2. Crystal 1, Crystal 2 (oscillator input and output, pins 3 and 2). These pins connect a parallel resonant crystal or an external source to the on-board clock oscillator and buffer.

7.3 CONFIGURING PORT 0

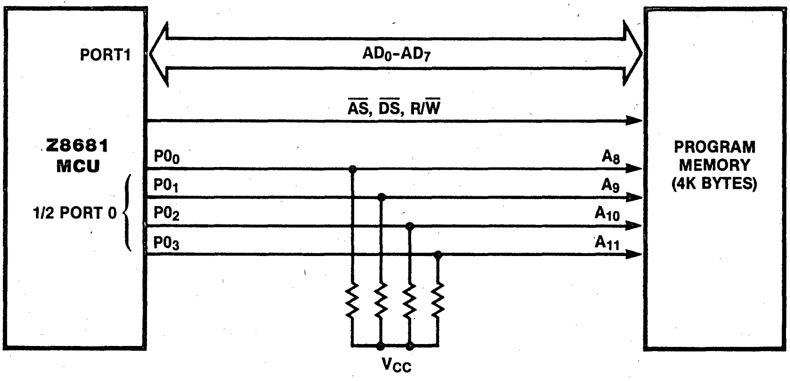
The minimum bus configuration uses Port 1 as a multiplexed Address/Data port (AD_0-AD_7) allowing access to 256 bytes of memory. In this configuration, the eight low order address bits (A_0-A_7) are multiplexed with the data (D_0-D_7).

Port 0 can be programmed to provide either four additional address lines (A_8-A_{11}) which increases the addressable memory to 4K bytes, or eight additional address lines (A_8-A_{15}) which increases the addressable memory to 64K bytes for the Z8681 and 62K bytes for the Z8682. Refer to Chapter 3, Figures 3-5 and 3-6, for the memory maps.

In the Z8681, Port 0 lines intended for use as address lines are automatically configured as inputs after a Reset. These lines therefore float and their logic state remains unknown until an initialization routine configures Port 0. In the Z8682, Port 0 lines are configured as address lines A_8-A_{15} following a Reset.

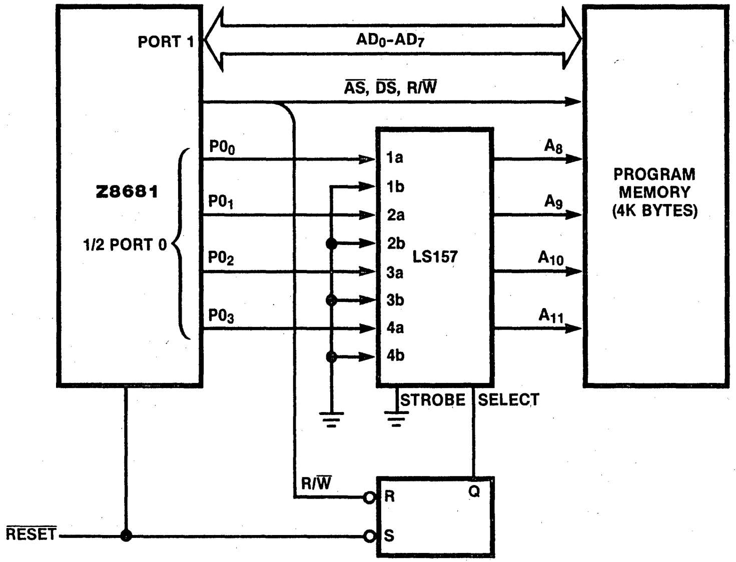
7.3.1 Z8681 Initialization

The initialization routine must reside within the first 256 bytes of executable code and must be physically mapped into memory by forcing the port 0 address lines to a known state. Figures 7-3 and 7-4 illustrate how a 4K byte memory space can be addressed.



The initialization routine is mapped in the top 256 bytes of program memory. Depending on the application, the interrupt vectors may need to be written in the first 12 byte locations of program memory by the initialization routine.

Figure 7-3. Example Z8681/Memory Interface



The initialization routine is mapped in the first 256 bytes of program memory. Any memory write operation will cause the flip-flop to select Port 0 outputs as addresses.

Figure 7-4. Example Z8681/Memory Interface

Port 0 is programmed for memory operation by writing the appropriate bits in the Port 0-1 Mode register (Figure 7-5). The proper port initialization sequence is:

- Load Port 0 with initial address value.
- Configure Port 0-1 Mode register.
- Fetch the next three bytes without changing the address in Port 0. (This is necessary due to instruction pipelining.)

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)

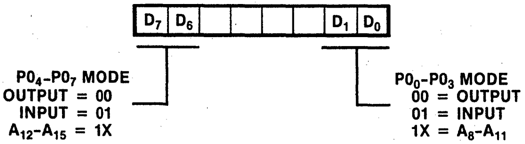


Figure 7-5. Z8681 Port 0 Memory Operation

The lower nibble of Port 0 can be defined as address lines A_8-A_{11}, by setting D_1 to 1. Similarly, setting D_7 to 1 defines the upper nibble of Port 0 as address lines A_{12}-A_{15}.

Whenever Port 0 is configured to output address lines A_{12}-A_{15}, A_8-A_{11} must also be selected as address lines.

7.3.2 Z8682 Initialization

The Z8682 must be operated in Test mode only. Section 8.4 gives a complete description of the proper technique for entering Test mode.

The user initialization routine must begin at location %812 and must reside in memory fast enough for normal memory timing. In the Z8682, the user is not protected from reconfiguring Port 1 by writing to R248 (P01M). Therefore whenever a write is made to P01M, the value 10 (binary) must be written to bits D_4 and D_3. Any other value will cause complete loss of program control.

The lower nibble of Port 0 can be defined as address lines A_8-A_{11}, by setting D_1 to 1. Similarly, setting D_7 to 1 defines the upper nibble of Port 0 as address lines A_{12}-A_{15}.

Whenever Port 0 is configured to output address lines A_{12}-A_{15}, A_8-A_{11} must also by selected as address lines.

7.3.3 Read/Write Operations

If Port 0 is configured for address lines A_7-A_{15}, it can no longer be used as a register; however, if only the lower nibble of Port 0 is defined as address lines A_8-A_{11}, the upper nibble is still addressable as an I/O register. When only the lower nibble is defined as address outputs, reading Port 0 returns XF, where X equals the data in bits D_4-D_7. Writing to Port 0 transfers data to the I/O nibble only.

The instruction used to change the mode of Port 0 should not be immediately followed by an instruction that performs a stack operation, because this will cause indeterminate program flow. In addition, after setting the mode of Port 0 for memory, the next three bytes must be fetched without changing the value of the upper byte of the Program Counter (PC).

7.4 EXTERNAL STACKS

The Z8681/82 architecture supports stack operations in either the register file or data memory. A stack's location is determined by bit D_2 in the Port 0-1 Mode register. For example, if D_2 is set to 0, the stack is in external data memory (Figure 7-7).

The instruction used to change the stack selection bit should not be immediately followed by the instructions RET or IRET, because this will cause indeterminate program flow.

7.5 DATA MEMORY

The two memory spaces, data and program, can be addressed as a single memory space or as two separate spaces of equal size; i.e. 64K bytes each for the Z8681 and 62K bytes each for the Z8682. If the memory spaces are separated, program memory and data memory are logically selected by Data Memory select output ($\overline{DM}$). $\overline{DM}$ is made available on Port 3, line 4 ($P3_4$) by setting bits D_4 and D_3 in the Port 3 Mode register to 10 or 01 (Figure 7-8). $\overline{DM}$ is active Low during the execution of the LDE, LDEI instructions. $\overline{DM}$ is also active Low during the execution of CALL, POP, PUSH, RET and IRET instructions if the stack resides in memory.

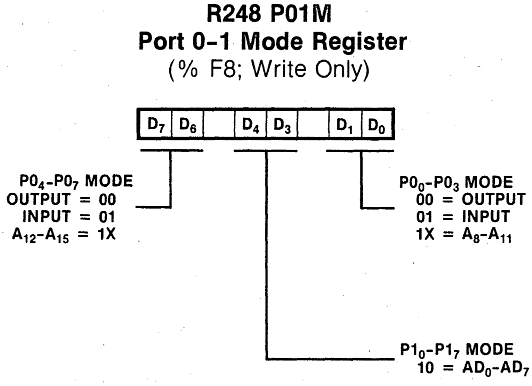


Figure 7-6. Z8682 Port 0 Memory Operation

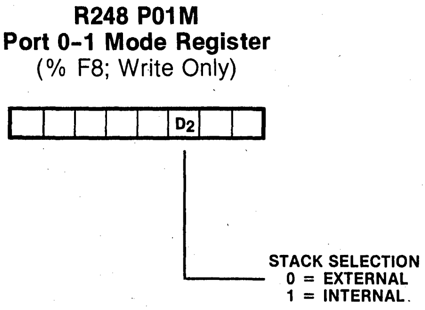


Figure 7-7. External Stack Operation

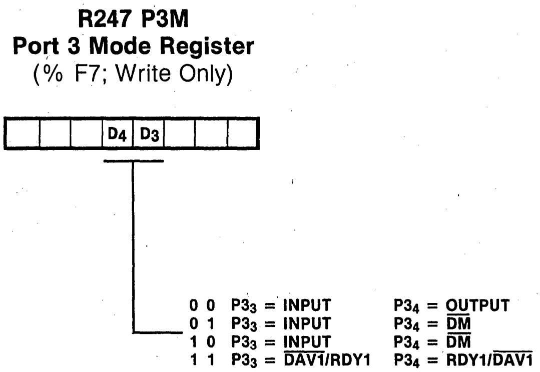


Figure 7-8. Port 3 Data Memory Operation

7.6 BUS OPERATION

Typical data transfers between the Z8681/82 and memory are illustrated in Figure 6-6. Machine cycles can vary from six to twelve clock periods depending on the operation being performed. The notations used to describe the basic timing periods of the Z8681/82 are: machine cycles (Mn), timing states (Tn), and clock periods. All timing references are made with respect to the output signals $\overline{AS}$ and $\overline{DS}$. The clock is shown for clarity only and does not have a specific timing relationship with other signals.

7.6.1 Address Strobe ($\overline{AS}$)

All transactions start with $\overline{AS}$ driven Low and then raised High by the Z8681/82. The rising edge of $\overline{AS}$ indicates that R/$\overline{W}$, $\overline{DM}$ (if used), and the addresses output from Ports 0 and 1 are valid. The addresses output via Port 1 remain valid only during MnT1 and typically need to be latched using $\overline{AS}$, whereas Port 0 address outputs remain stable throughout the machine cycle.

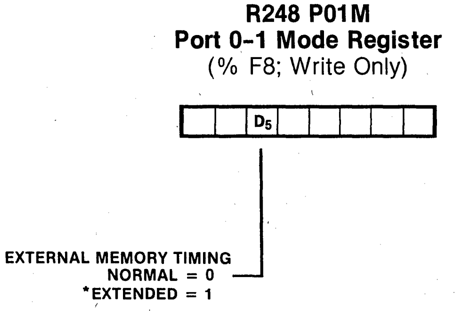
7.6.2 Data Strobe ($\overline{DS}$)

The Z8681/82 uses $\overline{DS}$ to time the actual data transfer. For Write operations (R/$\overline{W}$ = Low), a Low on $\overline{DS}$ indicates that valid data is on the Port 1 AD_0-AD_7 lines. For Read operations (R/$\overline{W}$ = High), the Address/Data bus is placed in a high-impedance state before driving $\overline{DS}$ Low so that the addressed device can put its data on the bus. The Z8681/82 samples this data prior to raising $\overline{DS}$ High.

7.7 EXTENDED BUS TIMING

The Z8681/82 accommodates slow memory access times by automatically inserting an additional software-controlled state time (Tx). This stretches the $\overline{DS}$ timing by two clock periods. Timing is extended by setting bit D_5 in the Port 0-1 Mode register to 1 (Figure 7-9).

Refer to Section 6.7 for other figures pertaining to extended bus timing.



*ALWAYS EXTENDED TIMING AFTER RESET EXCEPT Z8682

Figure 7-9. Extended Bus Timing

7.8 INSTRUCTION TIMING

The high throughput of the Z8681/82 is due, in part, to the use of instruction pipelining, in which the instruction fetch and execution cycles are overlapped. During the execution of the current instruction the opcode of the next instruction is fetched as illustrated in Figure 6-10.

Figures 6-11 and 6-12 show typical instruction cycle timing for instructions fetched from memory. For those instructions that require execution time longer than that of the overlapped fetch, or reference program or data memory as part of their execution, the pipe must be flushed. In order to calculate the execution time of a program, the internal clock periods shown in the cycles column of the instruction formats in Section 5.6 should be added together. The cycles are equal to one-half the crystal or input clock rate.

7.9 Z8681 RESET CONDITIONS

After a hardware reset, Port 0 is configured as input port, extended timing is set to accommodate slow memory access during the configuration routine, $\overline{DM}$ is inactive, and the stack resides in the register file. Figure 7-10 shows the binary values reset into P01M.

7.10 Z8682 RESET CONDITIONS

After a hardware reset, Port 0 is configured as address lines A_8-A_{15}, memory timing is normal, $\overline{DM}$ is inactive, and the stack resides in the register file. Figure 7-11 shows the binary values reset into P01M.

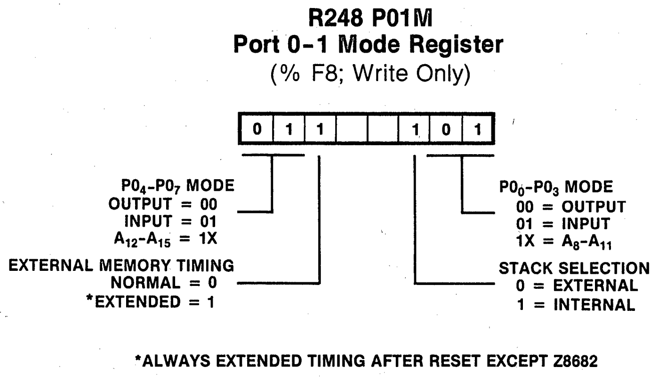


Figure 7-10. Z8681 Port 0 and 1 Reset Conditions

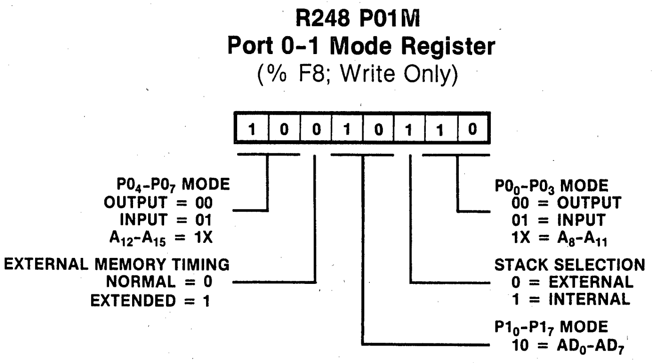


Figure 7-11. Z8682 Ports 0 and 1 Reset Conditions

Chapter 8
Reset and Clock

8.1 RESET

This section describes Z8 reset conditions, reset timing, and register initialization procedures.

A system reset overrides all other operating conditions and puts the Z8 into a known state. To initialize the chip's internal logic, the reset input must be held Low for at least 18 clock periods.

While $\overline{RESET}$ is Low, $\overline{AS}$ is output at the internal clock rate (XTAL frequency divided by 2), $\overline{DS}$ is forced Low and R/$\overline{W}$ remains High. (Zilog Z-BUS compatible peripherals use the $\overline{AS}$ and $\overline{DS}$ coincident Low state as a peripheral reset function.) In addition, interrupts are disabled, Ports 0, 1, and 2 are put in input mode, and %C is loaded into the Program Counter.

The hardware Reset initializes the control and peripheral registers, as shown in Table 8.1. Specific reset values are shown by 1s or 0s, while bits whose states are unknown are indicated by the

Table 8-1. Control and Peripheral Register Reset Values

Register	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Comments
%F0 Serial I/O	undefined								
%F1 Timer Mode	0	0	0	0	0	0	0	0	Counter/Timers stopped
%F2 Counter/Timer 1	undefined								
%F3 T1 Prescaler	u	u	u	u	u	u	0	0	Single Pass count mode, external clock source
%F4 Counter/Timer 0	undefined								
%F5 T0 Prescaler	u	u	u	u	u	u	u	0	Single Pass count mode
%F6 Port 2 Mode	1	1	1	1	1	1	1	1	All lines input
%F7 Port 3 Mode	0	0	0	0	0	0	u	0	Port 2 open-drain $P3_0$-$P3_3$ input; $P3_4$-$P3_7$ output
%F8 Port 0-1 Mode Z8601/Z8611	0	1	1	0	1	1	0	1	Ports 0 and 1 inputs; internal stack; extended external memory timing
%F8 Port 0-1 Mode Z8681	0	1	1	1	0	1	0	1	Port 0 inputs Port 1 Address/Data; internal stack; extended external memory timing
%F8 Port 0-1 Mode Z8682	1	0	0	1	0	1	1	0	Port 0 Address Port 1 Address/Data internal stack; normal external memory timing
%F9 Interrupt Priority	undefined								
%FA Interrupt Request	u	u	0	0	0	0	0	0	Reset all interrupt disabled
%FB Interrupt Mask	0	u	u	u	u	u	u	u	Interrupts disabled
%FC Flags	undefined								
%FD Register Pointer	undefined								
%FE Stack Pointer	undefined								Most significant byte
%FF Stack Pointer	undefined								Least significant byte

letter u. Registers that are not predictable are listed as undefined.

Program execution starts four clock cycles after RESET has returned High. The initial instruction fetch is from location %C. Figure 8-1 shows reset timing.

After a reset, the first program executed should be a routine that initializes the control registers to the required system configuration. The Interrupt Request register remains inactive until an EI instruction is executed. This guarantees that program execution can proceed free from interrupts.

RESET is the input of a Schmitt trigger circuit. To form the internal reset line, the output of the trigger is synchronized with the internal clock (xtal frequency divided by 2). The clock must therefore be running for RESET to function. For a power-up reset operation, the RESET input must be held Low for at least 50 ms after the power supply is within tolerance. This allows the on-board clock oscillator to stabilize. An internal pull-up combined with an external capacitor of 1 ^{e}F provides enough time to properly reset the Z8 (Figure 8-2).

8.2 CLOCK

The Z8 derives its timing from on-board clock circuitry connected to pins XTAL1 and XTAL2. The clock circuitry consists of an oscillator, a divide-by-2 shaping circuit, and a clock buffer. Figure 8-3 illustrates the clock circuitry. The oscillator's input is XTAL1; its output is XTAL2. The clock can be driven by a crystal, a ceramic resonator, or an external clock source.

Crystals and ceramic resonators should have the following characteristics to ensure proper oscillator operation:

Cut: AT (crystal only)
Mode: Parallel, Fundamental
Output Frequency: 1 MHz - 12 MHz
Resistance: 100 ohms max

Depending on operation frequency, the oscillator may require the addition of capacitors C1 and C2 (shown in Figure 8-4). The range of recommended capacitance values is dependent on crystal specifications but should not exceed 15 pF. The ratio of the values of C1 to C2 can be adjusted to shift the operating frequency of the circuit by approximately ±.005%.

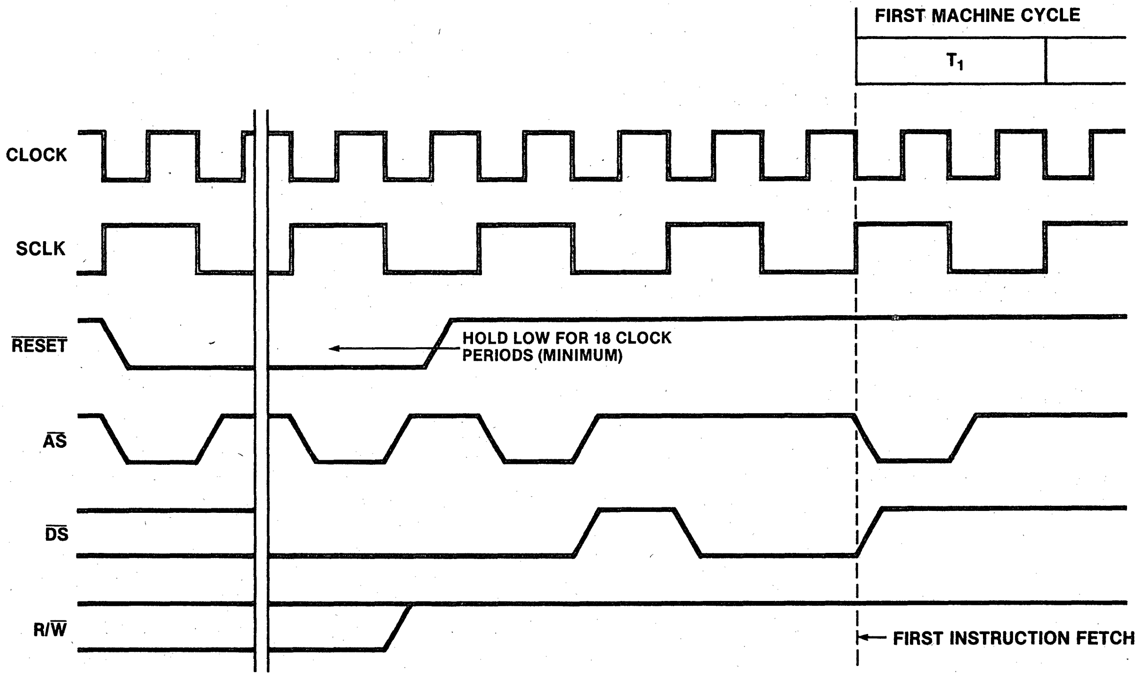


Figure 8-1. Reset Timing

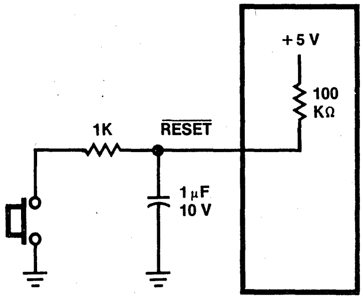


Figure 8-2. Power-Up Reset Circuit

When an external frequency source is used, it must drive both XTAL1 and XTAL2 inputs. This differential drive requirement arises from the loading on the oscillator output (XTAL2) without the reactive feedback network of a crystal or resonator. A typical clock interface circuit is shown in Figure 8-5.

The capacitors shown represent the maximum parasitic loading when using a 74LS04 driver. The pull-up resistors can be eliminated by using a 74HC04 driver.

8.3 POWER-DOWN OPERATION

The Z8 has a power-down option which can be used to maintain the contents of the register file with a low-power battery. The circuitry has its XTAL2 output replaced by a power supply input (V_{MM}). V_{MM} powers the general-purpose registers %04 - %7F as well as a portion of the reset logic that protects the register file. When V_{MM} is maintained at 3 to 5 V, this power-down option preserves the contents of the general-purpose registers whenever V_{CC} is removed. During normal operation, V_{MM} provides +5 V along with V_{CC}.

The following sequence is necessary to preserve data:

- Power failure must be externally detected early enough for a software routine to store the required data that is not already in the register file. An interrupt is typically used for this purpose.

- RESET must be held Low after data is saved and during the removal of V_{CC}. RESET is a write protect input to the register file.

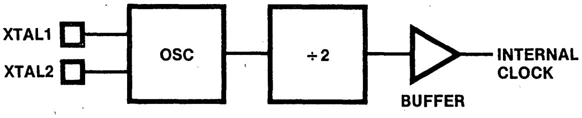


Figure 8-3. Z8 Clock Circuit

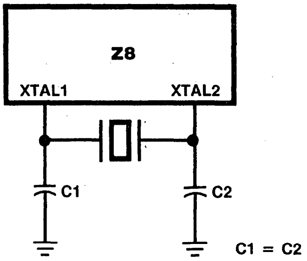


Figure 8-4. Crystal/Ceramic Resonator Oscillator

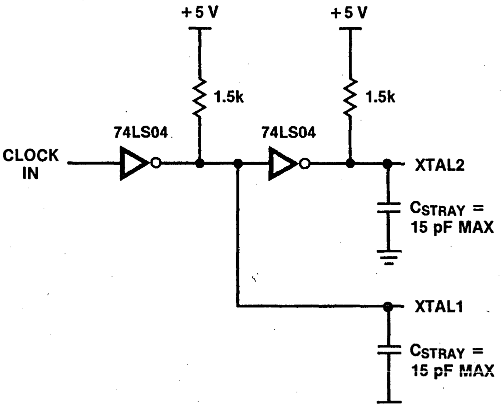


Figure 8-5. External Clock Interface

- RESET must be held Low during the power-up sequence. Again, RESET is a write protect input to the register file.

As V_{CC} powers down, on-board circuitry associated with RESET automatically protects the general-purpose registers. The circuit shown in Figure 8-2 satisfies the power-up requirement of holding RESET Low to protect the register file data.

Figure 8-6 shows the recommended circuit configuration for a battery-backed supply system.

Since XTAL2 is replaced by V_{MM}, an external clock generator must be used to input the Z8 clock via the XTAL1 input.

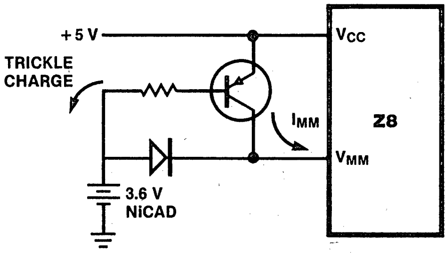


Figure 8-6. Battery-Backed Register Supply

8.4 TEST MODE

Test mode is a special mode of operation that facilitates testing of Z8 devices containing on-board ROM (Z8601 and Z8611). Test mode must also be used to reset the Z8682. When Test mode is invoked, an additional on-board ROM is mapped into the first 64 locations of program memory. Figure 8-7 shows the difference between Normal and Test modes of operation.

Test mode is entered by driving the $\overline{\text{RESET}}$ input to a voltage level of V_{CC} + 2.5 V after a normal Reset cycle (Figure 8-8). This voltage is absolutely essential for proper operation.

After entering Test mode, instructions are fetched from the internal test ROM. Port 1 is configured for Address/Data operation, followed by a JUMP to external memory location %812 for the Z8601 and Z8682, or %1012 for the Z8611. Once in external memory, diagnostic routines, invoked via the

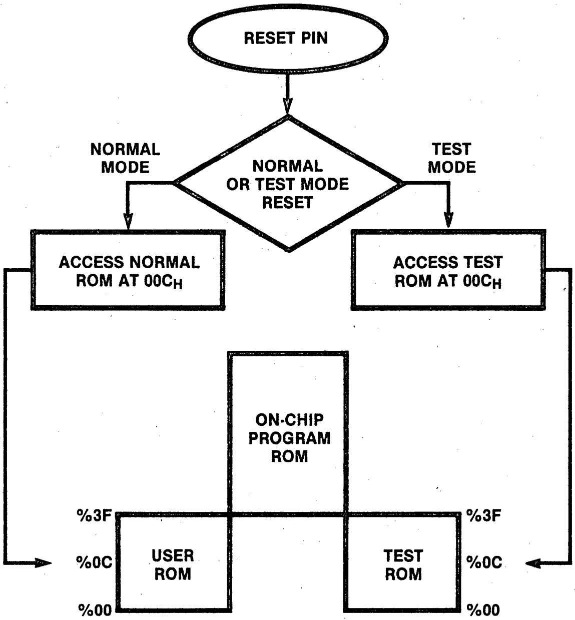


Figure 8-7. Normal and Test Mode Flow

Address/Data bus, verify the Z8's functionality. Since Port 1 is used only in Address/Data mode in this process, additional routines in the test ROM verify Port 1's I/O and Handshake modes.

Programs run with Test mode active can use the LDE instruction to access contents of the test ROM. The LDC instruction accesses the normal program ROM.

The Z8 stays in the Test mode until a normal reset occurs.

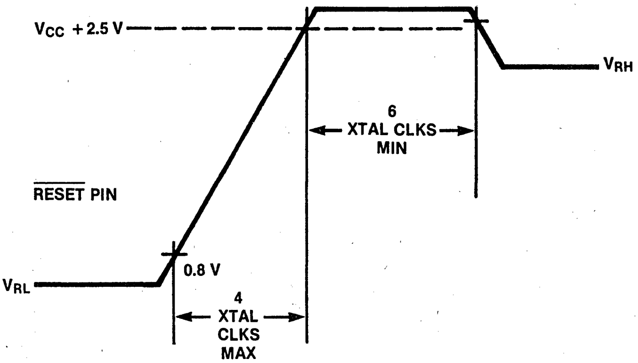
8.4.1 Interrupt Testing

To test the interrupt structure, the first twelve locations of test ROM contain interrupt vectors. Interrupt vectors in the Z8601 and Z8682 point to external memory locations %800, %803, %806, %809, %80C, and %80F; interrupt vectors in the Z8611 point to external memory locations %1000, %1003, %1006, %1008. %100C, and %100F. These interrupt vectors allow the external program to have a 2- or 3-byte JUMP instruction to each interrupt service routine.

Programs that are run with Test mode active can use the LDE instruction for accessing the contents of the Test ROM. The LDC instruction can be used for accessing the program ROM as normal.

8.4.2 ROMless Operation

ROMless operation of the Z8601 or Z8611 can be achieved by always entering Test mode after a reset. Execution begins at %812 or %1012, respectively. (The Z8682 is a modified Z8601 sold as a ROMless part.)



Note the maximum ramp for application of +7.5 VDC to RESET pin. After a minimum of 6 XTAL CLK cycles, the RESET voltage can be relaxed to VRH.

Figure 8-8. Voltage Waveform for Test Mode

Chapter 9
I/O Ports

9.1 INTRODUCTION

The Z8 has 32 lines dedicated to input and output. These lines are grouped into four 8-bit ports and are configurable as input, output, or address/data. Under software control, the ports can be programmed to provide address/data, timing, status, serial, and parallel input/output with or without handshake.

All ports have active pull-ups and pull-downs compatible with TTL loads. In addition, the pull-ups of Port 2 can be turned off for open-drain operation.

9.1.1 Mode Registers

Each port has an associated mode register which determines the port's functions and allows dynamic change in port functions during program execution. Ports and mode registers are mapped into the register file as shown in Figure 9-1.

Because of their close association, ports and mode registers are treated like any other general-purpose register. There are no special instructions for port manipulation; any instruction that addresses a register can address the ports. Data can be directly accessed in the port register, with no extra moves.

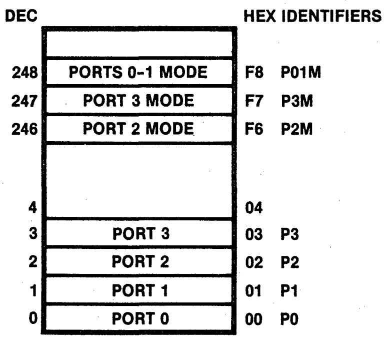


Figure 9-1. I/O Port and Port Mode Registers

9.1.2 Input and Output Registers

Each bit of Ports 0, 1, and 2 has an input register, an output register, associated buffer, and control logic. Since there are separate input and output registers associated with each port, writing to bits defined as inputs stores the data in the output register. This data cannot be read as long as the bits are defined as inputs. However, if the bits are reconfigured as output, the data stored in the output register is reflected on the output pins and can then be read. This mechanism allows the user to initialize the outputs prior to driving their loads.

Since port inputs are asynchronous to the Z8's internal clock, a Read operation could occur during an input transition. In this case, the logic level might be uncertain--somewhere between a logic 1 and 0. To eliminate this meta-stable condition, the Z8 latches the input data two clock periods prior to the execution of the current instruction. The input register uses these two clock periods to stabilize to a legitimate logic level before the instruction reads the data.

9.2 PORT 0

This section deals only with the I/O operation of Port 0. Refer to Sections 6.2 and 7.2 for a description of the port's external memory interface operation.

Port 0 is a general I/O port. Bits within each nibble can be independently programmed as inputs, outputs or address lines. Figure 9-2 shows a block diagram of Port 0. This diagram also applies to Ports 1 and 2.

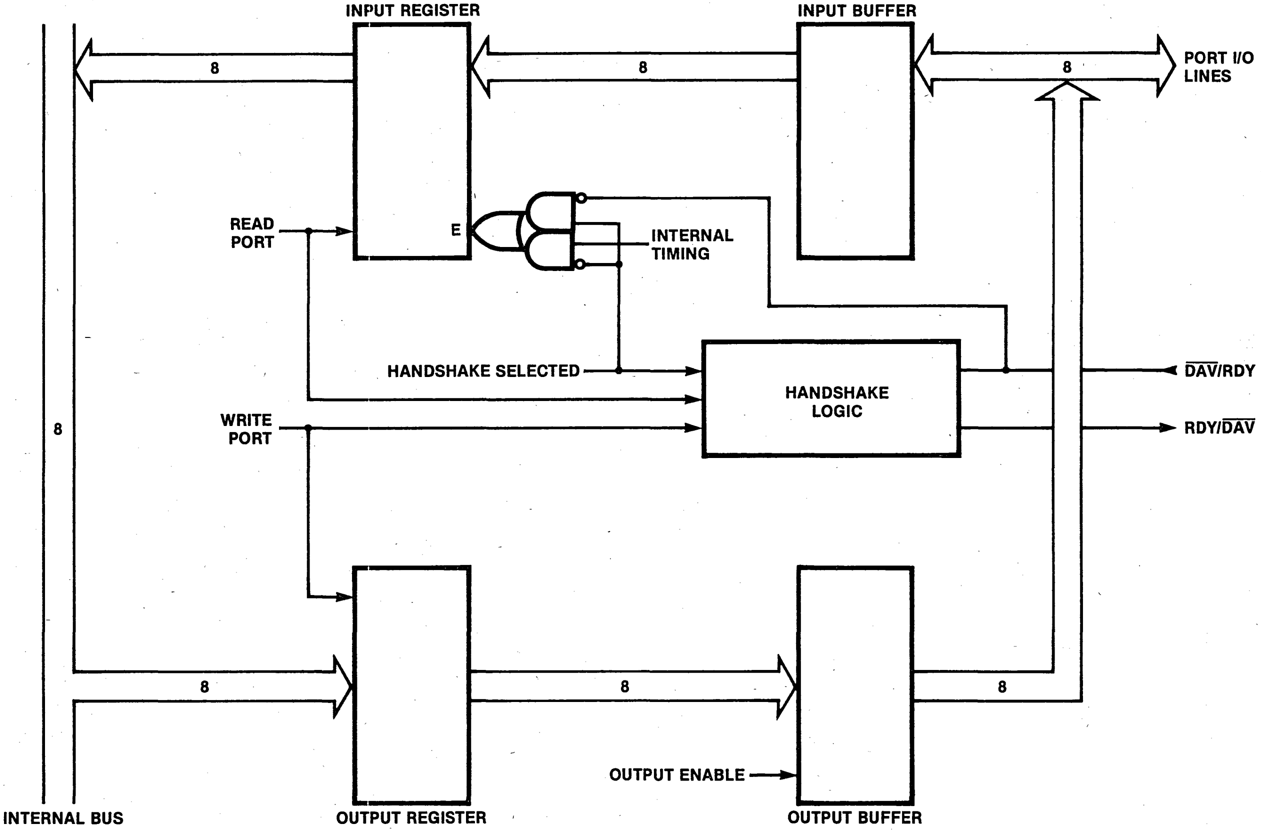


Figure 9-2. Ports 0, 1, and 2 Block Diagram

9.2.1 Read/Write Operations

In the nibble I/O mode, Port 0 is accessed as general-purpose register P0 (%00). The port is written by specifying P0 as an instruction's destination register. Writing the port causes data to be stored in the port's output register.

The port is read by specifying P0 as the source register of an instruction. When an output nibble is read, data on the external pins is returned. Under normal loading conditions this is equivalent to reading the output register. Reading a nibble defined as input also returns data on the external pins. However, input bits under handshake control return data latched into the input register via the input strobe.

The Port 0-1 Mode register bits D_1D_0 and D_7D_6 are used to configure Port 0 nibbles (Figure 9-3). The lower nibble ($P0_0$-$P0_3$) can be defined as inputs by setting bits D_1 to 0 and D_0 to 1, or as outputs by setting both D_1 and D_0 to 0. Likewise, the upper nibble ($P0_4$-$P0_7$) can be defined as inputs by setting bits D_7 to 0 and D_6 to 1, or as outputs by setting both D_6 and D_7 to 0.

9.2.2 Handshake Operation

When used as an I/O port, Port 0 can be placed under handshake control by programming the Port 3 Mode register bit D_2 to 1 (Figure 9-4). In this configuration, handshake control lines are $\overline{DAV_0}$ ($P3_2$) and RDY_0 ($P3_5$) when Port 0 is an input port, or RDY_0 ($P3_2$) and $\overline{DAV_0}$ ($P3_5$) when Port 0 is an output port.

Handshake direction is determined by the configuration (input or output) assigned to Port 0's upper nibble, $P0_4$-$P0_7$. The lower nibble must have the same I/O configuration as the upper nibble to be under handshake control. Figure 9-5 illustrates the Port 0 upper and lower nibbles, and the associated handshake lines of Port 3.

Handshake operation is discussed in detail in Section 9.6.

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)

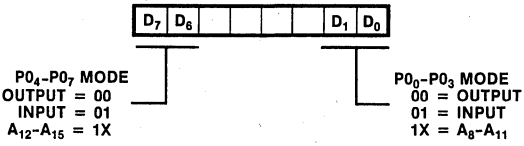


Figure 9-3. Port 0 I/O Operation

R247 P3M
Port 3 Mode Register
(% F7; Write Only)

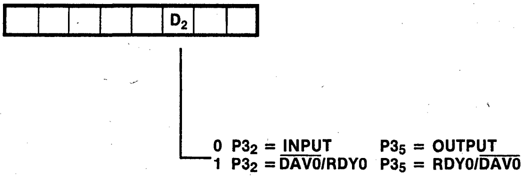

D_2
0 $P3_2$ = INPUT $P3_5$ = OUTPUT
1 $P3_2$ = $\overline{DAV0}$/RDY0 $P3_5$ = RDY0/$\overline{DAV0}$

Figure 9-4. Port 0 Handshake Operation

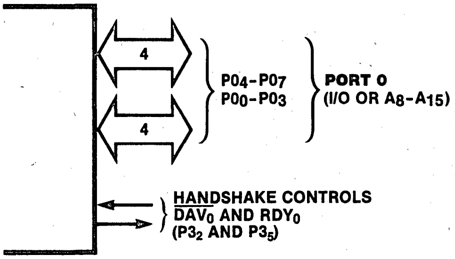


Figure 9-5. Port 0

9.3 PORT 1

This section deals only with the I/O operation of Port 1 and does not apply to the Z8681/82 ROMless devices. Refer to Sections 6.2 and 7.2 for a description of the port's external memory interface operation.

Port 1 is a general-purpose I/O port that can be programmed as a byte I/O port with or without handshake, or as an address/data port for interfacing with external memory. Refer to Figure 9-2 for a block diagram of Port 1.

9.3.1 Read/Write Operations

In byte input or byte output mode, the port is accessed as general-purpose register P1 (%01). The port is written by specifying P1 as an instruction's destination register. Writing the port causes data to be stored in the port's output register.

The port is read by specifying P1 as the source register of an instruction. When an output is read, data on the external pins is returned. Under normal loading conditions, this is equivalent to reading the output register. When Port 1 is defined as an input, reading also returns data on the external pins. However, inputs under handshake control return data latched into the input register via the input strobe.

Using the Port 0-1 Mode register, Port 1 is configured as an output port by setting bits D_4 and D_3 to 0s, or as an input port by setting D_4 to 0 and D_3 to 1 (Figure 9-6).

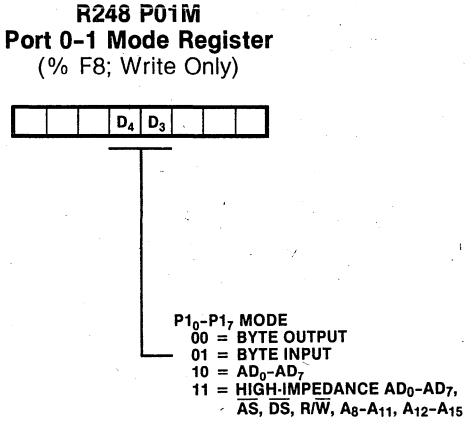


Figure 9-6. Port 1 I/O Operation

9.3.2 Handshake Operations

When used as an I/O port, Port 1 can be placed under handshake control by programming the Port 3 Mode register bits D_4 and D_3 both to 1 (Figure 9-7). In this configuration, handshake control lines are $\overline{DAV}_1$ ($P3_3$) and RDY_1 ($P3_4$) when Port 1 is an input port, or RDY_1 ($P3_3$) and $\overline{DAV}_1$ ($P3_4$) when Port 1 is an output port.

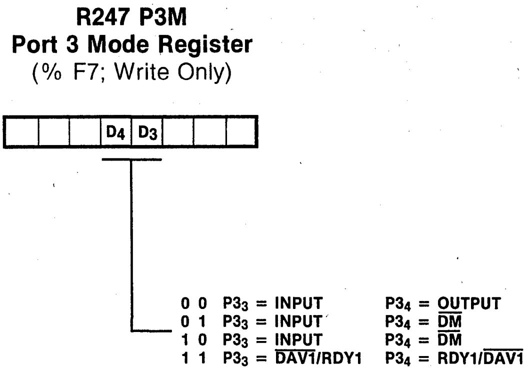


Figure 9-7. Port 1 Handshake Operation

Handshake direction is determined by the configuration (input or output) assigned to Port 1. For example, if Port 1 is an output port then handshake is defined as output. Figure 9-8 illustrates the Port 1 lines and the associated handshake lines of Port 3.

Handshake operation is discussed in detail in Section 9.6.

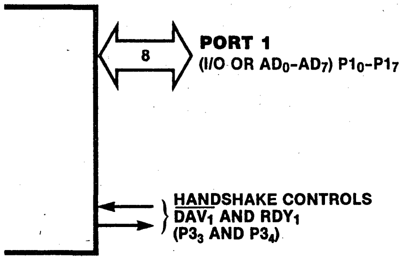


Figure 9-8. Port 1

9.4 PORT 2

Port 2 is a general-purpose port. Each of its lines can be independently programmed as input or output via the Port 2 Mode register (Figure 9-9). A bit set to a 1 in P2M configures the corresponding bit in Port 2 as an input, while a bit set to 0 determines an output line.

R246 P2M
Port 2 Mode Register
(%F6; Write Only)

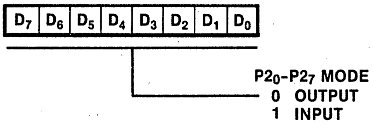


Figure 9-9. Port 2 I/O Operation

9.4.1 Read/Write Operations

Port 2 is accessed as general-purpose register P2 (%02). The port is written by specifying P2 as an instruction's destination register. Writing the port causes data to be stored in the port's output register, and reflected externally on any bit configured as an output.

The port is read by specifying P2 as the source register of an instruction. When an output bit is read, data on the external pin is returned. Under normal loading conditions, this is equivalent to reading the output register. However, if a bit of Port 2 is defined as an open-drain output, the data returned is the value forced on the output pin by the external system. This may not be the same as the data in the output register.

Reading input bits of Port 2 also returns data on the external pins. However, inputs under handshake control return data latched into the input register via the input strobe.

9.4.2 Handshake Operation

Port 2 can be placed under handshake control by programming the Port 3 Mode register (Figure 9-10). In this configuration, Port 3 lines $P3_1$ and $P3_6$ are used as the handshake control lines $\overline{DAV}_2$ and RDY_2 for input handshake, or RDY_2 and $\overline{DAV}_2$ for output handshake.

R247 P3M
Port 3 Mode Register
(% F7; Write Only)

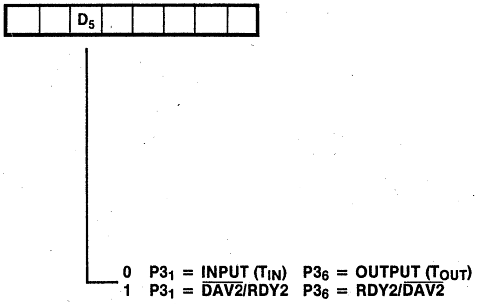

D_5
0 $P3_1$ = INPUT (T_{IN}) $P3_6$ = OUTPUT (T_{OUT})
1 $P3_1$ = $\overline{DAV2}$/RDY2 $P3_6$ = RDY2/$\overline{DAV2}$

Figure 9-10. Port 3 Handshake Operation

Handshake direction is determined by the configuration (input or output) assigned to bit 7 of Port 2. Only those bits with the same configuration as $P2_7$ will be under handshake control. Figure 9-11 illustrates Port 2's bit lines and the associated handshake lines of Port 3.

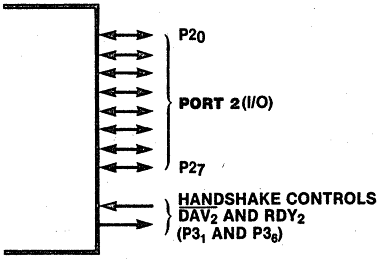


Figure 9-11. Port 2

Port 2 can also by configured to provide open-drain outputs by programming Port 3 Mode register (P3M) bit D_0 to 0 (Figure 9-12).

Regardless of the bit input/output configuration, Port 2 is always written and read as a byte-wide port.

R247 P3M
Port 3 Mode Register
(% F7; Write Only)

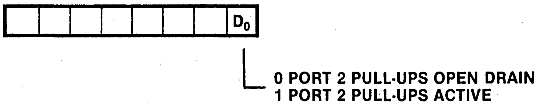


Figure 9-12. Port 2 Open-Drain Outputs

9.5 PORT 3

Port 3 differs structurally from the other three ports. Port 3 lines are fixed as four input ($P3_0$-$P3_3$) and four output ($P3_4$-$P3_7$) and do not have an input and output register for each bit. Instead, all the input lines have one input register, and output lines have an output register. Under software control, the lines can be configured as input or output, special control lines for handshake, or as I/O lines for the on-board serial and timer facilities. Figure 9-13 is a block diagram of Port 3.

9.5.1 Read/Write Operations

Port 3 is accessed as general-purpose register P3 (%03). The port is written by specifying P3 as an instruction's destination register. However,

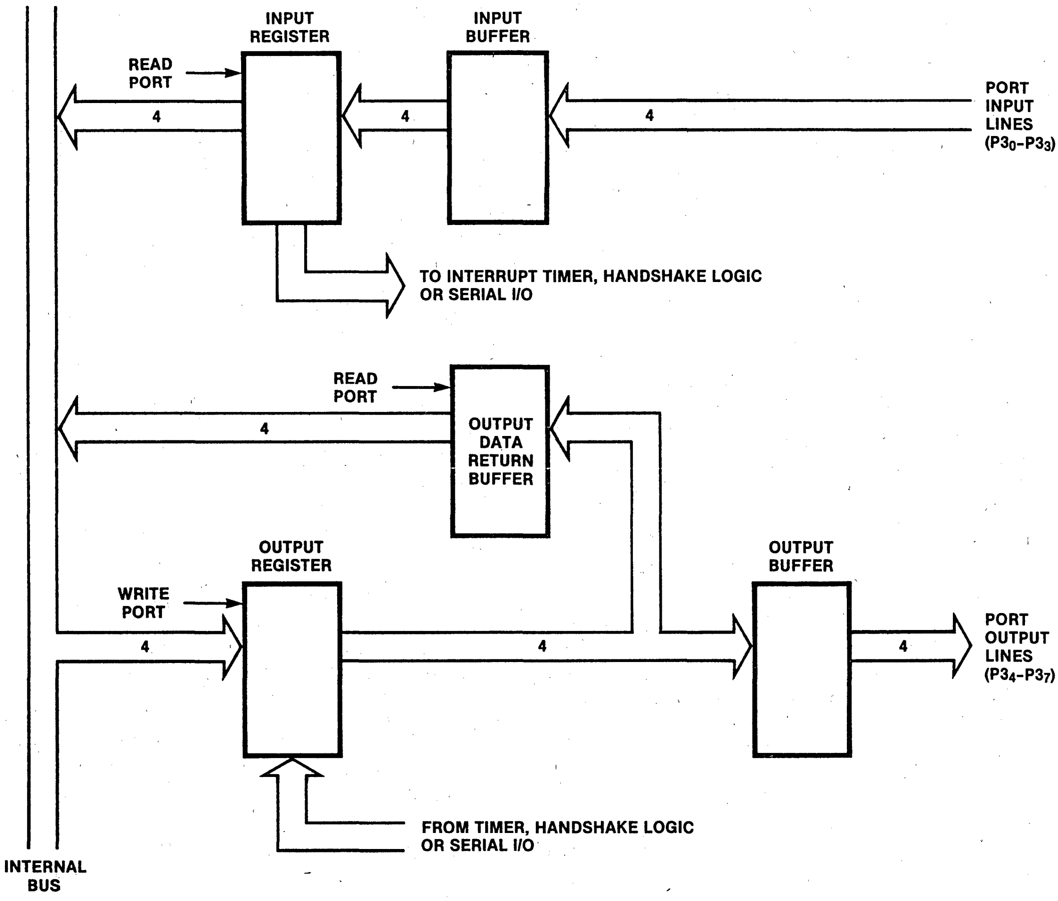

INPUT REGISTER
INPUT BUFFER
READ PORT
4
4
4
PORT INPUT LINES (P30-P33)
TO INTERRUPT TIMER, HANDSHAKE LOGIC OR SERIAL I/O
READ PORT
4
OUTPUT DATA RETURN BUFFER
OUTPUT REGISTER
OUTPUT BUFFER
WRITE PORT
4
4
4
PORT OUTPUT LINES (P34-P37)
FROM TIMER, HANDSHAKE LOGIC OR SERIAL I/O
INTERNAL BUS

Figure 9-13. Port 3 Block Diagram

Port 3 outputs cannot be written if they are used for special functions. When writing to Port 3, data is stored in the output register.

The port is read by specifying P3 as the source register of an instruction. When reading from Port 3, the data returned is both the data on the input pins and in the output register.

9.5.2 Special Functions

Special functions for Port 3 are defined by programming the Port 3 Mode register. By writing 0s in D_2-D_6, lines $P3_0$-$P3_7$ ar configured in input/output pairs (Figure 9-14). Table 9-1 shows available functions for Port 3. The special functions indicated in the table are discussed in detail in their corresponding sections in this manual.

Port 3 input lines $P3_0$-$P3_3$ always function as interrupt requests regardless of the configuration specified in the Port 3 Mode register. Unwanted interrupts must be masked off as described in Chapter 10.

Table 9.1 Port 3 Line Functions

Function	Line	Signal
Input	$P3_0$-$P3_3$	Input
Output	$P3_4$-$P3_7$	Output
Handshake	$P3_1$	$\overline{DAV_2}/RDY_2$
Inputs	$P3_2$	$\overline{DAV_0}/RDY_0$
	$P3_3$	$\overline{DAV_1}/RDY_1$
Handshake	$P3_4$	$RDY_1/\overline{DAV_1}$
Outputs	$P3_5$	$RDY_0/\overline{DAV_0}$
	$P3_6$	$RDY_2/\overline{DAV_2}$
Interrupt	$P3_0$	IRQ_3
Requests	$P3_1$	IRQ_2
	$P3_2$	IRQ_0
	$P3_3$	IRQ_1
Serial Input	$P3_0$	SI
Output	$P3_7$	SO
Counter/Timer	$P3_1$	T_{in}
	$P3_6$	T_{out}
Status	$P3_4$	$\overline{DM}$

R247 P3M
Port 3 Mode Register
(% F7; Write Only)

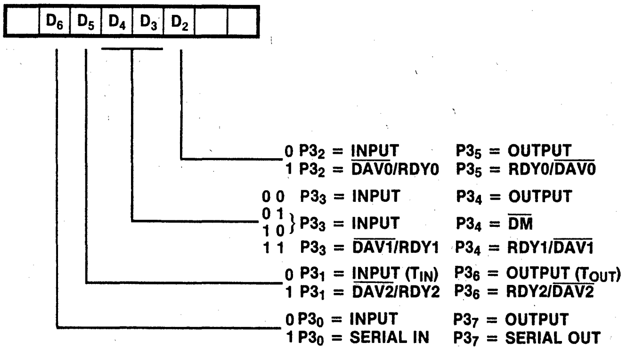

D6
D5
D4
D3
D2
0 P3₂ = INPUT P3₅ = OUTPUT
1 P3₂ = DAV0/RDY0 P3₅ = RDY0/DAV0
0 0 P3₃ = INPUT P3₄ = OUTPUT
0 1 / 1 0 } P3₃ = INPUT P3₄ = DM
1 1 P3₃ = DAV1/RDY1 P3₄ = RDY1/DAV1
0 P3₁ = INPUT (TIN) P3₆ = OUTPUT (TOUT)
1 P3₁ = DAV2/RDY2 P3₆ = RDY2/DAV2
0 P3₀ = INPUT P3₇ = OUTPUT
1 P3₀ = SERIAL IN P3₇ = SERIAL OUT

Figure 9-14. Port 3 I/O Operation

9.6 PORT HANDSHAKE

When Ports 0, 1, or 2 are configured for handshake operation, a pair of lines from Port 3 is used for handshake controls for each port. The handshake controls are interlocked to properly time asynchronous data transfers between the Z8 and its peripheral. One control line ($\overline{DAV}_n$) functions as a strobe from the sender to indicate to the receiver that data is available. The second control line (RDY_n) acknowledges receipt of the sender's data, and indicates when the receiver is ready to accept another data transfer.

In the input mode, data is latched into the port's input register by the first $\overline{DAV}$ signal, and is protected from being overwritten if additional pulses occur on the $\overline{DAV}$ line. This overwrite protection is maintained until the port data is read. In the output mode, data written to the port is not protected and can be overwritten by the Z8 during the handshake sequence. To avoid losing data, the software must not overwrite the port until the corresponding interrupt request indicates that the external device has latched the data.

The software can always read Port 3 output and input handshake lines, but cannot write to the output handshake lines.

Following is the recommended setup sequence when configuring a port for handshake operation for the first time after a reset:

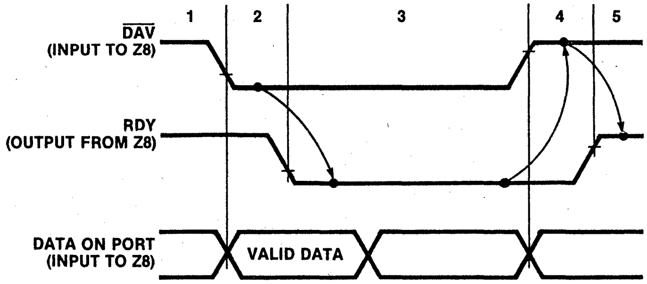
- Load P01M or P2M to configure the port for input/output.
- Load P3 to set the Output Handshake bit to a logic 1.
- Load P3M to select the Handshake mode for the port.

Once a data transfer begins, the configuration of the handshake lines should not be changed until handshake is completed.

Figures 9-15 and 9-16 show detailed operation for the handshake sequence.

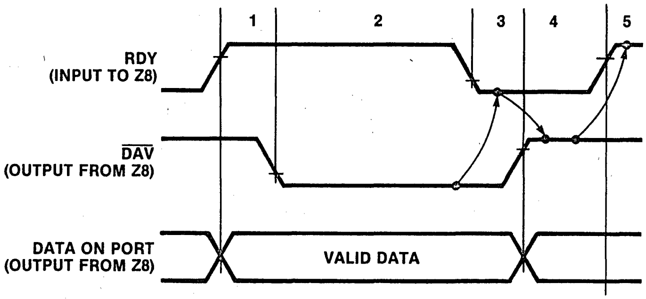
In applications requiring a strobed signal instead of the interlocked handshake, the Z8 can satisfy this requirement as follows:

- In the Strobed Input mode, data can be latched in the port input register using the $\overline{DAV}$ input. The data transfer rate must allow enough time for the software to read the port before strobing in the next character. The RDY output is ignored.
- In the Strobed Output mode, the RDY input should be tied to the $\overline{DAV}$ output.



State 1. Port 3 Ready output is High, indicating that the Z8 is ready to accept data.
State 2. The I/O device puts data on the port and then activates the $\overline{DAV}$ input. This causes the data to be latched into the port input register and generates an interrupt request.
State 3. The Z8 forces the Ready (RDY) output Low, signaling to the I/O device that the data has been latched.
State 4. The I/O device returns the $\overline{DAV}$ line High in response to RDY going Low.
State 5. The Z8 software must respond to the interrupt request and read the contents of the port in order for the handshake sequence to be completed. The RDY line goes High if and only if the port has not been read and $\overline{DAV}$ is High. This returns the interface to its initial state.

Figure 9-15. Z8 Input Handshake



State 1. RDY input is High indicating that the I/O device is ready to accept data.
State 2. The Z8 writes to the port register to initiate a data transfer. Writing the port outputs new data and forces $\overline{DAV}$ Low if and only if RDY is High.
State 3. The I/O device forces RDY Low after latching the data. RDY Low causes an interrupt request to be generated. The Z8 can write new data in response to RDY going Low; however, the data is not output until State 5.
State 4. The $\overline{DAV}$ output from the Z8 is driven High in response to RDY going Low.
State 5. After $\overline{DAV}$ goes High, the I/O device is free to raise RDY High thus returning the interface to its initial state.

Figure 9-16. Z8 Output Handshake

Figures 9-17 and 9-18 illustrate the strobed handshake connections.

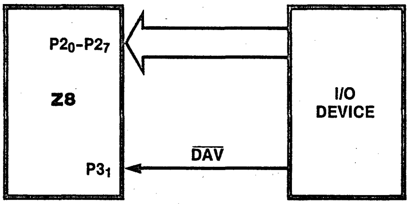


Figure 9-17. Input Strobed Handshake using Port 2

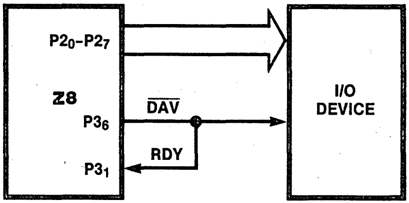


Figure 9-18. Output Strobed Handshake using Port 2

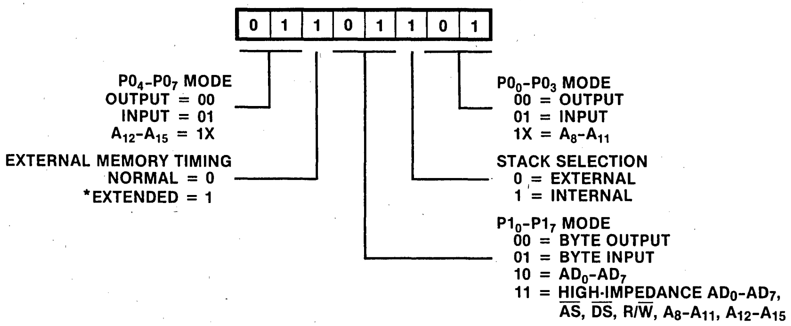
9.7 I/O PORT RESET CONDITIONS

After a hardware reset, mode registers P01M, P2M, and P3M are set as shown in Figures 9-19 - 9-22. Ports 0, 1 and 2 are configured for input operation on all bits, except Port 1 in the Z8681 and Ports 0 and 1 in the Z8682 as shown.

The pull-ups of Port 2 are set for open-drain. If active pull-ups are desired for Port 3 outputs, remember to configure them using P3M (Figure 9-22).

All special I/O functions of Port 3 are inactive, with $P3_0$-$P3_3$ set as inputs and $P3_4$-$P3_7$ set as outputs (Figure 9-23).

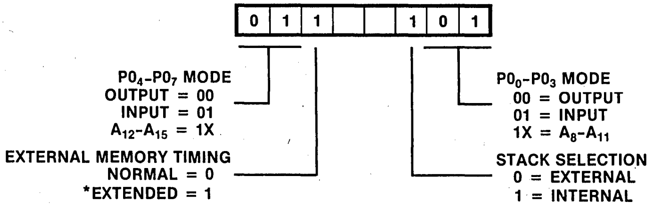
R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)



*ALWAYS EXTENDED TIMING AFTER RESET EXCEPT Z8682

Figure 9-19. Z8601/11 Port 0 and 1 Reset

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)



*ALWAYS EXTENDED TIMING AFTER RESET EXCEPT Z8682

Figure 9-20. Z8681 Ports 0 and 1 Reset

R248 P01M
Port 0-1 Mode Register
(% F8; Write Only)

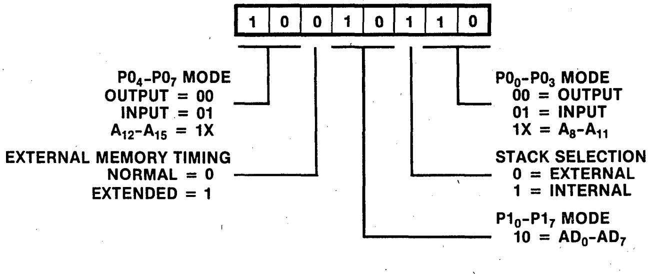


Figure 9-21. Z8682 Ports 0 and 1 Reset

R246 P2M
Port 2 Mode Register
(% F6; Write Only)

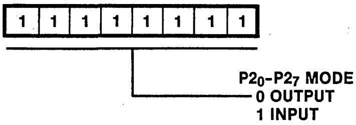


Figure 9-22. Port 2 Reset

R247 P3M
Port 3 Mode Register
(% F7; Write Only)

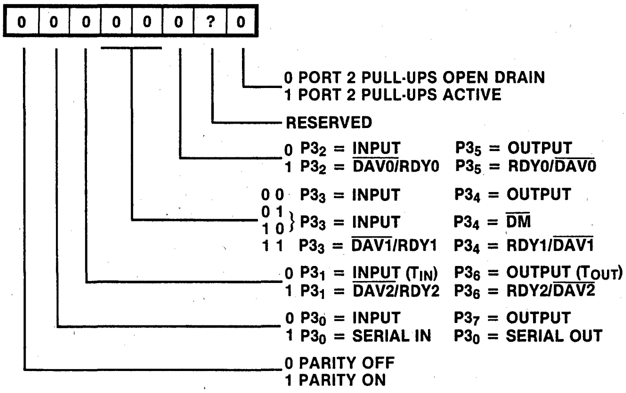

0 0 0 0 0 0 ? 0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
RESERVED
0 $P3_2$ = $\underline{\text{INPUT}}$ $P3_5$ = $\underline{\text{OUTPUT}}$
1 $P3_2$ = $\overline{\text{DAV0}}$/RDY0 $P3_5$ = RDY0/$\overline{\text{DAV0}}$
0 0 $P3_3$ = INPUT $P3_4$ = OUTPUT
0 1 / 1 0 } $P3_3$ = INPUT $P3_4$ = $\overline{\text{DM}}$
1 1 $P3_3$ = $\overline{\text{DAV1}}$/RDY1 $P3_4$ = RDY1/$\overline{\text{DAV1}}$
0 $P3_1$ = $\underline{\text{INPUT}}$ (T_{IN}) $P3_6$ = $\underline{\text{OUTPUT}}$ (T_{OUT})
1 $P3_1$ = $\overline{\text{DAV2}}$/RDY2 $P3_6$ = RDY2/$\overline{\text{DAV2}}$
0 $P3_0$ = INPUT $P3_7$ = OUTPUT
1 $P3_0$ = SERIAL IN $P3_0$ = SERIAL OUT
0 PARITY OFF
1 PARITY ON

Figure 9-23. Port 3 Reset

Chapter 10
Interrupts

10.1 INTRODUCTION

The Z8 microcomputer allows six different interrupt levels from eight sources: the four Port 3 lines $P3_0$-$P3_3$ make up the external interrupt sources while serial in, serial out, and the two counter/timers make up the internal sources. These interrupts can be masked and their priorities set by using the Interrupt Mask and the Interrupt Priority registers. All six interrupts can be globally disabled by resetting the master Interrupt Enable bit D_7 in the Interrupt Mask register with a Disable Interrupt (DI) instruction. Interrupts are globally enabled by setting D_7 with an Enable Interrupt (EI) instruction.

There are three interrupt control registers: the Interrupt Request register (IRQ), the Interrupt Mask register (IMR), and the Interrupt Priority register (IPR). Figure 10-1 shows addresses and identifiers for the interrupt control registers. Figure 10-2 is a block diagram showing the Interrupt Mask and Interrupt Priority logic.

The Z8 family supports both vectored and polled interrupt handling. Details on vectored and polled interrupts can be found in Sections 10.6 and 10.7.

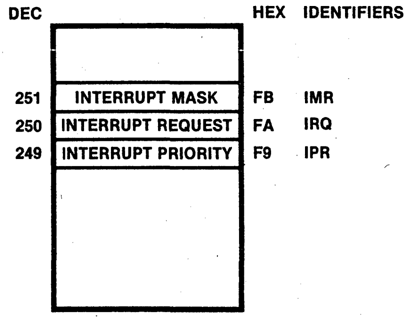


Figure 10-1. Interrupt Control Registers

10.2 INTERRUPT SOURCES

Table 10-1 presents the interrupt types, sources, and vectors available in the Z8 family of processors.

10.2.1 External Interrupt Sources

External sources involve interrupts request lines IRQ_0-IRQ_3. IRQ_0, IRQ_1, and IRQ_2 are always generated by a negative edge signal on the corresponding Port 3 pin ($P3_2$, $P3_3$, $P3_1$ correspond to IRQ_0, IRQ_1, and IRQ_2, respectively). Figure 10-3 is a block diagram for interrupt sources IRQ_0, IRQ_1, and IRQ_2.

When the Port 3 pin ($P3_1$, $P3_2$, or $P3_3$) goes Low, the first flip-flop is set. The next two flip-flops synchronize the request to the internal clock and delay it by four external clock periods. The output of the last flip-flop (IRQ_0, IRQ_1, or IRQ_3) goes to the corresponding Interrupt Request register.

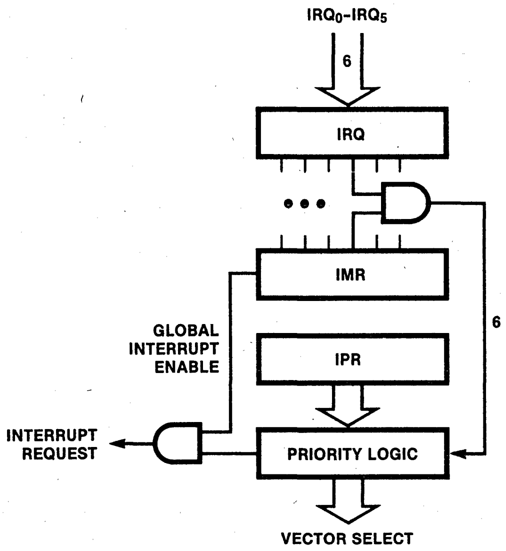


Figure 10-2. Interrupt Block Diagram

Table 10-1.
Interrupt Types, Sources, and Vectors

Name	Source	Vector Location	Comments
IRQ_0	$\overline{DAV_0}$, IRQ_0	0,1	External ($P3_2$), ↓ Edge Triggered
IRQ_1	$\overline{DAV_1}$, IRQ_1	2,3	External ($P3_3$), ↓ Edge Triggered
IRQ_2	$\overline{DAV_2}$, IRQ_2, T_{IN}	4,5	External ($P3_1$), ↓ Edge Triggered
IRQ_3	IRQ_3	6,7	External ($P3_0$), ↓ Edge Triggered
	Serial In	6,7	Internal
IRQ_4	T_0	8,9	Internal
	Serial Out	8,9	Internal
IRQ_5	T_1	10,11	Internal

IRQ_3 can be generated from an external source only if Serial In is not enabled; otherwise, its source is internal. The external request is generated by a negative edge signal on $P3_0$ as shown in Figure 10-4. Again, the external request is synchronized and delayed before reaching IRQ.

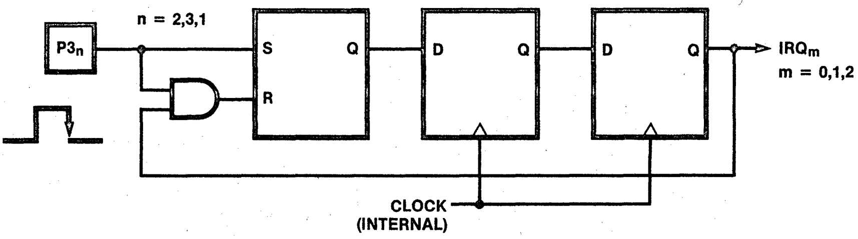


Figure 10-3. Interrupt Sources IRQ_0-IRQ_2 Block Diagram

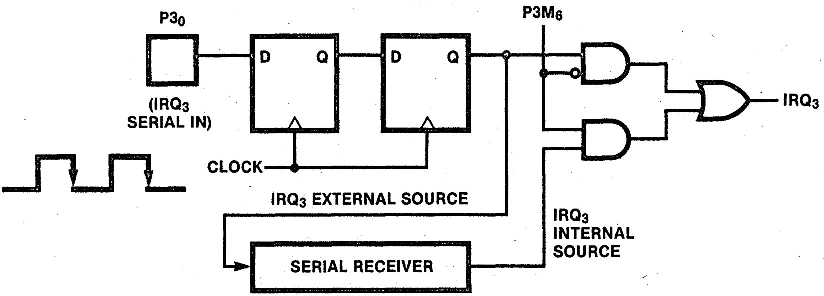


Figure 10-4. Interrupt Source IRQ_3 Block Diagram

10.2.2 Internal Interrupt Sources

Internal sources involve interrupt requests IRQ_3-IRQ_5. If Serial In is enabled, IRQ_3 generates an interrupt request whenever the receiver assembles a complete byte. Interrupt level IRQ_4 has two mutually exclusive sources, Counter/Timer 0 (T_0) and the Serial Out transmitter. If Serial Out is enabled, an interrupt request is generated when the transmit buffer is empty. If T_0 is enabled, an interrupt request is generated at T_0 end-of-count. IRQ_5 generates an interrupt request at Counter/Timer 1's (T_1) end-of-count.

For more details on the internal interrupt sources, refer to the chapters describing serial I/O and the counter/timers.

10.3 INTERRUPT REQUEST (IRQ) REGISTER LOGIC AND TIMING

Figure 10-5 shows the logic diagram for the Interrupt Request register. The leading edge of the request will set the first flip-flop, which will remain set until interrupt requests are sampled. Requests are sampled internally during the last clock cycle before an opcode fetch (Figure 10-6). External requests are sampled two internal clocks earlier, due to the synchronizing flip-flops shown in Figures 10-3 and 10-4.

At sample time the request is transferred to the second flip-flop in Figure 10-5, which drives the interrupt mask and priority logic. When an interrupt cycle occurs, this flip-flop will be reset only for the highest priority level that is enabled.

The user has direct access to the second flip-flop by reading and writing the IRQ register. IRQ is read by specifying it as the source register of an instruction and written by specifying it as the destination register.

10.4 INTERRUPT INITIALIZATION

After reset, all interrupts are disabled and must be initialized before vectored or polled interrupt processing can begin. The Interrupt Priority register (IPR), Interrupt Mask register (IMR) and Interrupt Request register (IRQ) must be initialized, in that order, to start the interrupt process. However, IPR need not be initialized for polled processing.

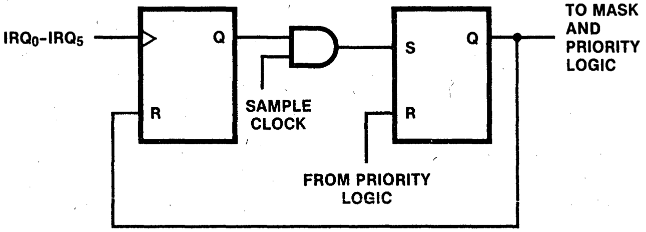


Figure 10-5. IRQ Register Logic

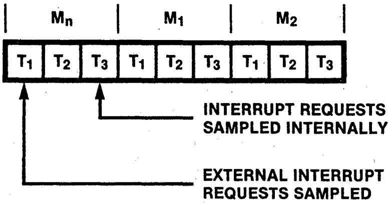

Mn
M1
M2
T1
T2
T3
T1
T2
T3
T1
T2
T3
INTERRUPT REQUESTS SAMPLED INTERNALLY
EXTERNAL INTERRUPT REQUESTS SAMPLED

Figure 10-6. Interrupt Request Timing

10.4.1 Interrupt Priority Register (IPR) Initialization

IPR (Figure 10-7) is a write-only register that sets priorities for the six levels of vectored interrupts in order to resolve simultaneous interrupt requests. (There are 48 sequence possibilities for interrupts.) The six interrupt levels IRQ_0-IRQ_5 are divided into three groups of two interrupt requests each. One group contains IRQ_3 ($SI/P3_0$) and IRQ_5 (T_1), another group contains IRQ_0 ($P3_2$) and IRQ_2 ($P3_1$), and the third group contains IRQ_1 ($P3_3$) and IRQ_4 (SO/T_0).

Priorities can be set both within and between groups as shown in Table 10-2. Bits D_1, D_2, and D_5 define the priority of the individual members within the three groups. Bits D_0, D_3, and D_4 are encoded to define six priority orders between the three groups. Bits D_6 and D_7 are not used.

R249 IPR
Interrupt Priority Register
(% F9; Write Only)

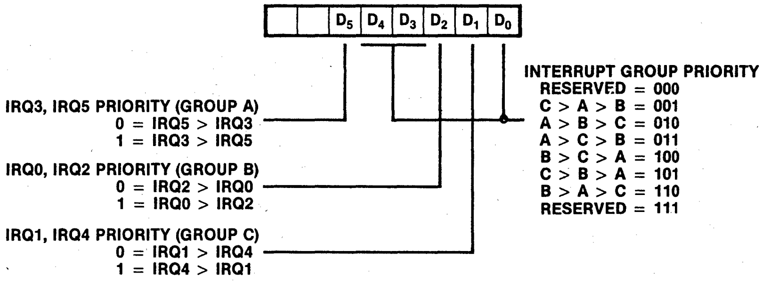


Figure 10-7. Interrupt Priority Register

Table 10-2. Interrupt Priority

Group	Bit	Priority Highest	 Lowest
C	D_1=0	IRQ_1	IRQ_4
	1	IRQ_4	IRQ_1
B	D_2=0	IRQ_2	IRQ_0
	1	IRQ_0	IRQ_2
A	D_5=0	IRQ_5	IRQ_3
	1	IRQ_3	IRQ_5

Bit Pattern D_4	 D_2	 D_0	Group Priority Highest --> Lowest
0	0	0	NOT USED
0	0	1	C A B
0	1	0	A B C
0	1	1	A C B
1	0	0	B C A
1	0	1	C B A
1	1	0	B A C
1	1	1	NOT USED

10.4.2 Interrupt Mask Register (IMR) Initialization

IMR (Figure 10-8) individually or globally enables or disables the six interrupt requests. When bits D_0-D_5 are set to 1, the corresponding interrupt requests are enabled. D_7 is the master enable and must be set before any of the individual interrupt requests can be recognized. Resetting D_7 globally disables all of the interrupt requests. D_7 is set and reset by the EI and DI instructions. It is automatically reset during an interrupt machine cycle and set following the execution of an Interrupt Return (IRET) instruction.

NOTE

> D_7 must be reset by the DI instruction before the contents of the Interrupt Mask register or the Interrupt Priority register are changed except:
>
> - Immediately after a hardware reset, or
>
> - Immediately after executing an interrupt cycle and before IMR_7 has been set by any instruction.

10.4.3 Interrupt Request (IRQ) Register Initialization

IRQ (Figure 10-9) is a read/write register that stores the interrupt requests for both vectored and polled interrupts. When an interrupt is made on any of the six levels, the corresponding bit position in the register is set to 1. Bits D_0-D_5 are assigned to interrupt requests IRQ_0-IRQ_5, respectively.

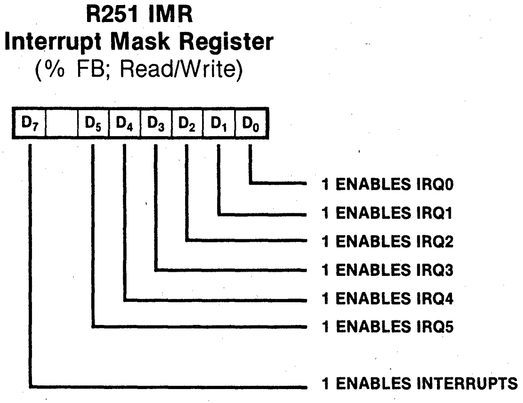


Figure 10-8. Interrupt Mask Register

IRQ is held in a Reset state until an EI instruction is executed. For polled processing, IRQ must still be initialized by an EI instruction, but IMR should first be cleared to 0 to individually inhibit all interrupt requests while interrupts are globally enabled:

```
CLR    IMR
EI
DI
```

10.5 IRQ SOFTWARE INTERRUPT GENERATION

IRQ can be used to generate software interrupts by specifying IRQ as the destination of any instruction referencing the register file. These Software Interrupts (SWI) are controlled in the same manner as hardware-generated requests, i.e., the IPR and the IMR control the priority and enabling of each SWI level.

To generate an SWI, the desired request bit in the IRQ is set as follows:

```
OR IRQ,#IRQLVL
```

where the immediate data, IRQLVL, has a 1 in the bit position corresponding to the level of the SWI desired. For example, if an SWI on level 5 is desired, IRQLVL would have a 1 in the bit 5 position:

```
OR IRQ,#%200100000
```

where the immediate data is preceded by %2 to indicate a binary constant. With this instruction, if the interrupt system is globally enabled, level 5 is enabled, and there are no higher priority pending requests, control is transferred to the service routine pointed to by the level 5 vector.

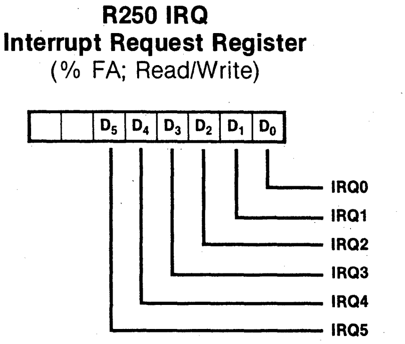


Figure 10-9. Interrupt Request Register

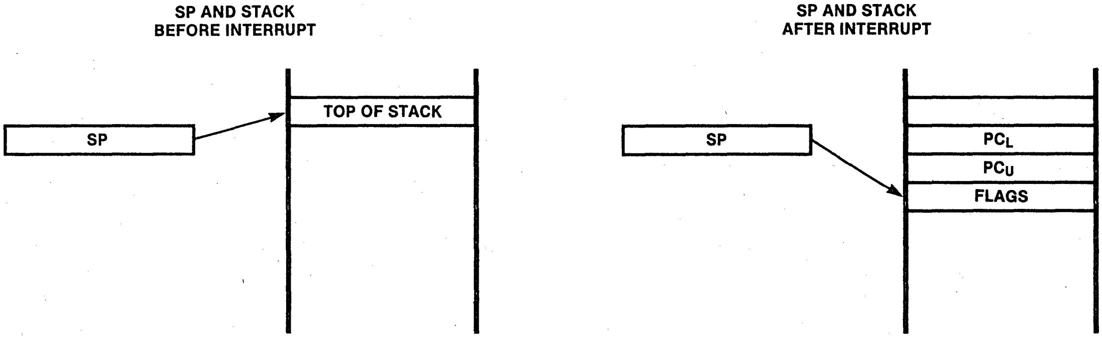


Figure 10-10. Effect of Interrupt on Stack

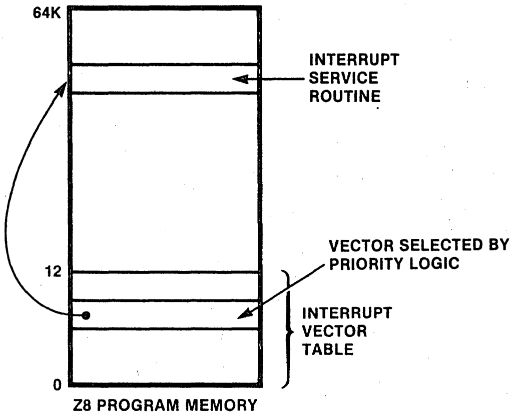


Figure 10-11. Interrupt Vectoring

10.6 VECTORED PROCESSING

Each Z8 interrupt level has its own vector. When an interrupt occurs, control passes to the service routine pointed to by the interrupt's location in program memory. The sequence of events for vectored interrupts is as follows:

- PUSH PC lower byte on stack
- PUSH PC upper byte on stack
- PUSH FLAGS on stack
- Fetch upper byte of vector
- Fetch lower byte of vector
- Branch to service routine specified by vector

Figures 10-10 and 10-11 show the vectored interrupt operation.

10.6.1 Vectored Interrupt Cycle Timing

Interrupt cycle timing for all Z8 devices except the Z8681 is diagrammed in Figure 10-12. Timing for the Z8681 ROMless device is different and is shown in Figure 10-13.

10.6.2 Nesting of Vectored Interrupts

Nesting of vectored interrupts allows higher priority requests to interrupt a lower priority request. To initiate vectored interrupt nesting, do the following during the interrupt service routine:

- Push the old IMR on the stack.
- Load IMR with a new mask to disable lower priority interrupts.
- Execute EI instruction.
- Proceed with interrupt processing.
- After processing is complete, execute DI instruction.
- Restore the IMR to its original value by returning the previous mask from the stack.
- Execute IRET.

Depending on the application, some simplification of the above procedure may be possible.

10.7 POLLED PROCESSING

Polled interrupt processing is supported by masking off the IRQ levels to be polled. This is accomplished by clearing the corresponding bit in the IMR to 0.

To initiate polled processing, check the bits of interest in the IRQ using the Test Under Mask (TM) instruction. If the bit is set, call or branch to the service routine. The service routine services the request, resets its Request bit in the IRQ, and branches or returns back to the main program. An example of a polling routine is as follows:

```
     TM IRQ,#MASK    !Test for request         !
     JR Z   NEXT     !If no request go to NEXT !
     CALL SERVICE    !If request is there      !
                     !then service it          !
   NEXT:
         .
         .
         .
SERVICE:             !Process Request          !
         .
         .
         .
     AND IRQ,#MASK_  !Clear Request bit        !
     RET             !Return to next           !
```

In this example, if IRQ_2 is being polled, MASK will be %200000100 (in binary) and MASK_ will be %211111011.

10.8 RESET CONDITIONS

During a reset, all bits in IPR are undefined.

In IMR, bit D_7 is 0 and bits D_0-D_5 are undefined. Bit D_6 is not implemented, though reading this bit returns 0.

IRQ bits D_0-D_5 are held at 0 until an EI instruction is executed. Bits D_6 and D_7 are not implemented, but reading these bits returns 0.

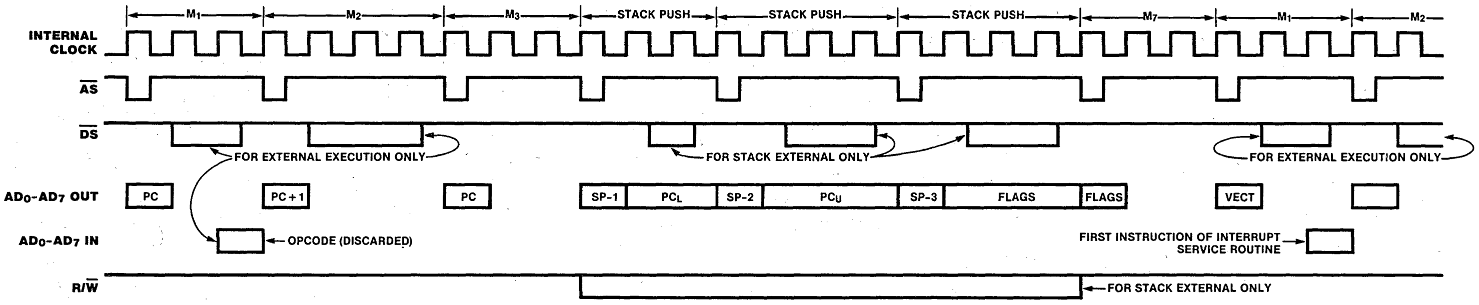


Figure 10-12. ROM Z8 Interrupt Timing (shrink parts)

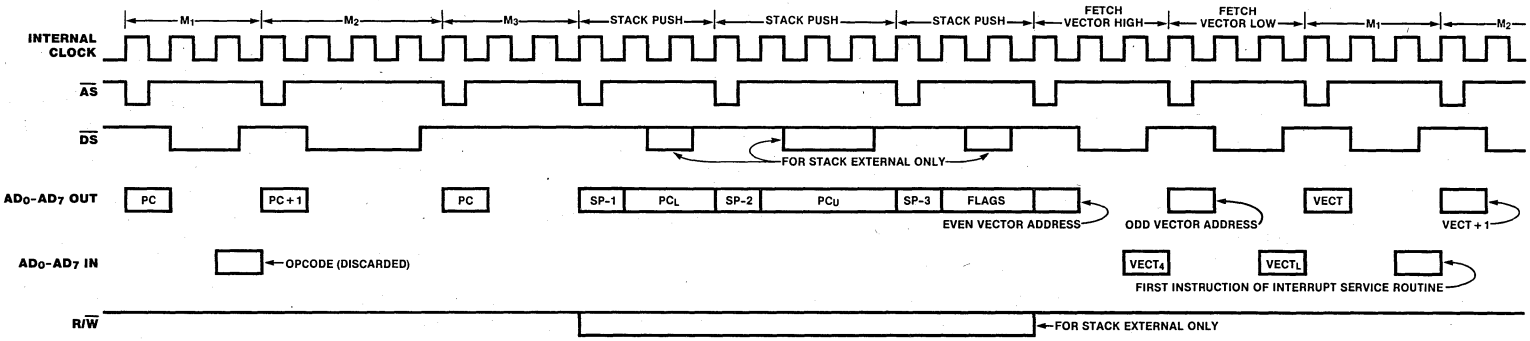


Figure 10-13. Z8681 ROMless Z8 Interrupt Timing

Chapter 11
Counter/Timers

11.1 INTRODUCTION

The Z8 provides two 8-bit counter/timers, T_0 and T_1, each driven by its own 6-bit prescaler, PRE_0 and PRE_1. Both counter/timers are independent of the processor instruction sequence, which relieves software from time-critical operations such as interval timing or event counting.

Each counter/timer operates in either Single-Pass or Continuous mode. At the end-of-count, counting either stops or the initial value is reloaded and counting continues. Under software control, new values are loaded immediately or when the end-of-count is reached. Software also controls counting mode, how a counter/timer is started or stopped, and its use of I/O lines. Both the counter and prescaler registers can be altered while the counter/timer is running.

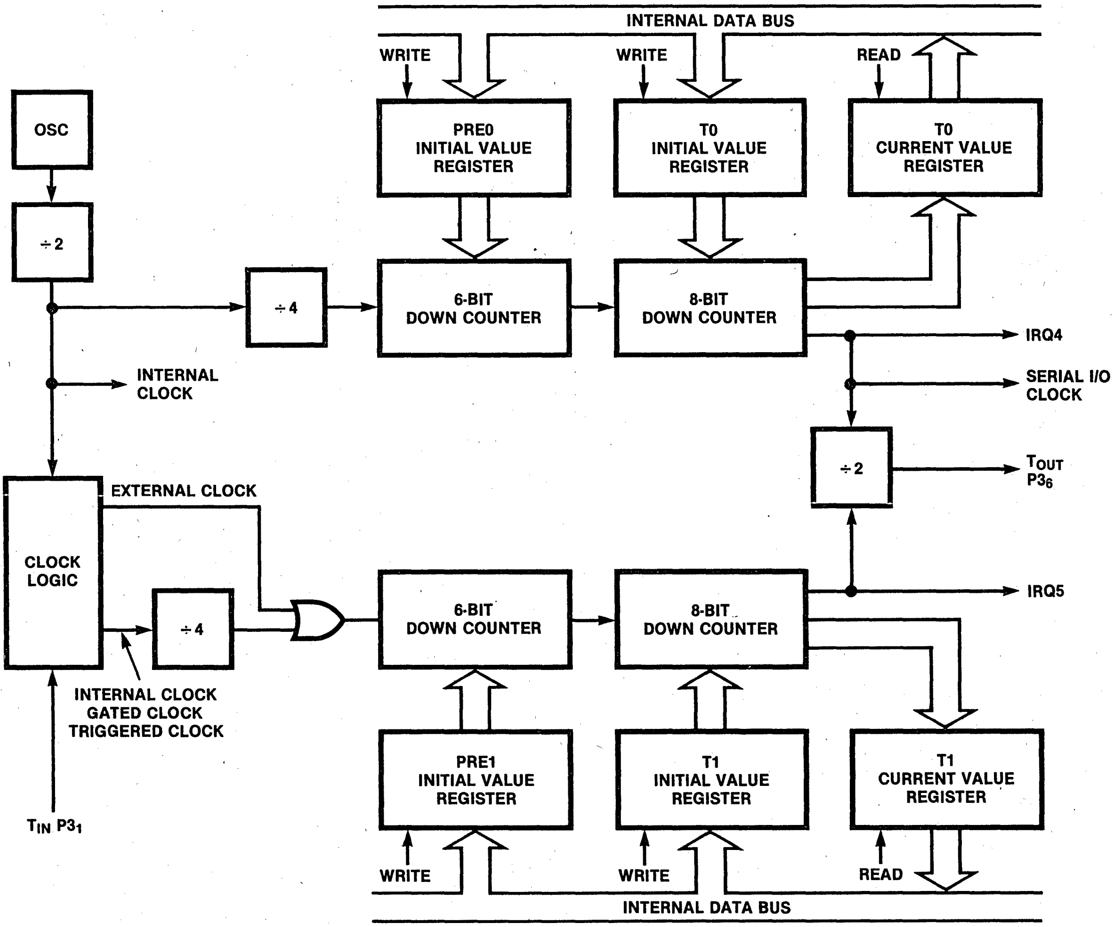


Figure 11-1. Counter/Timer Block Diagram

Counter/timers 0 and 1 are driven by a timer clock generated by dividing the internal clock by four. The divide-by-four stage, the 6-bit prescaler, and the 8-bit counter/timer form a synchronous 16-bit divide chain. Counter/timer 1 can also be driven by an external input (T_{IN}) via Port 3 line $P3_1$. Port 3 line $P3_6$ can serve as a timer output (T_{OUT}) through which T_0, T_1, or the internal clock can be output. The timer output will toggle at the end-of-count. Figure 11-1 is a block diagram of the counter/timers.

The counter/timer, prescaler, and associated mode registers are mapped into the register file as shown in Figure 11-2. This allows the software to treat the counter/timers as general-purpose registers, and eliminates the need for special instructions.

11.2 PRESCALERS AND COUNTER/TIMERS

The prescalers, PRE_0 (%F5) and PRE_1 (%F3), each consist of an 8-bit register and a 6-bit down-counter as shown in Figure 11-1. The prescaler registers are write-only registers. Reading the prescalers returns the value %FF. Figures 11-3 and 11-4 show the prescaler registers.

The six most significant bits (D_2-D_7) of PRE_0 or PRE_1 hold the prescalers count modulo, a value from 1 to 64 decimal. The prescaler registers also contain control bits that specify T_0 and T_1 counting modes. These bits also indicate whether the clock source for T_1 is internal or external. These control bits will be discussed in detail throughout this chapter.

The counter/timers, T_0 (%F4) and T1 (%F_2), each consist of an 8-bit down-counter, a write-only register which holds the initial count value, and a read-only register which holds the current count value (Figure 11-1). The initial value can range from 1 to 256 decimal (%01,%02,...,%00). Figure 11-5 illustrates the counter/timer registers.

DEC		HEX	IDENTIFIERS
247	PORT 3 MODE	F7	P3M
245	T0 PRESCALER	F5	PRE0
244	TIMER/COUNTER 0	F4	T0
243	T1 PRESCALER	F3	PRE1
242	TIMER/COUNTER 1	F2	T1
241	TIMER MODE	F1	TMR

R245 PRE0
Prescaler 0 Register
(% F5; Write Only)

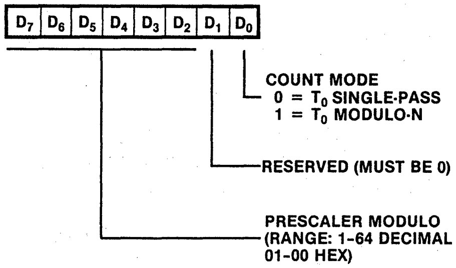


Figure 11-3. Prescaler 0 Register

R243 PRE1
Prescaler 1 Register
(% F3; Write Only)

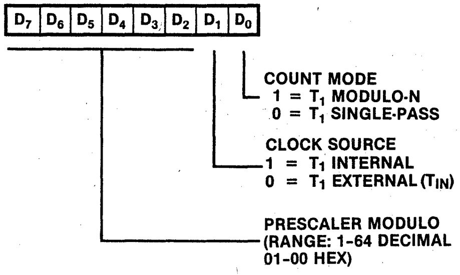
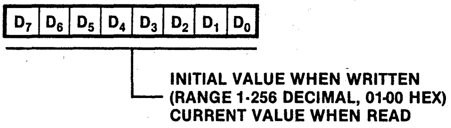


Figure 11-4. Prescaler 1 Register

R242 T1
Counter/Timer 1 Register
(% F2; Read/Write)
R244 T0
Counter/Timer 0 Register
(% F4; Read/Write)



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
INITIAL VALUE WHEN WRITTEN
(RANGE 1-256 DECIMAL, 01-00 HEX)
CURRENT VALUE WHEN READ

11.3 COUNTER/TIMER OPERATION

Under software control, counter/timers are started and stopped via the Timer Mode register (%F1) bits D_0-D_3 (Figure 11-6). Each counter/timer is associated with a Load bit and an Enable Count bit.

11.3.1 Load and Enable Count Bits

Setting the Load bit (D_0 to 1 for T_0 and D_2 to 1 for T_1) transfers the initial value in the prescaler and the counter/timer registers into their respective down-counters. The next internal clock resets bits D_0 and D_2 to 0, readying the Load bit for the next load operation. The initial values may be loaded into the down-counters at any time. If the counter/timer is running, it continues to do so and starts the count over with the initial value. Therefore, the Load bit actually functions as a software re-trigger.

The counter/timers remain at rest as long as the Enable Count bits D_1 and D_3 are both 0. To enable counting, the Enable Count bit (D_1 for T_0 and D_3 for T_1) must be set to 1. Counting actually starts when the Enable Count bit is written by an instruction. The first decrement occurs four internal clock periods after the Enable Count bit has been set.

The Load and Enable Count bits can be set at the same time. For example, using the instruction OR TMR #%03 sets both D_0 and D_1 of TMR to 1. This loads the initial values of PRE_0 and T_0 into their respective counters and starts the count after the M2T2 machine state after the operand is fetched (Figure 11-7).

11.3.2 Prescaler Operations

During counting, the programmed clock source drives the prescaler 6-bit counter. The counter is counted down from the value specified by bits D_2-D_7 of the corresponding prescaler register, PRE_0 or PRE_1 (Figure 11-8). When the prescaler counter reaches its end-of-count, the initial value is reloaded and counting continues. The prescaler never actually reaches 0. For example, if the prescaler is set to divide by 3, the count sequence is:

3-2-1-3-2-1-3-2....

Each time the prescaler reaches its end-of-count a carry is generated, which allows the counter/timer to decrement by one on the next timer clock input. When the counter/timer and the prescaler both reach their end-of-count, an interrupt request is generated -- IRQ_4 for T_0 and IRQ_5 for T_1. Depending on the counting mode selected, the counter/timer will either come to rest with its value at %00 (Single-Pass mode) or the initial value will be automatically reloaded and counting will continue (Continuous mode).

R241 TMR
Timer Mode Register
(% F1; Read/Write)

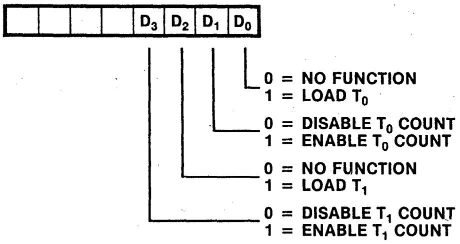


Figure 11-6. Timer Mode Register

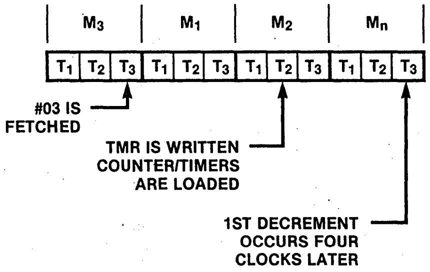

M_3
M_1
M_2
M_n
T_1 T_2 T_3 T_1 T_2 T_3 T_1 T_2 T_3 T_1 T_2 T_3
#03 IS FETCHED
TMR IS WRITTEN COUNTER/TIMERS ARE LOADED
1ST DECREMENT OCCURS FOUR CLOCKS LATER

Figure 11-7. Starting The Count

R243 PRE1
Prescaler 1 Register
(% F3; Write Only)
R245 PRE0
Prescaler 0 Register
(% F5; Write Only)

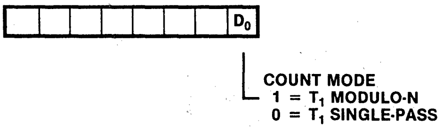


Figure 11-8. Counting Modes

The counting modes are controlled by bit D_0 of PRE_0 and PRE_1, with D_0 cleared to 0 for Single-pass counting mode or set to 1 for Continuous mode.

The counter/timers can be stopped at any time by setting the Enable Count bit to 0, and restarted by setting it back to 1. The counter/timer will continue its count value at the time it was stopped. The current value in the counter/timer (T_0 or T_1) can be read at any time without affecting the counting operation.

New initial values can be written to the prescaler or the counter/timer registers at any time. These values will be transferred to their respective down-counters on the next load operation. If the counter/timer mode is Continuous, the next load occurs on the timer clock following an end-of-count. New initial values should be written before the desired load operation, since the prescalers always effectively operate in Continuous count mode.

The time interval (i) until end-of-count, is given by the equation

$$i = t \times p \times v$$

in which t is 8 divided by XTAL frequency, p is the prescaler value (1 - 64), and v is the counter/timer value (1 - 256). It should be apparent that the prescaler and counter/timer are true divide-by-n counters.

11.4 T_{OUT} MODES

The Timer Mode register TMR (%F1) (Figure 11-10) is used in conjunction with the Port 3 Mode register P3M (%F7) (Figure 11-9) to configure $P3_6$ for T_{OUT} operation. In order for T_{OUT} to function, $P3_6$ must be defined as an output line by setting P3M bit D_5 to 0. Output is controlled by one of the counter/timers (T_0 or T_1) or the internal clock.

The counter/timer to be output is selected by TMR bits D_7 and D_6. T_0 is selected to drive the T_{OUT} line by setting D_7 to 0 and D_6 to 1. Likewise, T1 is selected by setting D_7 and D_6 to 1 and 0 respectively. The counter/timer T_{OUT} mode is turned off by setting TMR bits D_7 and D_6 both to 0, freeing $P3_6$ to be a data output line.

T_{OUT} is initialized to a logic 1 whenever the TMR Load bit (D_0 for T_0 or D_2 for T_1) is set to 1.

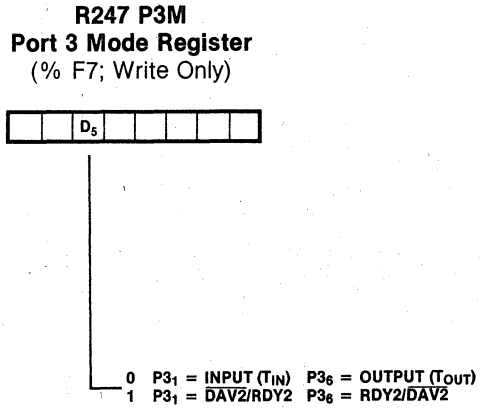

R247 P3M
Port 3 Mode Register
(% F7; Write Only)
D5
0 P31 = INPUT (TIN) P36 = OUTPUT (TOUT)
1 P31 = DAV2/RDY2 P36 = RDY2/DAV2

Figure 11-9.
Port 3 Mode Register T_{OUT} Operation

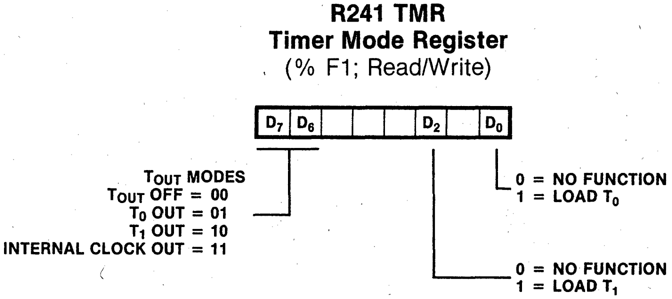


Figure 11-10. Timer Mode Register T_{OUT} Operation

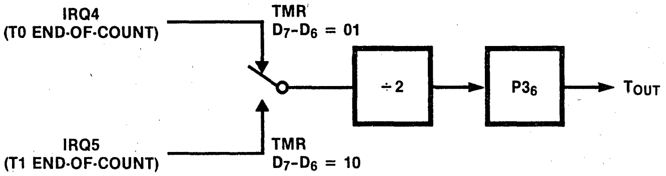


Figure 11-11. Counter/Timers Output Via T_{OUT}

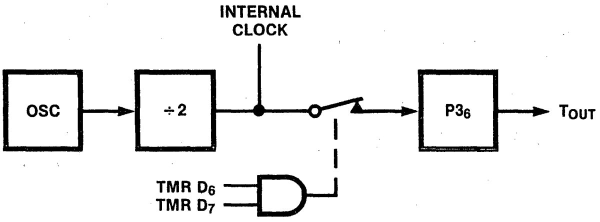


Figure 11-12. Internal Clock Output Via T_{OUT}

At end-of-count, the interrupt request line (IRQ_4 or IRQ_5), clocks a toggle flip-flop. The output of this flip-flop drives the T_{OUT} line, $P3_6$. In all cases, when the selected counter/timer reaches its end-of-count, T_{OUT} toggles to its opposite state (Figure 11-11). If, for example, the counter/timer is in Continuous counting mode, T_{OUT} will have a 50% duty cycle output. This duty cycle can easily be controlled by varying the initial values after each end-of-count.

The internal clock can be selected as output instead of T_0 or T_1 by setting TMR bits D_7 and D_6 both to 1. The internal clock (XTAL frequency/2) is then directly output on $P3_6$ (Figure 11-12).

While programmed as T_{OUT}, $P3_6$ cannot be modified by a write to port register P3. However, the Z8 software can examine $P3_6$'s current output by reading the port register.

11.5 T_{IN} MODES

The Timer Mode register TMR (%F1) (Figure 11-13) is used in conjunction with the Prescaler register PRE_1 (%F3) (Figure 11-14) to configure $P3_1$ as T_{IN}. T_{IN} is used in conjunction with T_1 in one of four modes:

- External clock input
- Gated internal clock
- Triggered internal clock
- Retriggerable internal clock

R241 TMR
Timer Mode Register
(% F1; Read/Write)

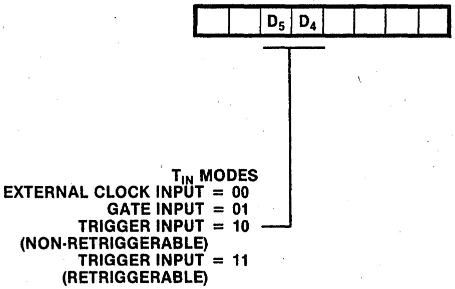


Figure 11-13. Timer Mode Register T_{IN} Operation

R243 PRE1
Prescaler 1 Register
(% F3; Write Only)

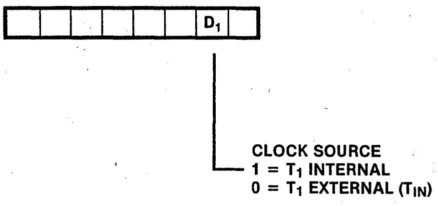


Figure 11-14. Prescaler 1 T_{IN} Operation

The counter/timer clock source must be configured for external by setting PRE_1 bit D_2 to 0. The Timer Mode register bits D_5 and D_4 can then be used to select the desired T_{IN} operation.

For T_1 to start counting as a result of a T_{IN} input, the Enable Count bit D_3 in TMR must be set to 1. When using T_{IN} as an external clock or a gate input, the initial values must be loaded into the down-counters by setting the Load bit D_2 in TMR to a 1 before counting begins. In the descriptions of T_{IN} that follow, it is assumed that the programmer has performed these operations. Initial values are automatically loaded in Trigger and Retrigger modes so software loading is unnecessary.

It is suggested that $P3_1$ be configured as an input line by setting P3M bit D_5 to 0 although T_{IN} is still functional if $P3_1$ is configured as a handshake input.

Each High-to-Low transition on T_{IN} generates interrupt request IRQ_2, regardless of the selected T_{IN} mode or the enabled/disabled state of T_1. IRQ_2 must therefore be masked or enabled according to the needs of the application.

11.5.1 External Clock Input Mode

The T_{IN} External Clock Input mode (TMR bits D_5 and D_4 both set to 0) supports counting of external events, where an event is considered to be a High-to-Low transition on T_{IN} (Figure 11-15). occurrence (Single-Pass mode) or on every nth occurrence (Continuous mode) of that event.

11.5.2 Gated Internal Clock Mode

The T_{IN} Gated Internal Clock mode (TMR bits D_5 and D_4 set to 0 and 1 respectively) measures the duration of an external event. In this mode, the T_1 prescaler is driven by the internal timer clock, gated by a High level on T_{IN} (Figure 11-16). T_1 counts while T_{IN} is High and stops counting while T_{IN} is Low. Interrupt request IRQ_2 is generated on the High-to-Low transition of T_{IN}, signaling the end of the gate input. Interrupt request IRQ_5 is generated if T_1 reaches its end-of-count.

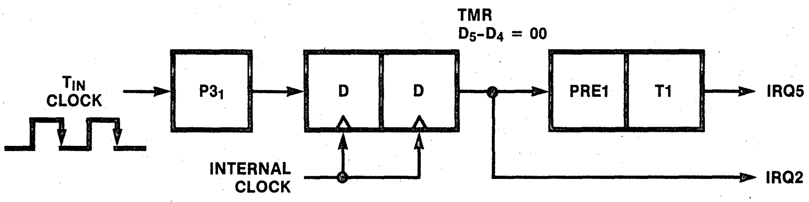


Figure 11-15. External Clock Input Mode

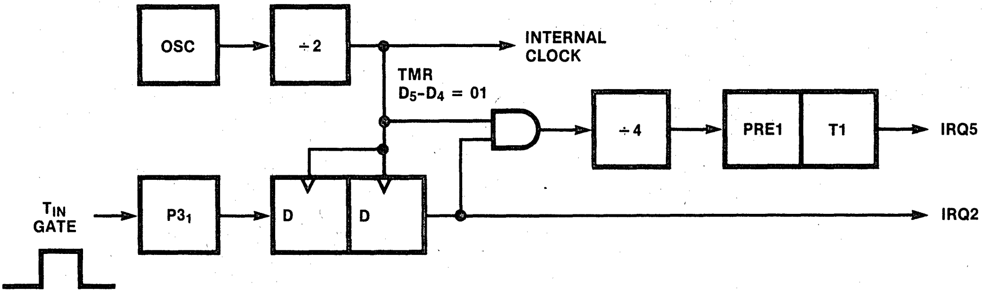


Figure 11-16. Gated Clock Input Mode

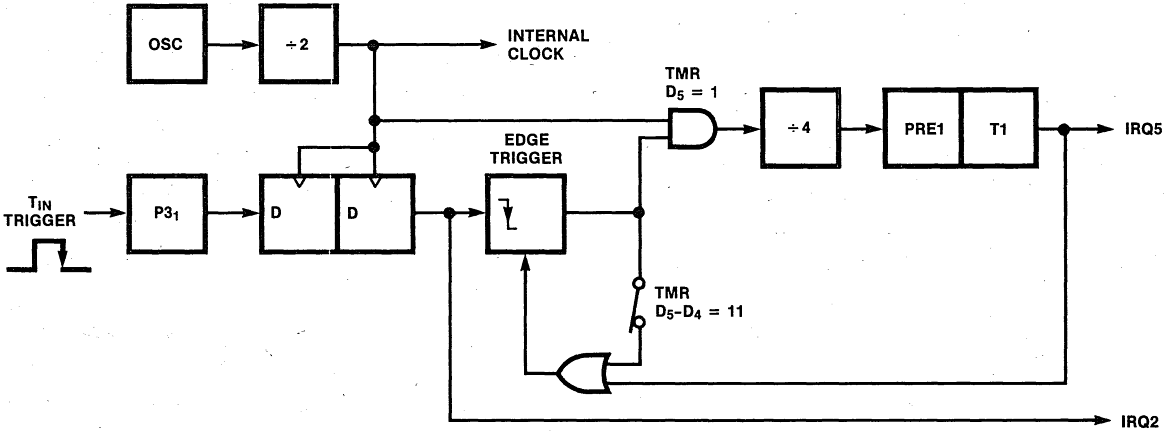


Figure 11-17. Triggered Clock Mode

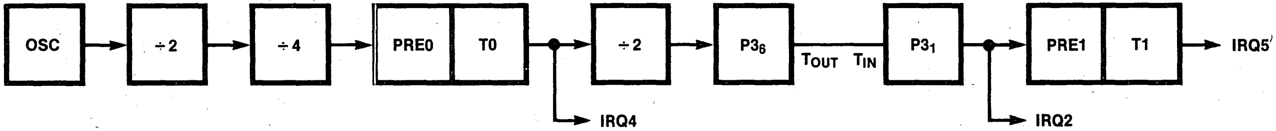


Figure 11-18. Cascaded Counter/Timers

11.5.3 Triggered Input Mode

The T_{IN} Triggered Input mode (TMR bits D_5 and D_4 set to 1 and 0 respectively) causes T_1 to start counting as the result of an external event (Figure 11-17). T_1 is then loaded and clocked by the internal timer clock following the first High-to-Low transition on the T_{IN} input. Subsequent T_{IN} transitions do not affect T_1. In the Single-Pass mode, the Enable bit is reset whenever T_1 reaches its end-of-count. Further T_{IN} transitions will have no effect on T_1 until software sets the Enable Count bit again. In Continuous mode, once T_1 is triggered counting continues until software resets the Enable Count bit. Interrupt request IRQ_5 is generated when T_1 reaches its end-of-count.

11.5.4 Retriggerable Input Mode

The T_{IN} Retriggerable Input mode (TMR bits D_5 and D_4 both set to 1) causes T_1 to load and start counting on every occurrence of a High-to-Low transition on T_{IN} (Figure 11-17). Interrupt request IRQ_5 will be generated if the programmed time interval (determined by T_1 prescaler and counter/timer register initial values) has elapsed since the last High-to-Low transition on T_{IN}. In Single-Pass mode, the end-of-count resets the Enable Count bit. Subsequent T_{IN} transitions will not cause T_1 to load and start counting until software sets the Enable Count bit again. In Continuous mode, counting continues once T_1 is triggered until software resets the Enable Count bit. When enabled, each High-to-Low T_{IN} transition causes T_1 to reload and restart counting. Interrupt request IRQ_5 is generated on every end-of-count.

11.6 CASCADING COUNTER/TIMERS

For some applications, it may be necessary to measure a time interval greater than a single counter/timer can measure. In this case, T_{IN} and T_{OUT} can be used to cascade T_0 and T_1 as a single unit (Figure 11-18). T_0 should be configured to operate in Continuous mode and to drive T_{OUT}. T_{IN} should be configured as an external clock input to T_1 and wired back to T_{OUT}. On every other T_0 end-of-count, T_{OUT} undergoes a High-to-Low transition which causes T_1 to count. T_1 can operate in either Single-Pass or Continuous mode. Each time T_1's end-of-count is reached, interrupt request IRQ_5 is generated. Interrupt requests IRQ_2 (T_{IN} High-to-Low transitions) and IRQ_4 (T_0 end-of-count) are also generated but are most likely of no importance in this configuration and should be disabled.

11.7 RESET CONDITIONS

After a hardware reset, the counter/timers are disabled and the contents of both the counter/timer registers and the prescaler modulos are undefined. However, the counting modes are configured for Single-Pass and T_1's clock source is set for external. T_{IN} is set for External Clock mode, and the T_{OUT} mode is off. Figures 11-19 through 11-22 show the binary reset values of the Prescaler, Counter/Timer, and Timer Mode registers.

R242 T1
Counter/Timer 1 Register
(% F2; Read/Write)
R244 T0
Counter/Timer 0 Register
(% F4; Read/Write)

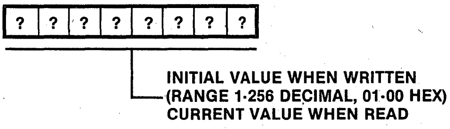


Figure 11-19. Counter/Timer Reset

R243 PRE1
Prescaler 1 Register
(% F3; Write Only)

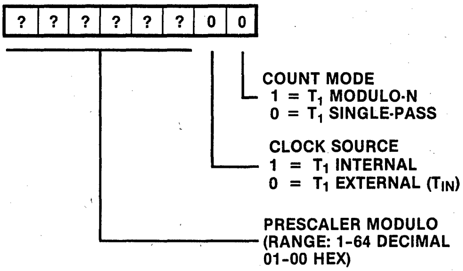


Figure 11-20. Prescaler 1 Register Reset

R245 PRE0
Prescaler 0 Register
(% F5; Write Only)

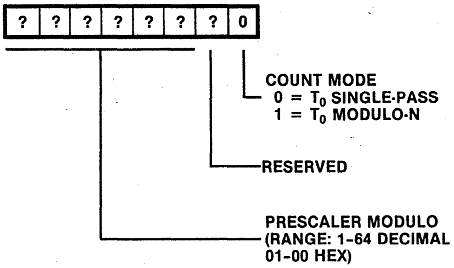


Figure 11-21. Prescaler 0 Reset

R241 TMR
Timer Mode Register
(% F1; Read/Write)

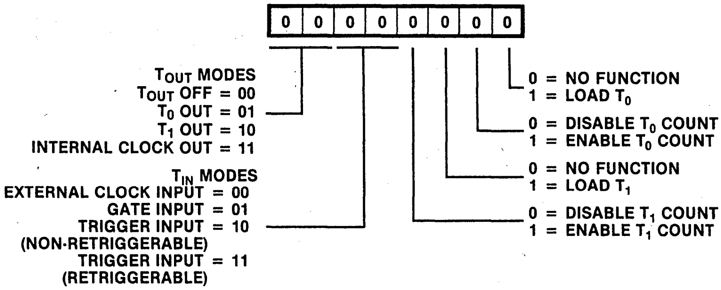


Figure 11-22. Timer Mode Register Reset

Chapter 12
Serial I/O

12.1 INTRODUCTION

The Z8 microcomputer contains an on-board full-duplex receiver/transmitter for asynchronous data communications. The receiver/transmitter consists of a Serial I/O register SIO (%F1) and its associated control logic (Figure 12-1). The SIO is actually two registers--the receiver buffer and the transmitter buffer--which are used in conjunction with counter/timer T_0 and Port 3 I/O lines $P3_0$ (input) and $P3_7$ (output). Counter/timer T_0 provides the clock input for control of the data rates.

Configuration of the serial I/O is controlled by the Port 3 Mode register, P3M. The Z8 always transmits 8 bits between the start and stop bits; that is, 8 data bits or 7 data bits and 1 parity bit. Odd parity generation and detection is supported.

The Serial I/O register and its associated Mode Control registers are mapped into the register file as shown in Figure 12-2. This organization allows the software to access the serial I/O as general-purpose registers, eliminating the need for special instructions.

12.2 BIT RATE GENERATION

When Port 3 Mode register bit D_6 is set to 1, the serial I/O is enabled and T_0 automatically becomes the bit rate generator (Figure 12-3). T_0's end-of-count signal no longer generates interrupt request IRQ_4; instead, the signal is used as the input to the divide-by-16 counters (one each for the receiver and the transmitter) which clock the data stream.

The divide chain that generates the bit rate is shown in Figure 12-4. The bit rate is given by the following equation:

$$\text{bit rate} = \text{XTAL frequency}/(2 \times 4 \times p \times t \times 16)$$

where p and t are the initial values in the Prescaler and Counter/Timer registers, respectively.

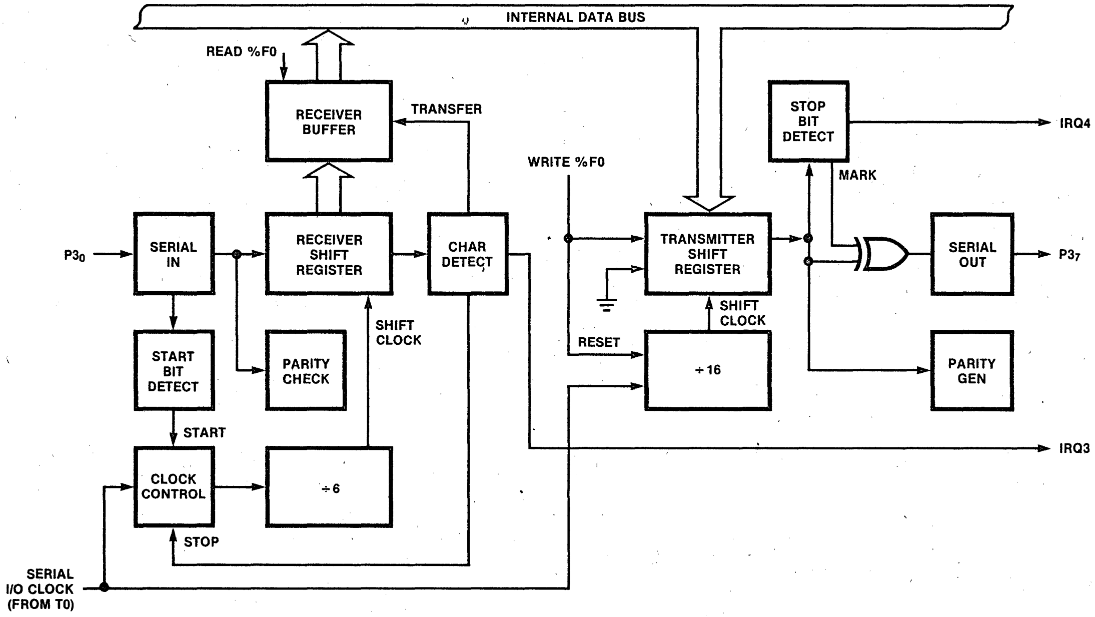


Figure 12-1. Serial I/O Block Diagram

The final divide-by-16 is required since T_0 runs at 16 times the bit rate in order to synchronize on the incoming data.

To configure the Z8 for a specific bit rate, appropriate values as determined by the above equation must be loaded into registers PRE_0 (%F5) and T_0 (%F4). PRE_0 also controls the counting mode for T_0 and should therefore be set to the Continuous mode (D_1 set to 1).

For example, given an input clock frequency (fXTAL) of 11.9808 MHz and a selected bit rate of 1200 bits per second, the equation is satisfied by p=39 and t=2. Counter/timer T_0 should be set to %02. With T_0 in Continuous mode, the value of PRE_0 becomes %9D (Figure 12-5).

Table 12-1 lists several commonly used bit rates and the values of fXTAL, p, and t required to derive them. This list is presented for convenience and is not intended to be exhaustive.

The bit rate generator is started by setting the Timer Mode register TMR (%F1) bits D_1 and D_0 both to 1 (Figure 12-6). This transfers the contents of the Prescaler and Counter/Timer registers to their corresponding down-counters. In addition, counting is enabled so that serial I/O operations begin.

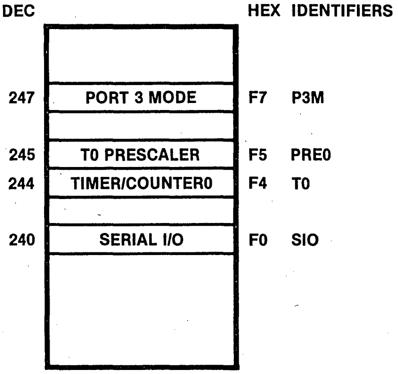


Figure 12-2. Serial I/O Register Map

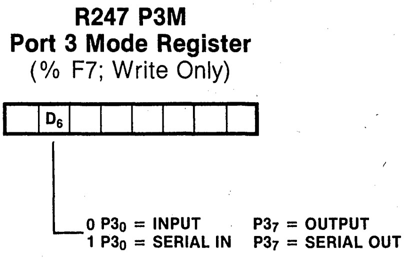

R247 P3M
Port 3 Mode Register
(% F7; Write Only)
D6
0 P30 = INPUT P37 = OUTPUT
1 P30 = SERIAL IN P37 = SERIAL OUT

Figure 12-3. Port 3 Mode Register and Bit Rate Generation

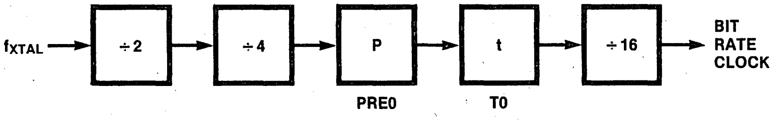


Figure 12-4. Bit Rate Divide Chain

Table 12-1. Bit Rate

Bit Rate	7,3728		7,9872		9,8304		11,0592		11,6736		11,9808		12,2880	
	p	t	p	t	p	t	p	t	p	t	p	t	p	t
19200	3	1	--	--	4	1	--	--	--	--	--	--	5	1
9600	3	2	--	--	4	2	9	1	--	--	--	--	5	2
4800	3	4	13	1	4	4	9	2	19	1	--	--	5	4
2400	3	8	13	2	4	8	9	4	19	2	39	1	5	8
1200	3	16	13	4	4	16	9	8	19	4	39	2	5	16
600	3	32	13	8	4	32	9	16	19	8	39	4	5	32
300	3	64	13	16	4	64	9	32	19	16	39	8	5	64
150	3	128	13	32	4	128	9	64	19	32	39	16	5	128
110	3	175	3	189	4	175	5	157	4	207	17	50	8	109

R245 PRE0
Prescaler 0 Register
(% F5; Write Only)

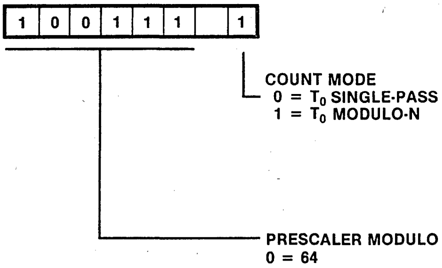


Figure 12-5. Prescaler 0 Register and Bit Rate Generation

R241 TMR
Timer Mode Register
(% F1; Read/Write)

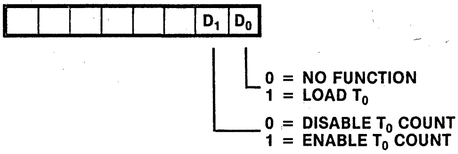


Figure 12-6. Timer Mode Register and Bit Rate Generation

12.3 RECEIVER OPERATION

The receiver consists of a receiver buffer (SIO [%F0]), a serial-in, parallel-out Shift register, parity checking, and data synchronizing logic. The receiver block diagram is shown as part of Figure 12-1.

12.3.1 Receiver Shift Register

After a hardware reset or after a character has been received, the Receiver Shift register is initialized to all 1s and the shift clock is stopped. Serial data, input through Port 3 pin $P3_0$, is synchronized to the internal clock by two D-type flip flops before being input to the Shift register and the start bit detection circuitry.

The start bit detection circuitry monitors the incoming data stream, looking for a start bit (a High-to-Low input transition). When a start bit is detected, the shift clock logic is enabled. The T_0 input is divided by 16 and, when the count equals 8, the divider outputs a shift clock. This clock shifts the start bit into the Receiver Shift register at the center of the bit time. Before the shift actually occurs, the input is rechecked to ensure that the start bit is valid. If the detected start bit is false, the receiver is reset and the process of looking for a start bit is repeated. If the start bit is valid, the data is shifted into the Shift register every sixteen counts until a full character is assembled (Figure 12-7).

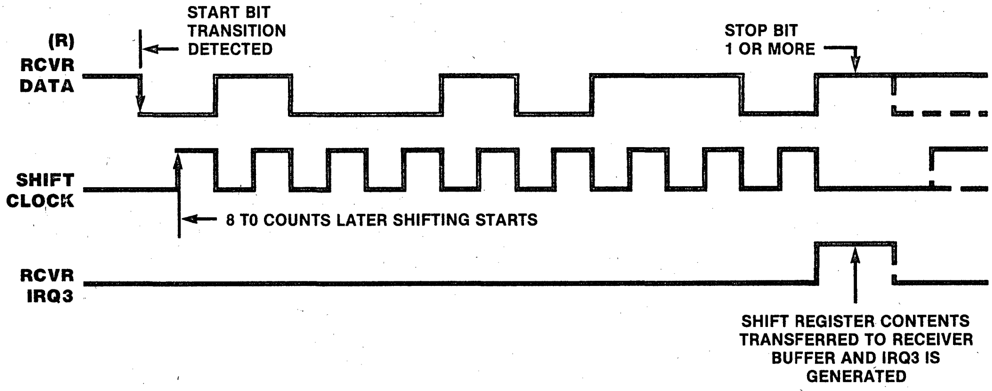

START BIT TRANSITION DETECTED
(R) RCVR DATA
STOP BIT 1 OR MORE
SHIFT CLOCK
8 T0 COUNTS LATER SHIFTING STARTS
RCVR IRQ3
SHIFT REGISTER CONTENTS TRANSFERRED TO RECEIVER BUFFER AND IRQ3 IS GENERATED

Figure 12-7. Receiver Timing

After a full character has been assembled in the Shift register, the data is transferred to the receiver's buffer, SIO (%F0), and interrupt request IRQ_3 is generated. The shift clock is stopped and the Shift register reset to all 1s. The start bit detection circuitry begins monitoring the data input for the next start bit. This cycle allows the receiver to synchronize on the center of the bit time for each incoming character.

12.3.2 Overwrites

Although the receiver is buffered, it is not protected from being overwritten, so the software must read the SIO register within one character time after the interrupt request. The Z8 does not have a flag to indicate this overrun condition. If polling is used, the IRQ_3 bit in the Interrupt Request register must be reset by software.

12.3.3 Framing Errors

Framing error detection is not supported by the receiver hardware, but by responding to the interrupt request within one character bit time, the software can test for a stop bit at $P3_0$. Port 3 bits are always readable, which facilitates break detection. For example, if a null character is received, testing $P3_0$ results in a 0 being read.

12.3.4 Parity

The data format supported by the receiver must have a start bit, eight data bits, and at least one stop bit. If parity is on, bit D_7 of the data received will be replaced by a Parity Error flag. A parity error sets D_7 to 1; otherwise, D_7 is set to 0. Figure 12-8 shows these data formats.

The Z8 hardware supports odd parity only, which is enabled by setting Port 3 Mode register bit D_7 to 1 (Figure 12-9). If even parity is required, the Parity mode should be disabled (i.e. P3M D_7 set to 0), and software must calculate the received data's parity.

12.4 TRANSMITTER OPERATION

The transmitter consists of a transmitter buffer (SIO (%F0)), a parity generator, and associated control logic. The transmitter block diagram is shown as part of Figure 12-1.

After a hardware reset or after a character has been transmitted, the transmitter is forced to a marking state (output always High) until a character is loaded into the transmitter buffer, SIO (%F0). The transmitter is loaded by specifying the SIO as the destination register of any instruction.

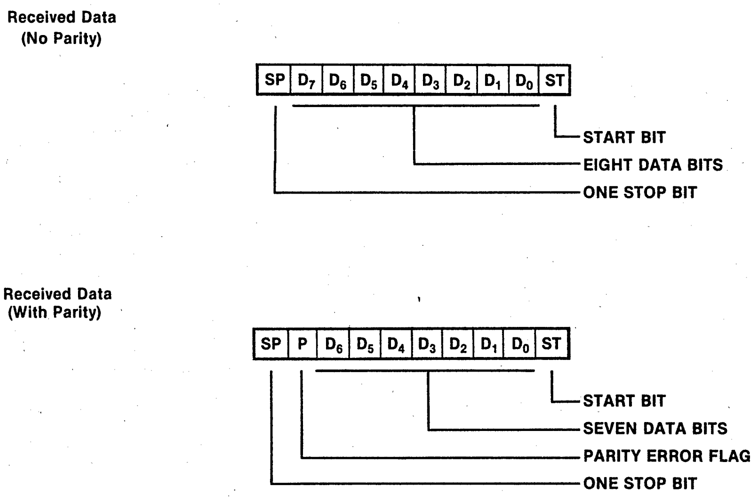


Figure 12-8. Receiver Data Formats

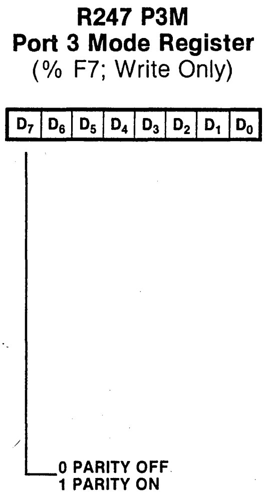


Figure 12-9. Parity and Port 3 Mode Register

T_0's output drives a divide-by-16 counter which in turn generates a shift clock every 16 counts. This counter is reset when the transmitter buffer is written by an instruction. This reset synchronizes the shift clock to the software. The transmitter then outputs one bit per shift clock, through Port 3 pin $P3_7$, until a start bit, the character written to the buffer, and two stop bits have been transmitted. After the second stop bit has been transmitted, the output is again forced to a marking state. Interrupt request IRQ_4 is generated and this notifies the processor that the transmitter is ready to accept another character.

12.4.1 Overwrites

The user is not protected from overwriting the transmitter, so it is up to the software to respond to IRQ_4 appropriately. If polling is used, the IRQ_4 bit in the Interrupt Request register must be reset.

12.4.2 Parity

The data format supported by the transmitter has a start bit, eight data bits, and at least two stop bits. If parity is on, bit D_7 of the data transmitted will be replaced by an odd parity bit. Figure 12-10 shows the transmitter data formats.

Parity is enabled by setting Port 3 Mode register bit D_7 to 1. If even parity is required, the parity mode should be disabled (i.e. P3M D_7 set to 0), and software must modify the data to include even parity.

Since the transmitter can be overwritten, the user is able to generate a break signal. This is done by writing null characters to the transmitter buffer (SIO, %F0) at a rate which does not allow the stop bits to be output. Each time the SIO is loaded, the divide-by-16 counter is re-synchronized and a new start bit is output followed by data.

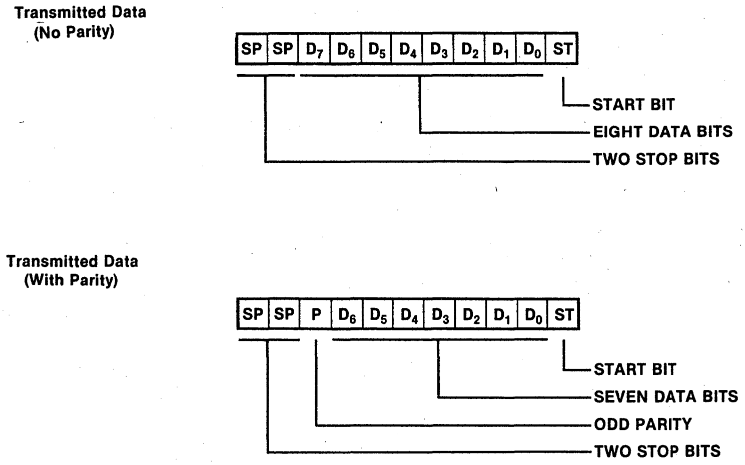


Figure 12-10. Transmitter Data Formats

12.5 RESET CONDITIONS

After a hardware reset, the Serial I/O register contents are undefined, and Serial mode and parity are disabled. Figures 12-11 and 12-12 show the binary reset values of the Serial I/O register and its associated mode register P3M.

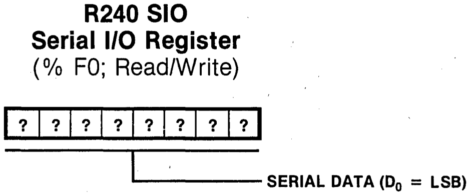


Figure 12-11. Serial I/O Register Reset

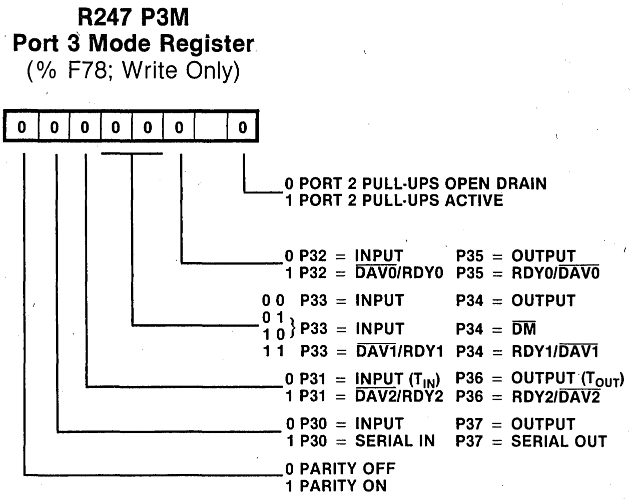

R247 P3M
Port 3 Mode Register
(% F78; Write Only)
0 0 0 0 0 0 0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
0 P32 = INPUT P35 = OUTPUT
1 P32 = $\overline{\text{DAV0}}$/RDY0 P35 = RDY0/$\overline{\text{DAV0}}$
0 0 P33 = INPUT P34 = OUTPUT
0 1
1 0 } P33 = INPUT P34 = $\overline{\text{DM}}$
1 1 P33 = $\overline{\text{DAV1}}$/RDY1 P34 = RDY1/$\overline{\text{DAV1}}$
0 P31 = INPUT (T_{IN}) P36 = OUTPUT (T_{OUT})
1 P31 = $\overline{\text{DAV2}}$/RDY2 P36 = RDY2/$\overline{\text{DAV2}}$
0 P30 = INPUT P37 = OUTPUT
1 P30 = SERIAL IN P37 = SERIAL OUT
0 PARITY OFF
1 PARITY ON

Figure 12-12. Port 3 Register Reset

A

Zilog

Appendix A Pin Descriptions and Functions

This appendix contains pin information and physical descriptions for the Z8 development device (Z8612) and Protopack emulator (Z8603/13). Pin descriptions for the Z8601/11 and Z8681/82 microcomputers can be found in Chapters 6 and 7, respectively.

A.1 DEVELOPMENT DEVICE (Z8612)

The pin mnemonics and descriptions presented for the Z8 microcomputers (Chapter 6) also apply to the development device. Additional pin descriptions are as follows:

A_0-A_{11}. Program Memory Address (outputs). These lines are used to access the first 4K bytes of the external program memory.

D_0-D_7. Program Data (inputs). Data from the external program memory is input through these pins.

$\overline{\text{IACK}}$. Interrupt Acknowledge (output, active High). $\overline{\text{IACK}}$ is driven High in response to an interrupt during the interrupt machine cycle.

$\overline{\text{MDS}}$. Program Memory Data Strobe (output, active Low). $\overline{\text{MDS}}$ is Low during an instruction fetch cycle when the first 4K bytes of program memory are being accessed.

SCLK. System Clock (output). SCLK is the internal clock output through a buffer. The clock rate is equal to one-half the crystal frequency.

$\overline{\text{SYNC}}$. Instruction Sync (output, active Low). This strobe output is forced Low during the internal clock period preceding an opcode fetch.

A.2 PROTOPACK EMULATOR (Z8603/13)

Both the Z8603 and Z8613 devices use a 40-pin package that also has a 24-pin "piggy-back" socket. An EPROM or ROM can be installed on the back of the emulator's standard 40-pin package via the socket (Figure A-3). A single +5 V dc power source is required. Figure A-4 illustrates the pinout for the socket carried piggyback. The socket is designed to accept a 2716 EPROM for the Z8603 and a 2732 EPROM for the Z8613 device.

Pin mnemonics and descriptions are the same as those for the Z8601/11 microcomputer (Chapter 6). Descriptions for the additional (24-pin socket) memory interface lines are the same as those given for the development devices above.

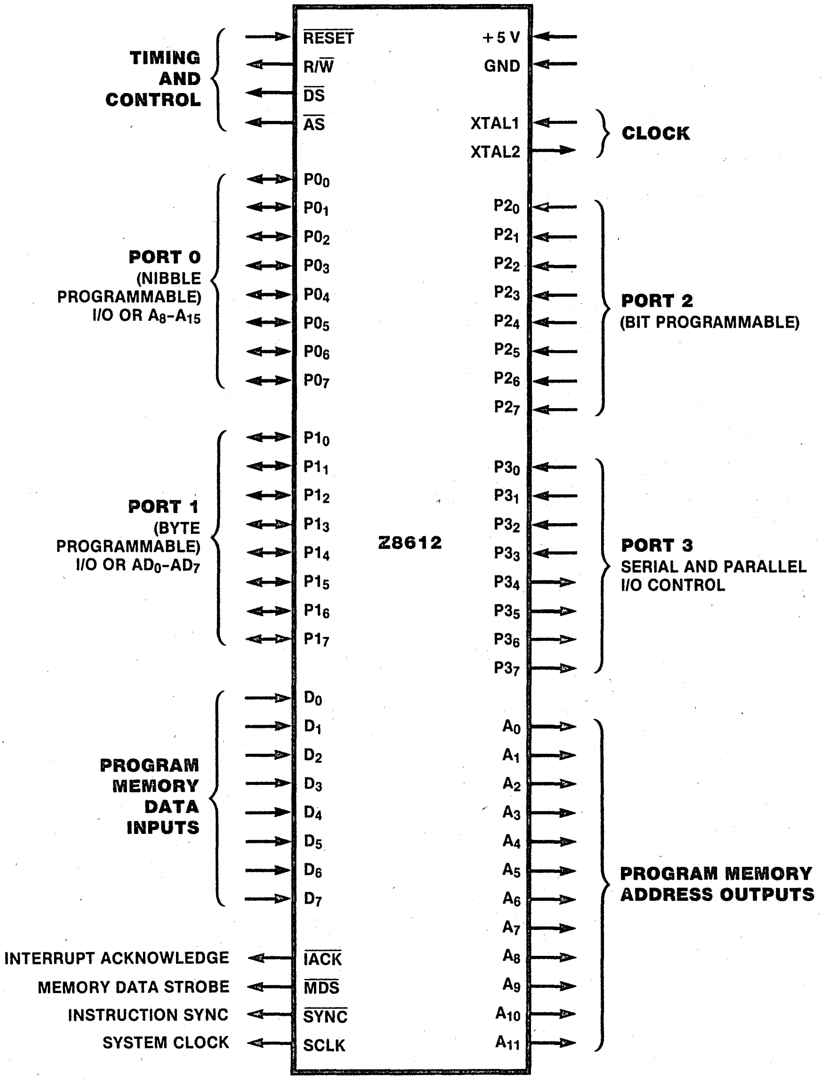


Figure A-1. Z8612 Pin Functions

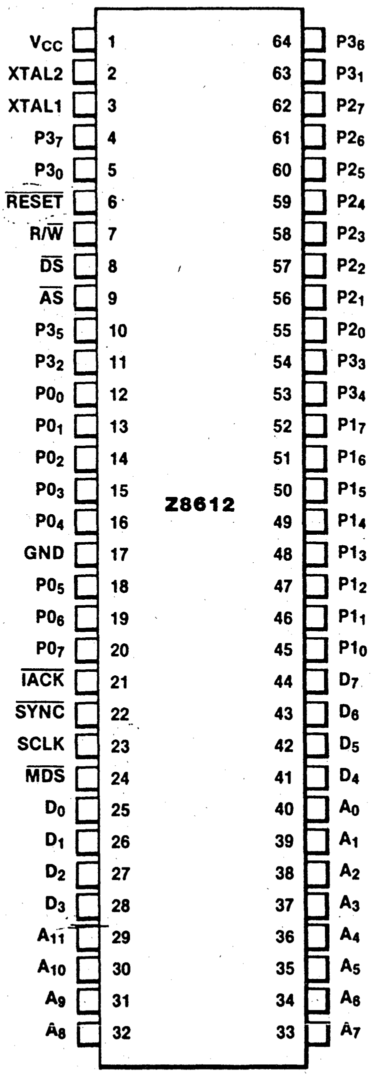


Figure A-2. Z8612 Pin Assignments

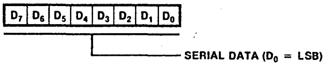
B

Zilog

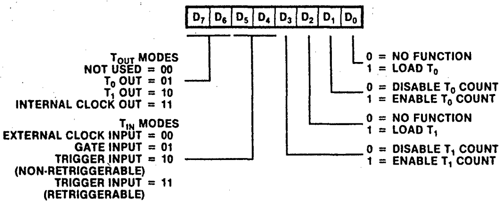
Appendix B
Control Registers

Registers

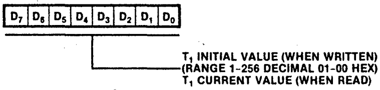
R240 SIO
Serial I/O Register
($F0_H$; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
SERIAL DATA (D_0 = LSB)

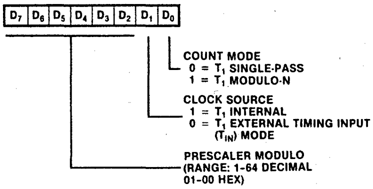
R241 TMR
Timer Mode Register
($F1_H$; Read/Write)



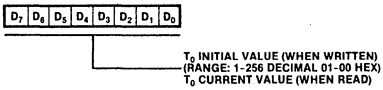
R242 T1
Counter Timer 1 Register
($F2_H$; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_1 INITIAL VALUE (WHEN WRITTEN)
(RANGE 1-256 DECIMAL 01-00 HEX)
T_1 CURRENT VALUE (WHEN READ)

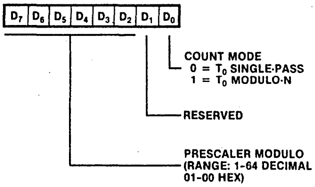
R243 PRE1
Prescaler 1 Register
($F3_H$; Write Only)



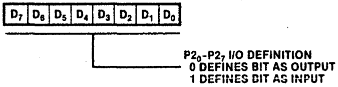
R244 T0
Counter/Timer 0 Register
($F4_H$; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
T_0 INITIAL VALUE (WHEN WRITTEN)
(RANGE: 1-256 DECIMAL 01-00 HEX)
T_0 CURRENT VALUE (WHEN READ)

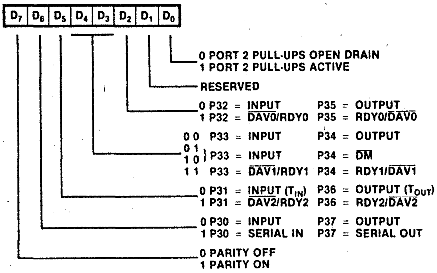
R245 PRE0
Prescaler 0 Register
($F5_H$; Write Only)



R246 P2M
Port 2 Mode Register
($F6_H$; Write Only)

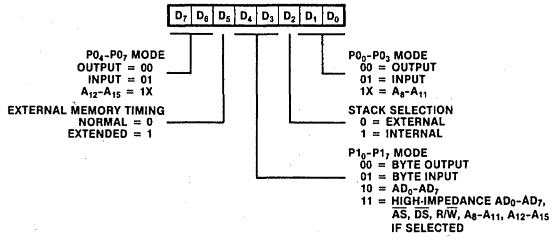

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
$P2_0$-$P2_7$ I/O DEFINITION
0 DEFINES BIT AS OUTPUT
1 DEFINES BIT AS INPUT

R247 P3M
Port 3 Mode Register
($F7_H$; Write Only)

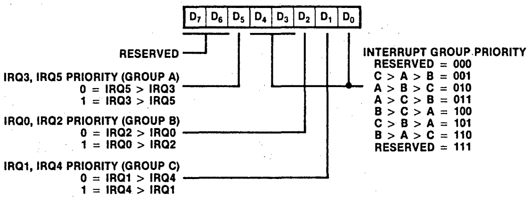

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
RESERVED
0 P32 = INPUT P35 = OUTPUT
1 P32 = $\overline{DAV0}$/RDY0 P35 = RDY0/$\overline{DAV0}$
0 0 P33 = INPUT P34 = OUTPUT
0 1
1 0 } P33 = INPUT P34 = $\overline{DM}$
1 1 P33 = $\overline{DAV1}$/RDY1 P34 = RDY1/$\overline{DAV1}$
0 P31 = INPUT (T_{IN}) P36 = OUTPUT (T_{OUT})
1 P31 = $\overline{DAV2}$/RDY2 P36 = RDY2/$\overline{DAV2}$
0 P30 = INPUT P37 = OUTPUT
1 P30 = SERIAL IN P37 = SERIAL OUT
0 PARITY OFF
1 PARITY ON

Registers
(Continued)

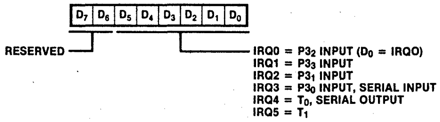
R248 P01M
Port 0 and 1 Mode Register
($F8_H$; Write Only)



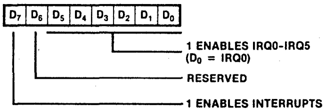
R249 IPR
Interrupt Priority Register
($F9_H$; Write Only)



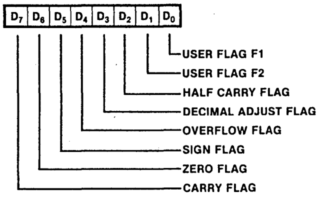
R250 IRQ
Interrupt Request Register
(FA_H; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
RESERVED
IRQ0 = $P3_2$ INPUT (D_0 = IRQ0)
IRQ1 = $P3_3$ INPUT
IRQ2 = $P3_1$ INPUT
IRQ3 = $P3_0$ INPUT, SERIAL INPUT
IRQ4 = T_0, SERIAL OUTPUT
IRQ5 = T_1

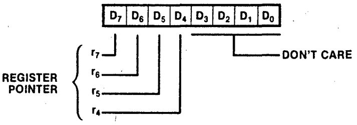
R251 IMR
Interrupt Mask Register
(FB_H; Read/Write)



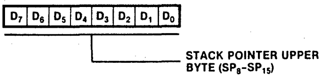
R252 FLAGS
Flag Register
(FC_H; Read/Write)



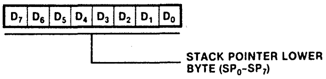
R253 RP
Register Pointer
(FD_H; Read/Write)



R254 SPH
Stack Pointer
(FE_H; Read/Write)


D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
STACK POINTER UPPER BYTE (SP_8-SP_{15})

R255 SPL
Stack Pointer
(FF_H; Read/Write)

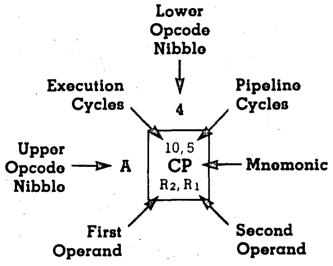

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
STACK POINTER LOWER BYTE (SP_0-SP_7)

Zilog

Opcode Map

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6,5 **DEC** R_1	6,5 **DEC** IR_1	6,5 **ADD** r_1,r_2	6,5 **ADD** r_1,Ir_2	10,5 **ADD** R_2,R_1	10,5 **ADD** IR_2,R_1	10,5 **ADD** R_1,IM	10,5 **ADD** IR_1,IM	6,5 **LD** r_1,R_2	6,5 **LD** r_2,R_1	12/10,5 **DJNZ** r_1,RA	12/10,0 **JR** cc,RA	6,5 **LD** r_1,IM	12/10,0 **JP** cc,DA	6,5 **INC** r_1	
1	6,5 **RLC** R_1	6,5 **RLC** IR_1	6,5 **ADC** r_1,r_2	6,5 **ADC** r_1,Ir_2	10,5 **ADC** R_2,R_1	10,5 **ADC** IR_2,R_1	10,5 **ADC** R_1,IM	10,5 **ADC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
2	6,5 **INC** R_1	6,5 **INC** IR_1	6,5 **SUB** r_1,r_2	6,5 **SUB** r_1,Ir_2	10,5 **SUB** R_2,R_1	10,5 **SUB** IR_2,R_1	10,5 **SUB** R_1,IM	10,5 **SUB** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
3	8,0 **JP** IRR_1	6,1 **SRP** IM	6,5 **SBC** r_1,r_2	6,5 **SBC** r_1,Ir_2	10,5 **SBC** R_2,R_1	10,5 **SBC** IR_2,R_1	10,5 **SBC** R_1,IM	10,5 **SBC** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
4	8,5 **DA** R_1	8,5 **DA** IR_1	6,5 **OR** r_1,r_2	6,5 **OR** r_1,Ir_2	10,5 **OR** R_2,R_1	10,5 **OR** IR_2,R_1	10,5 **OR** R_1,IM	10,5 **OR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
5	10,5 **POP** R_1	10,5 **POP** IR_1	6,5 **AND** r_1,r_2	6,5 **AND** r_1,Ir_2	10,5 **AND** R_2,R_1	10,5 **AND** IR_2,R_1	10,5 **AND** R_1,IM	10,5 **AND** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
6	6,5 **COM** R_1	6,5 **COM** IR_1	6,5 **TCM** r_1,r_2	6,5 **TCM** r_1,Ir_2	10,5 **TCM** R_2,R_1	10,5 **TCM** IR_2,R_1	10,5 **TCM** R_1,IM	10,5 **TCM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
7	10/12,1 **PUSH** R_2	12/14,1 **PUSH** IR_2	6,5 **TM** r_1,r_2	6,5 **TM** r_1,Ir_2	10,5 **TM** R_2,R_1	10,5 **TM** IR_2,R_1	10,5 **TM** R_1,IM	10,5 **TM** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	
8	10,5 **DECW** RR_1	10,5 **DECW** IR_1	12,0 **LDE** r_1,Irr_2	18,0 **LDEI** Ir_1,Irr_2					↓	↓	↓	↓	↓	↓	↓	6,1 **DI**
9	6,5 **RL** R_1	6,5 **RL** IR_1	12,0 **LDE** r_2,Irr_1	18,0 **LDEI** Ir_2,Irr_1					↓	↓	↓	↓	↓	↓	↓	6,1 **EI**
A	10,5 **INCW** RR_1	10,5 **INCW** IR_1	6,5 **CP** r_1,r_2	6,5 **CP** r_1,Ir_2	10,5 **CP** R_2,R_1	10,5 **CP** IR_2,R_1	10,5 **CP** R_1,IM	10,5 **CP** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	14,0 **RET**
B	6,5 **CLR** R_1	6,5 **CLR** IR_1	6,5 **XOR** r_1,r_2	6,5 **XOR** r_1,Ir_2	10,5 **XOR** R_2,R_1	10,5 **XOR** IR_2,R_1	10,5 **XOR** R_1,IM	10,5 **XOR** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	16,0 **IRET**
C	6,5 **RRC** R_1	6,5 **RRC** IR_1	12,0 **LDC** r_1,Irr_2	18,0 **LDCI** Ir_1,Irr_2				10,5 **LD** r_1, x, R_2	↓	↓	↓	↓	↓	↓	↓	6,5 **RCF**
D	6,5 **SRA** R_1	6,5 **SRA** IR_1	12,0 **LDC** r_2,Irr_1	18,0 **LDCI** Ir_2,Irr_1	20,0 **CALL*** IRR_1		20,0 **CALL** DA	10,5 **LD** r_2, x, R_1	↓	↓	↓	↓	↓	↓	↓	6,5 **SCF**
E	6,5 **RR** R_1	6,5 **RR** IR_1		6,5 **LD** r_1,Ir_2	10,5 **LD** R_2,R_1	10,5 **LD** IR_2,R_1	10,5 **LD** R_1,IM	10,5 **LD** IR_1,IM	↓	↓	↓	↓	↓	↓	↓	6,5 **CCF**
F	8,5 **SWAP** R_1	8,5 **SWAP** IR_1		6,5 **LD** Ir_1,r_2		10,5 **LD** R_2,IR_1			↓	↓	↓	↓	↓	↓	↓	6,0 **NOP**
Bytes per Instruction	2				3				2					3	1	



Legend:
R = 8-Bit Address
r = 4-Bit Address
R_1 or r_1 = Dst Address
R_2 or r_2 = Src Address

Sequence:
Opcode, First Operand, Second Operand

Note: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

January 1988

Super8™ MCU ROMless, ROM, and Prototyping Device with EPROM Interface

Z8800, Z8801, Z8820, Z8822

FEATURES

- Improved Z8® instruction set includes multiply and divide instructions, Boolean and BCD operations.
- Additional instructions support threaded-code languages, such as "Forth."
- 325 byte registers, including 272 general-purpose registers, and 53 mode and control registers.
- Addressing of up to 128K bytes of memory.
- Two register pointers allow use of short and fast instructions to access register groups within 600 nsec.
- Direct Memory Access controller (DMA).
- Two 16-bit counter/timers.
- Up to 32 bit-programmable and 8 byte-programmable I/O lines, with 2 handshake channels.
- Interrupt structure supports:
  - □ 27 interrupt sources
  - □ 16 interrupt vectors (2 reserved for future versions)
  - □ 8 interrupt levels
  - □ Servicing in 600 nsec. (1 level only)
- Full-duplex UART with special features.
- On-chip oscillator.
- 20 MHz clock.
- 8K byte ROM for Z8820

GENERAL DESCRIPTION

The Zilog Super8 single-chip MCU can be used for development and production. It can be used as I/O- or memory-intensive computers, or configured to address external memory while still supporting many I/O lines.

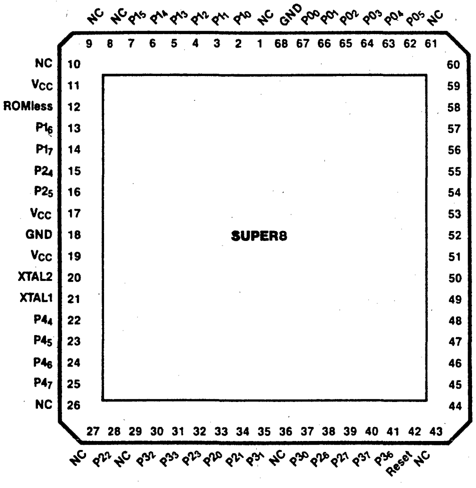
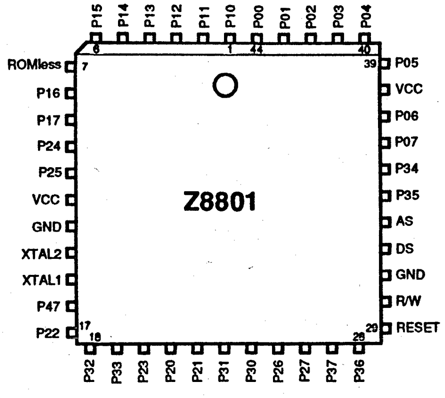


Figure 1a. Pin Assignments — 68-pin PLCC

The Super8 features a full-duplex universal asynchronous receiver/transmitter (UART) with on-chip baud rate generator, two programmable counter/timers, a direct memory access (DMA) controller, and an on-chip oscillator.

The Super8 is also available as a 48-pin and 68-pin ROMless microcomputer with four byte-wide I/O ports plus a byte-wide address/data bus. Additional address bits can be configured, up to a total of 16.



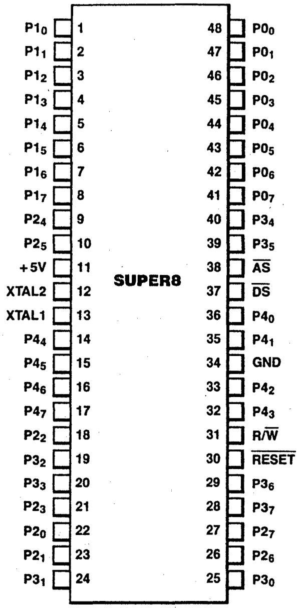


Figure 1b. Pin Assignments — 48-pin DIP

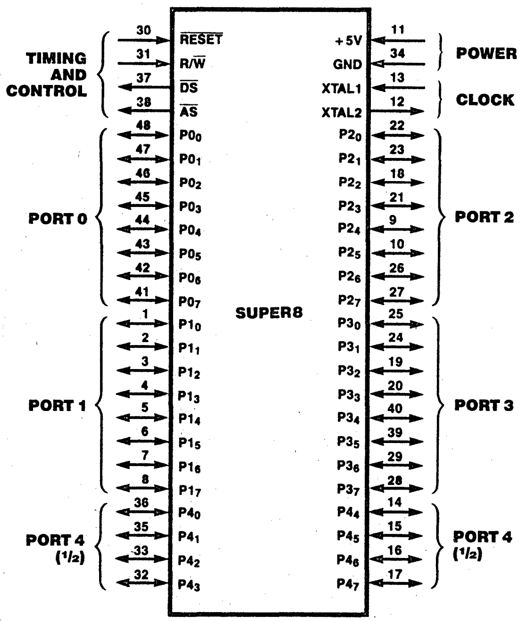
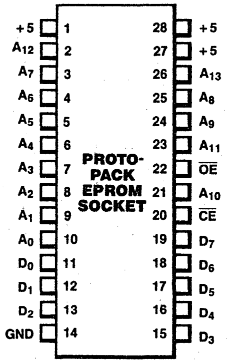


Figure 2. Pin Functions



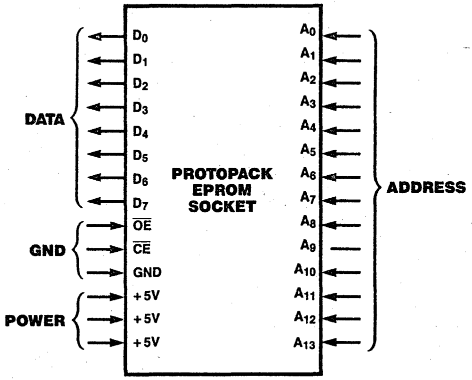


Figure 3. Pin Assignments—28-Pin Piggyback Socket

Figure 4. Pin Functions—28-Pin Piggyback Socket

Protopack

This part functions as an emulator for the basic microcomputer. It uses the same package and pin-out as the basic microcomputer but also has a 28-pin "piggy back" socket on the top into which a ROM or EPROM can be installed. The socket is designed to accept a type 2764 EPROM.

This package permits the protopack to be used in prototype and final PC boards while still permitting user program development. When a final program is developed, it can be mask-programmed into the production microcomputer device, directly replacing the emulator. The protopack part is also useful in situations where the cost of mask-programming is prohibitive or where program flexibility is desired.

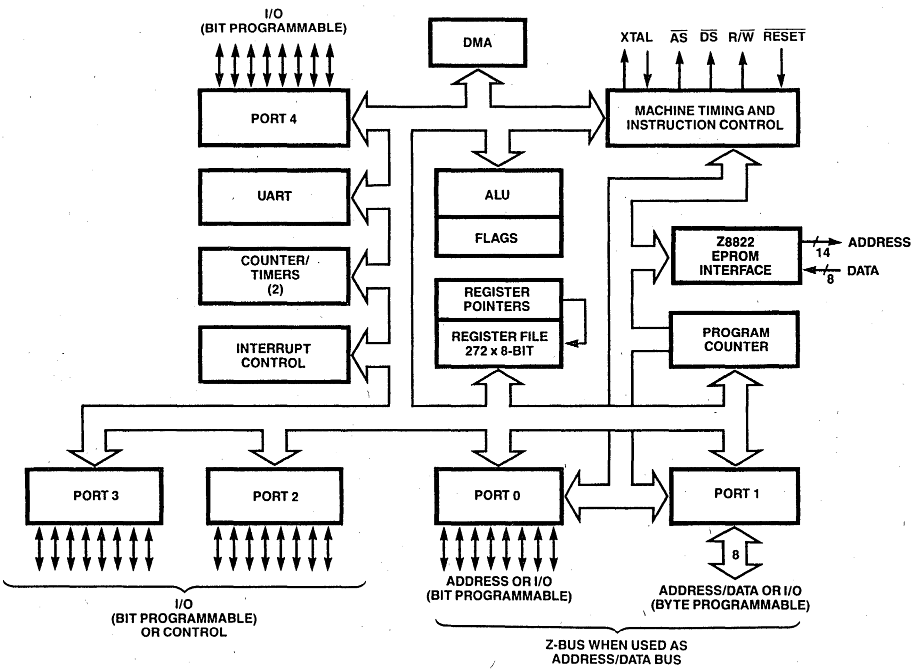


Figure 5. Functional Block Diagram

ARCHITECTURE

The Super8 architecture includes 325 byte-wide internal registers. 272 of these are available for general purpose use; the remaining 53 provide control and mode functions.

The instruction set is specially designed to deal with this large register set. It includes a full complement of 8-bit arithmetic and logical operations, including multiply and divide instructions and provisions for BCD operations. Addresses and counters can be incremented and decremented as 16-bit quantities. Rotate, shift, and bit manipulation instructions are provided. Three new instructions support threaded-code languages.

The UART is a full-function multipurpose asynchronous serial channel with many premium features.

The 16-bit counters can operate independently or be cascaded to perform 32-bit counting and timing operations. The DMA controller handles transfers to and from the register file or memory. DMA can use the UART or one of two ports with handshake capability.

The architecture appears in the block diagram (Figure 5).

PIN DESCRIPTIONS

The Super8 connects to external devices via the following TTL-compatible pins:

$\overline{AS}$. *Address Strobe* (output, active Low). $\overline{AS}$ is pulsed Low once at the beginning of each machine cycle. The rising edge indicates that addresses R/$\overline{W}$ and $\overline{DM}$, when used, are valid.

$\overline{DS}$. *Data Strobe* (output, active Low). $\overline{DS}$ provides timing for data movement between the address/data bus and external memory. During write cycles, data output is valid at the leading edge of $\overline{DS}$. During read cycles, data input must be valid prior to the trailing edge of $\overline{DS}$.

$P0_0$-$P0_7$, $P1_0$-$P1_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$, $P4_0$-$P4_7$. *Port I/O Lines* (input/output). These 40 lines are divided into five 8-bit I/O ports that can be configured under program control for I/O or external memory interface.

In the ROMless devices, Port 1 is dedicated as a multiplexed address/data port, and Port 0 pins can be assigned as additional address lines; Port 0 non-address pins may be assigned as I/O. In the ROM and protopack, Port 1 can be assigned as input or output, and Port 0 can be assigned as input or output on a bit by bit basis.

Ports 2 and 3 can be assigned on a bit-for-bit basis as general I/O or interrupt lines. They can also be used as special-purpose I/O lines to support the UART, counter/timers, or handshake channels.

Port 4 is used for general I/O.

During reset, all port pins are configured as inputs (high impedance) except for Port 1 and Port 0 in the ROMless devices. In these, Port 1 is configured as a multiplexed address/data bus, and Port 0 pins $P0_0$-$P0_4$ are configured as address out, while pins $P0_5$-$P0_7$ are configured as inputs.

$\overline{\text{RESET}}$. *Reset* (input, active Low). Reset initializes and starts the Super8. When it is activated, it halts all processing; when it is deactivated, the Super8 begins processing at address 0020_H.

ROMless. (input, active High). This input controls the operation mode of a 68-pin Super8. When connected to V_{CC}, the part will function as a ROMless Z8800. When connected to GND, the part will function as a Z8820 ROM part.

R/$\overline{\text{W}}$. *Read/Write* (output). R/$\overline{\text{W}}$ determines the direction of data transfer for external memory transactions. It is Low when writing to program memory or data memory, and High for everything else.

XTAL1, XTAL2. (Crystal oscillator input.) These pins connect a parallel resonant crystal or an external clock source to the on-board clock oscillator and buffer.

REGISTERS

The Super8 contains a 256-byte internal register space. However, by using the upper 64 bytes of the register space more than once, a total of 325 registers are available.

Registers from 00 to BF are used only once. They can be accessed by any register command. Register addresses C0 to FF contain two separate sets of 64 registers. One set, called control registers, can only be accessed by register direct commands. The other set can only be addressed by register indirect, indexed, stack, and DMA commands.

The uppermost 32 register direct registers (E0 to FF) are further divided into two banks (0 and 1), selected by the Bank Select bit in the Flag register. When a Register Direct command accesses a register between E0 and FF, it looks at the Bank Select bit in the Flag register to select one of the banks.

The register space is shown in Figure 6.

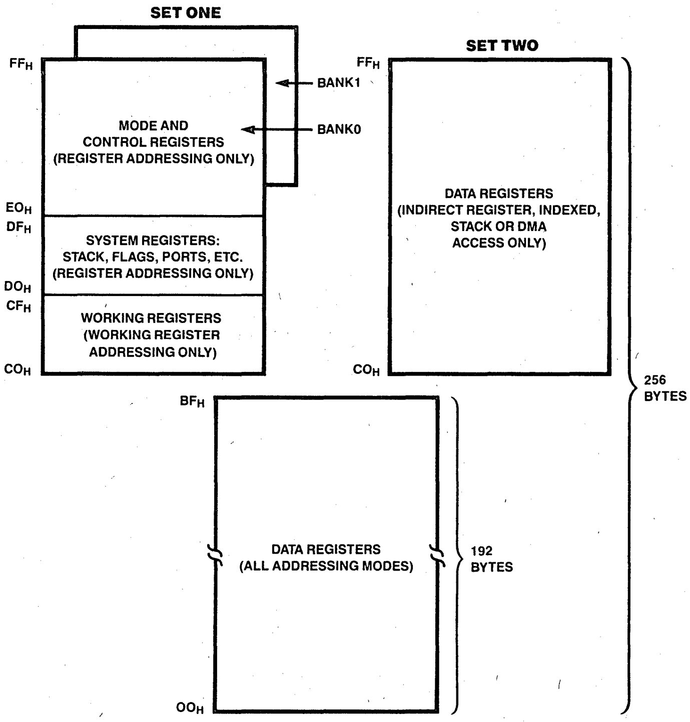

SET ONE
FFH
BANK1
MODE AND CONTROL REGISTERS (REGISTER ADDRESSING ONLY)
BANK0
EOH
DFH
SYSTEM REGISTERS: STACK, FLAGS, PORTS, ETC. (REGISTER ADDRESSING ONLY)
DOH
CFH
WORKING REGISTERS (WORKING REGISTER ADDRESSING ONLY)
COH
SET TWO
FFH
DATA REGISTERS (INDIRECT REGISTER, INDEXED, STACK OR DMA ACCESS ONLY)
COH
256 BYTES
BFH
DATA REGISTERS (ALL ADDRESSING MODES)
192 BYTES
OOH

Figure 6. Super8 Registers

Working Register Window

Control registers R214 and R215 are the register pointers, RP0 and RP1. They each define a moveable, 8-register section of the register space. The registers within these spaces are called working registers.

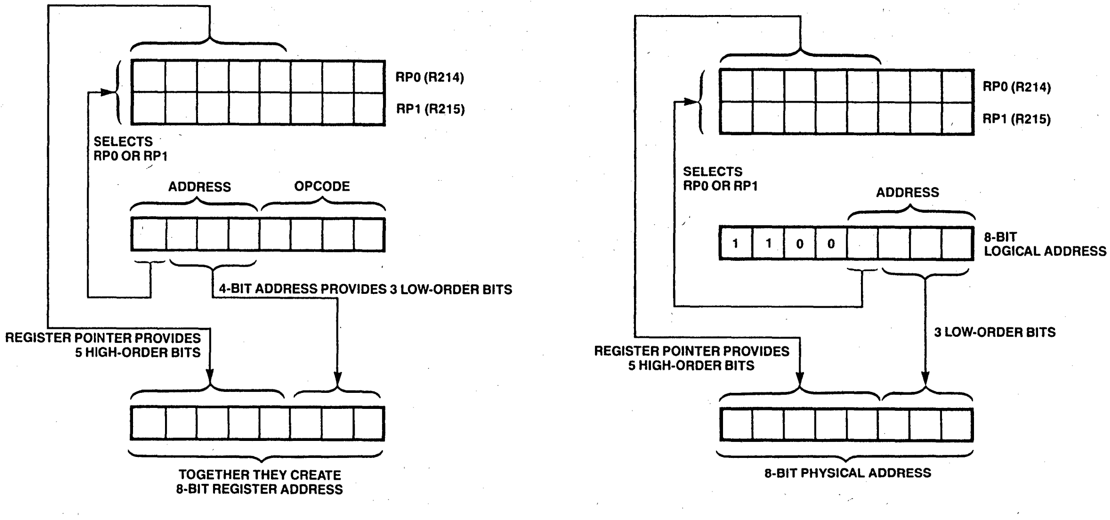
Working registers can be accessed using short 4-bit addresses. The process, shown in section a of Figure 4, works as follows:

- The high-order bit of the 4-bit address selects one of the two register pointers (0 selects RP0; 1 selects RP1).
- The five high-order bits in the register pointer select an 8-register (contiguous) slice of the register space.
- The three low-order bits of the 4-bit address select orte of the eight registers in the slice.

The net effect is to concatenate the five bits from the register pointer to the three bits from the address to form an 8-bit address. As long as the address in the register pointer remains unchanged, the three bits from the address will always point to an address within the same eight registers.

The register pointers can be moved by changing the five high bits in control registers R214 for RP0 and R215 for RP1.

The working registers can also be accessed by using full 8-bit addressing. When an 8-bit logical address in the range 192 to 207 (C0 to CF) is specified, the lower nibble is used similarly to the 4-bit addressing described above. This is shown in section b of Figure 7.



a. 4-Bit Addressing

b. 8-Bit Addressing

Figure 7. Working Register Window

Since any direct access to logical addresses 192 to 207 involves the register pointers, the physical registers 192 to 207 can be accessed only when selected by a register pointer. After a reset, RP0 points to R192 and RP1 points to R200.

Register List

Table 1 lists the Super8 registers. For more details, see Figure 8.

Table 1. Super-8 Registers

Address				
Decimal	**Hexadecimal**		**Mnemonic**	**Function**
General-Purpose Registers				
000-192	00-BF		—	General purpose (all address modes)
192-207	C0-CF		—	Working register (direct only)
192-255	C0-FF		—	General purpose (indirect only)
Mode and Control Registers				
208	D0		P0	Port 0 I/O bits
209	D1		P1	Port 1 (I/O only)
210	D2		P2	Port 2
211	D3		P3	Port 3
212	D4		P4	Port 4
213	D5		FLAGS	System Flags Register
214	D6		RP0	Register Pointer 0
215	D7		RP1	Register Pointer 1
216	D8		SPH	Stack Pointer High Byte
217	D9		SPL	Stack Pointer Low Byte
218	DA		IPH	Instruction Pointer High Byte
219	DB		IPL	Instruction Pointer Low Byte
220	DC		IRQ	Interrupt Request
221	DD		IMR	Interrupt Mask Register
222	DE		SYM	System Mode
224	E0	Bank 0	C0CT	CTR 0 Control
		Bank 1	C0M	CTR 0 Mode
225	E1	Bank 0	C1CT	CTR 1 Control
		Bank 1	C1M	CTR 1 Mode
226	E2	Bank 0	C0CH	CTR 0 Capture Register, bits 8-15
		Bank 1	CTCH	CTR 0 Timer Constant, bits 8-15
227	E3	Bank 0	C0CL	CTR 0 Capture Register, bits 0-7
		Bank 1	CTCL	CTR 0 Time Constant, bits 0-7
228	E4	Bank 0	C1CH	CTR 1 Capture Register, bits 8-15
		Bank 1	C1TCH	CTR 1 Time Constant, bits 8-15
229	E5	Bank 0	C1CL	CTR 1 Capture Register, bits 0-7
		Bank 1	C1TCL	CTR 1 Time Constant, bits 0-7
235	EB	Bank 0	UTC	UART Transmit Control
236	EC	Bank 0	URC	UART Receive Control
237	ED	Bank 0	UIE	UART Interrupt Enable
239	EF	Bank 0	UIO	UART Data
240	F0	Bank 0	P0M	Port 0 Mode
		Bank 1	DCH	DMA Count, bits 8-15
241	F1	Bank 0	PM	Port Mode Register
		Bank 1	DCL	DMA Count, bits 0-7
244	F4	Bank 0	H0C	Handshake Channel 0 Control
245	F5	Bank 0	H1C	Handshake Channel 1 Control
246	F6	Bank 0	P4D	Port 4 Direction
247	F7	Bank 0	P4OD	Port 4 Open Drain
248	F8	Bank 0	P2AM	Port 2/3 A Mode
		Bank 1	UBGH	UART Baud Rate Generator, bits 8-15

Table 1. Super-8 Registers (Continued)

Address				
Decimal	**Hexadecimal**		**Mnemonic**	**Function**
Mode and Control Registers (Continued)				
249	F9	Bank 0	P2BM	Port 2/3 B Mode
		Bank 1	UBGL	UART Baud Rate Generator, bits 0-7
250	FA	Bank 0	P2CM	Port 2/3 C Mode
		Bank 1	UMA	UART Mode A
251	FB	Bank 0	P2DM	Port 2/3 D Mode
		Bank 1	UMB	UART Mode B
252	FC	Bank 0	P2AIP	Port 2/3 A Interrupt Pending
253	FD	Bank 0	P2BIP	Port 2/3 B Interrupt Pending
254	FE	Bank 0	EMT	External Memory Timing
		Bank 1	WUMCH	Wakeup Match Register
255	FF	Bank 0	IPR	Interrupt Priority Register
		Bank 1	WUMSK	Wakeup Mask Register

MODE AND CONTROL REGISTERS

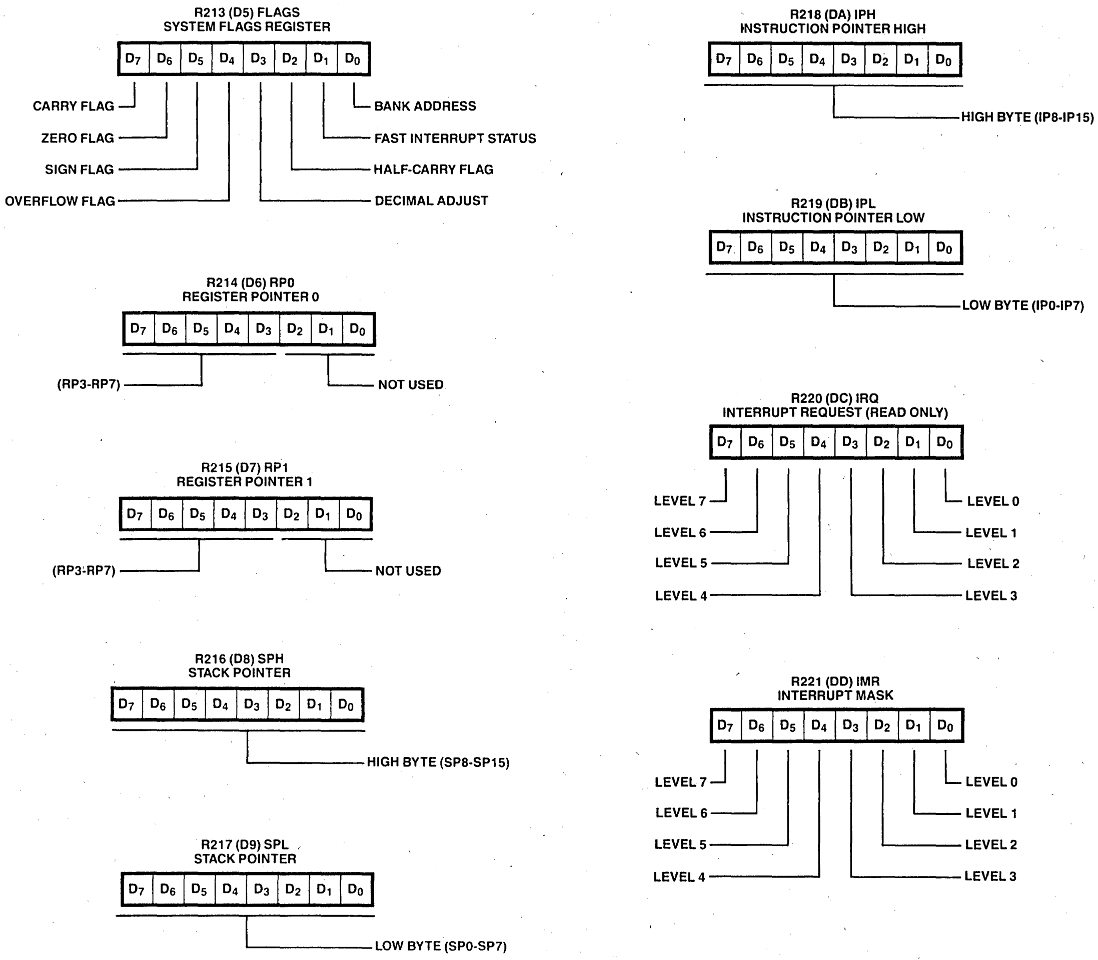
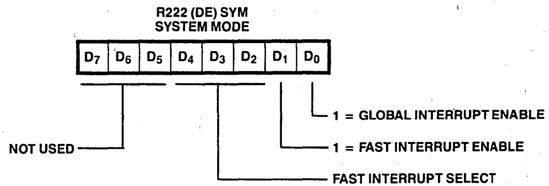
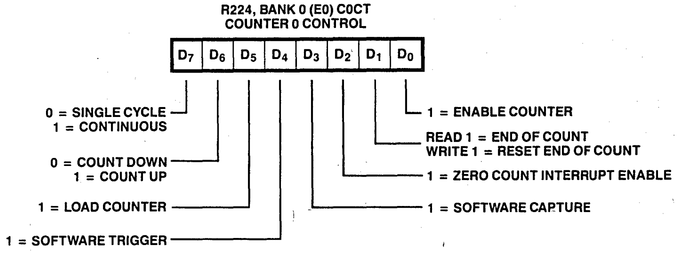


Figure 8. Mode and Control Registers

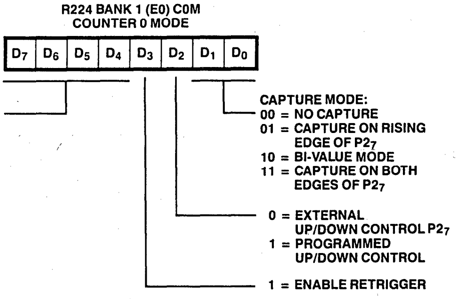


FAST INTERRUPT SELECT	
000	LEVEL 0
001	LEVEL 1
010	LEVEL 2
011	LEVEL 3
100	LEVEL 4
101	LEVEL 5
110	LEVEL 6
111	LEVEL 7



INPUT PIN ASSIGNMENTS:

D_7	D_6	D_5	D_4	$P2_7$	$P2_6$
0	0	0	0	I/O	I/O
0	0	0	1	I/O	TRIGGER
0	0	1	0	GATE	I/O
0	0	1	1	GATE	TRIGGER
0	1	0	0	I/O	C0 INPUT
0	1	0	1	TRIGGER	C0 INPUT
0	1	1	0	GATE	C0 INPUT
0	1	1	1	GATE/ TRIGGER	C0 INPUT
1	0	0	0	C0 OUTPUT	I/O
1	0	0	1	C0 OUTPUT	TRIGGER
1	0	1	0	C0 OUTPUT	GATE
1	0	1	1	C0 OUTPUT	GATE/TRIGGER
1	1	0	0	C0 OUTPUT	C0 INPUT
1	1	0	1	UNDEFINED	
1	1	1	0	UNDEFINED	
1	1	1	1	CASCADE COUNTERS	


R224 BANK 1 (E0) C0M
COUNTER 0 MODE
D7 D6 D5 D4 D3 D2 D1 D0
CAPTURE MODE:
00 = NO CAPTURE
01 = CAPTURE ON RISING EDGE OF P27
10 = BI-VALUE MODE
11 = CAPTURE ON BOTH EDGES OF P27
0 = EXTERNAL UP/DOWN CONTROL P27
1 = PROGRAMMED UP/DOWN CONTROL
1 = ENABLE RETRIGGER

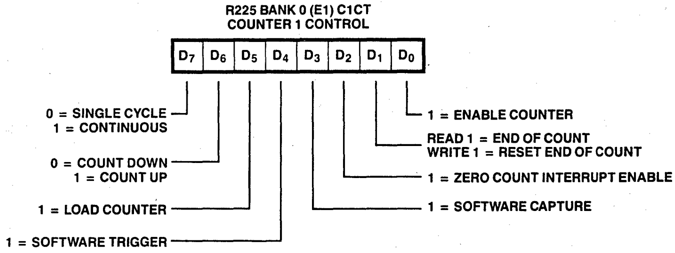


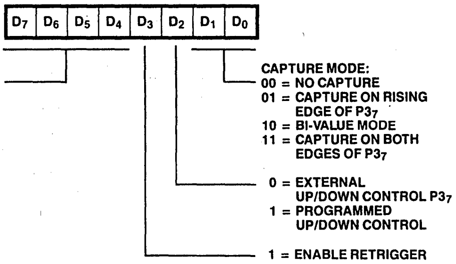
Figure 8. Mode and Control Registers (Continued)

MODE AND CONTROL REGISTERS (Continued)

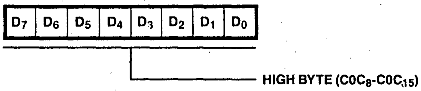
R225 BANK 1 (E1) C1M
COUNTER 1 MODE

INPUT PIN ASSIGNMENTS:

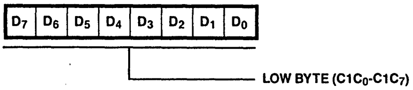
D_7	D_6	D_5	D_4	$P3_7$	$P3_6$
0	0	0	0	I/O	I/O
0	0	0	1	I/O	TRIGGER
0	0	1	0	GATE	I/O
0	0	1	1	GATE	TRIGGER
0	1	0	0	I/O	C0 INPUT
0	1	0	1	TRIGGER	C0 INPUT
0	1	1	0	GATE	C0 INPUT
0	1	1	1	GATE/ TRIGGER	C0 INPUT
1	0	0	0	C0 OUTPUT	I/O
1	0	0	1	C0 OUTPUT	TRIGGER
1	0	1	0	C0 OUTPUT	GATE
1	0	1	1	C0 OUTPUT	GATE/TRIGGER
1	1	0	0	C0 OUTPUT	C0 INPUT
1	1	0	1	UNDEFINED	
1	1	1	0	UNDEFINED	
1	1	1	1	UNDEFINED	


D7 D6 D5 D4 D3 D2 D1 D0
CAPTURE MODE:
00 = NO CAPTURE
01 = CAPTURE ON RISING EDGE OF P37
10 = BI-VALUE MODE
11 = CAPTURE ON BOTH EDGES OF P37
0 = EXTERNAL UP/DOWN CONTROL P37
1 = PROGRAMMED UP/DOWN CONTROL
1 = ENABLE RETRIGGER

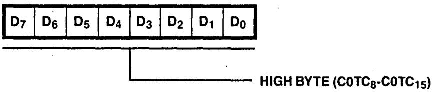
R226 BANK 0 (E2) C0CH
COUNTER 0 CAPTURE



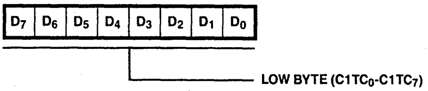
R229 BANK 0 (E5) C1CL
COUNTER 1 CAPTURE



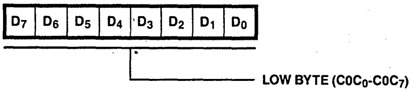
R226 BANK 1 (E2) C0TCH
COUNTER 0 TIME CONSTANT



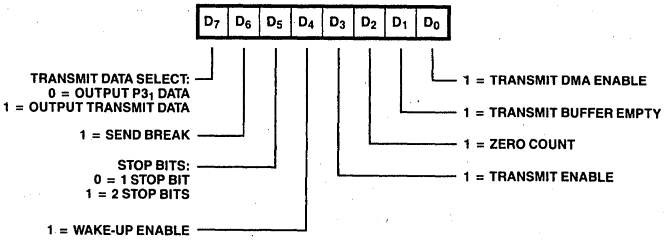
R229 BANK 1 (E5) C1TCL
COUNTER 1 TIME CONSTANT



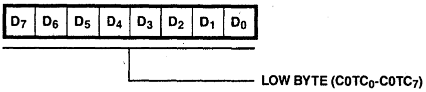
R227 BANK 0 (E3) C0CL
COUNTER 0 CAPTURE



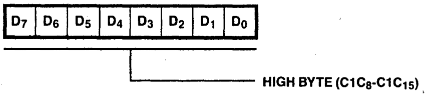
R235 BANK 0 (EB) UTC
UART TRANSMIT CONTROL



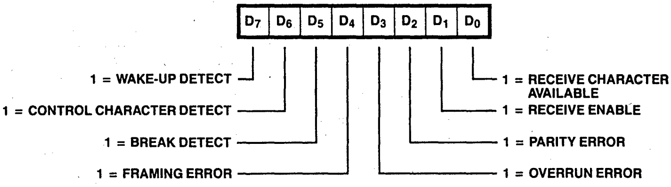
R227 BANK 1 (E3) C0TCL
COUNTER 0 TIME CONSTANT



R228 BANK 0 (E4) C1CH
COUNTER 1 CAPTURE



R236 BANK 0 (EC) URC
UART RECEIVE CONTROL



R228 BANK 1 (E4) C1TCH
COUNTER 1 TIME CONSTANT

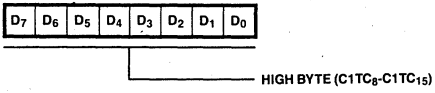


Figure 8. Mode and Control Registers (Continued)

MODE AND CONTROL REGISTERS (Continued)

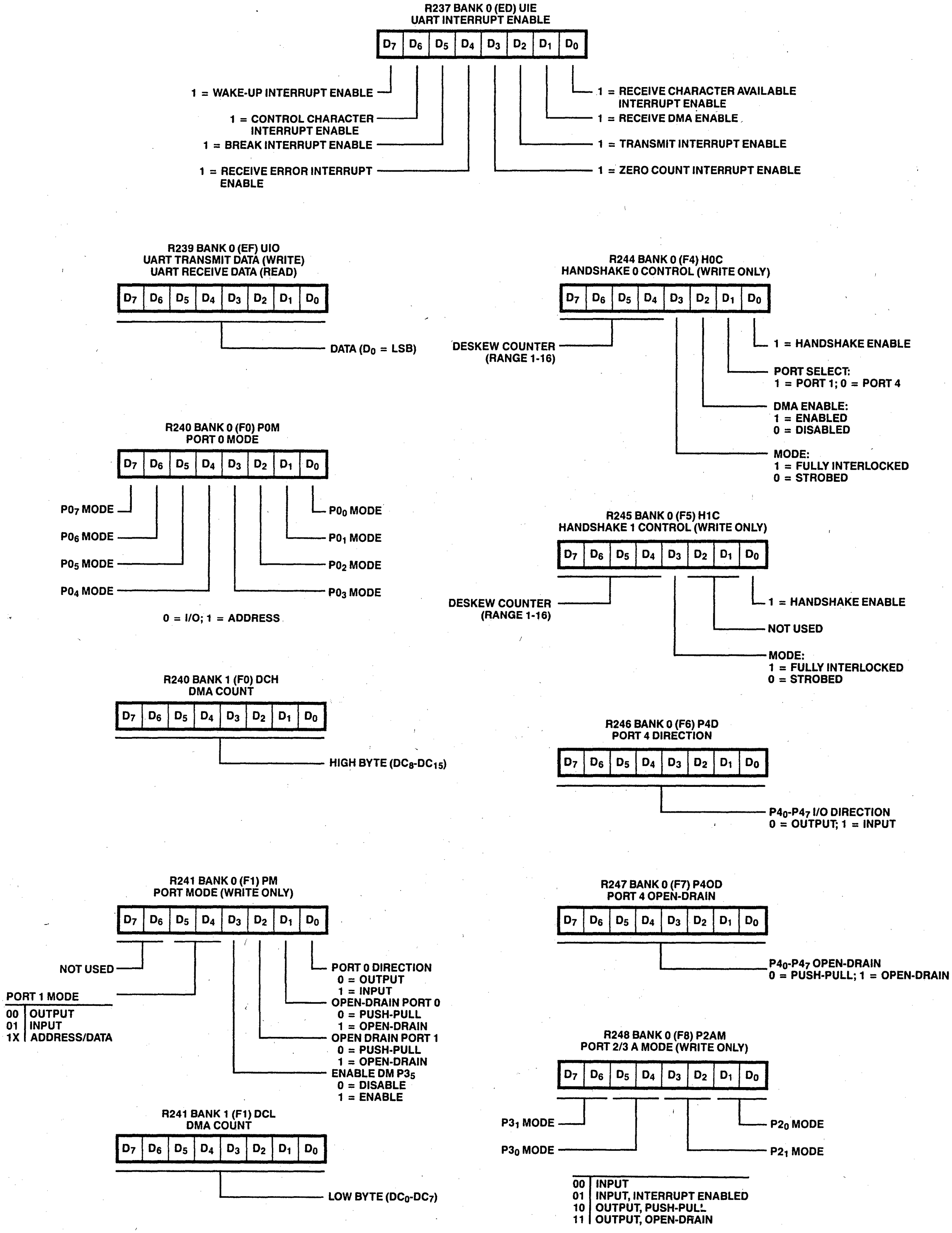


Figure 8. Mode and Control Registers (Continued)

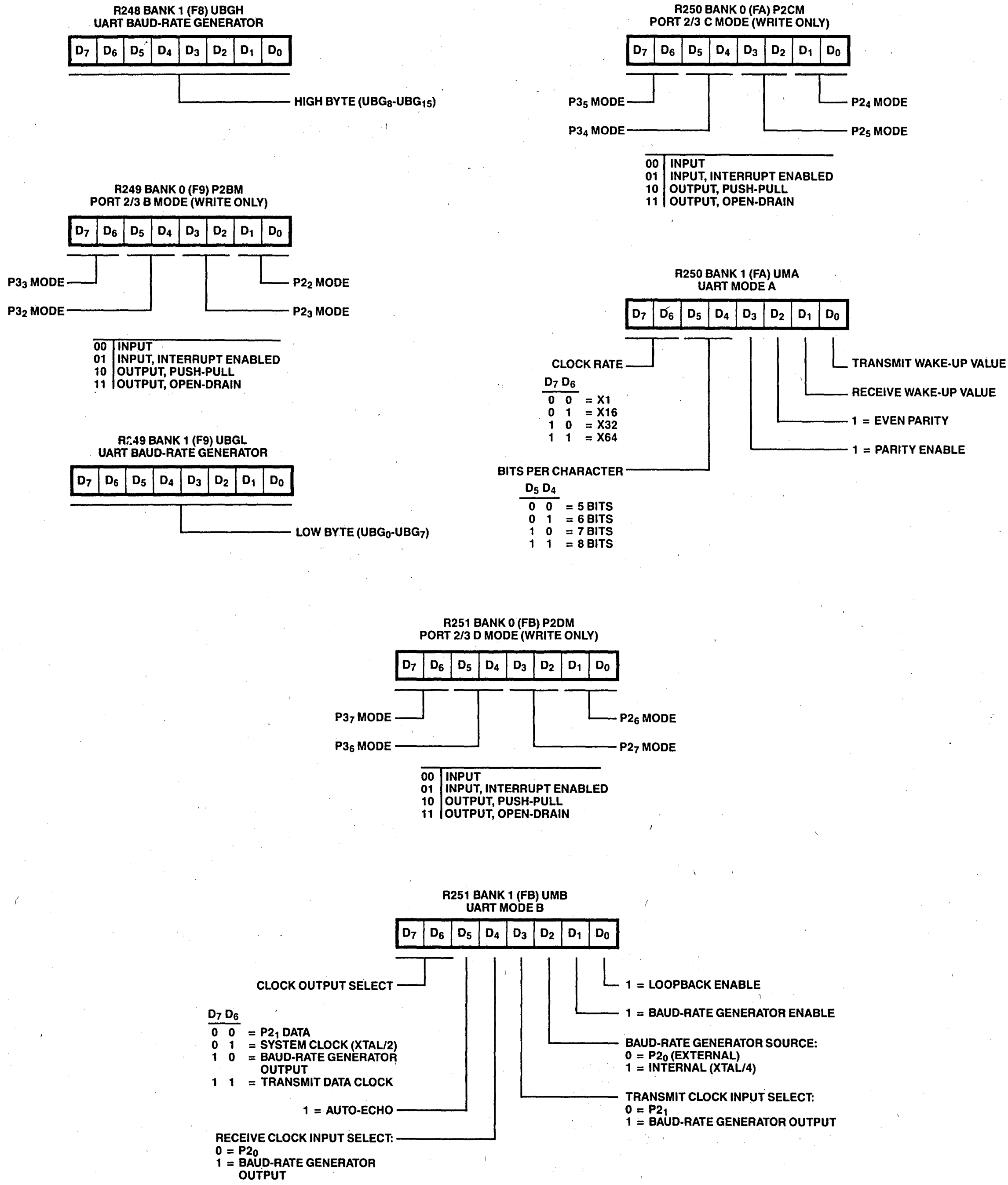
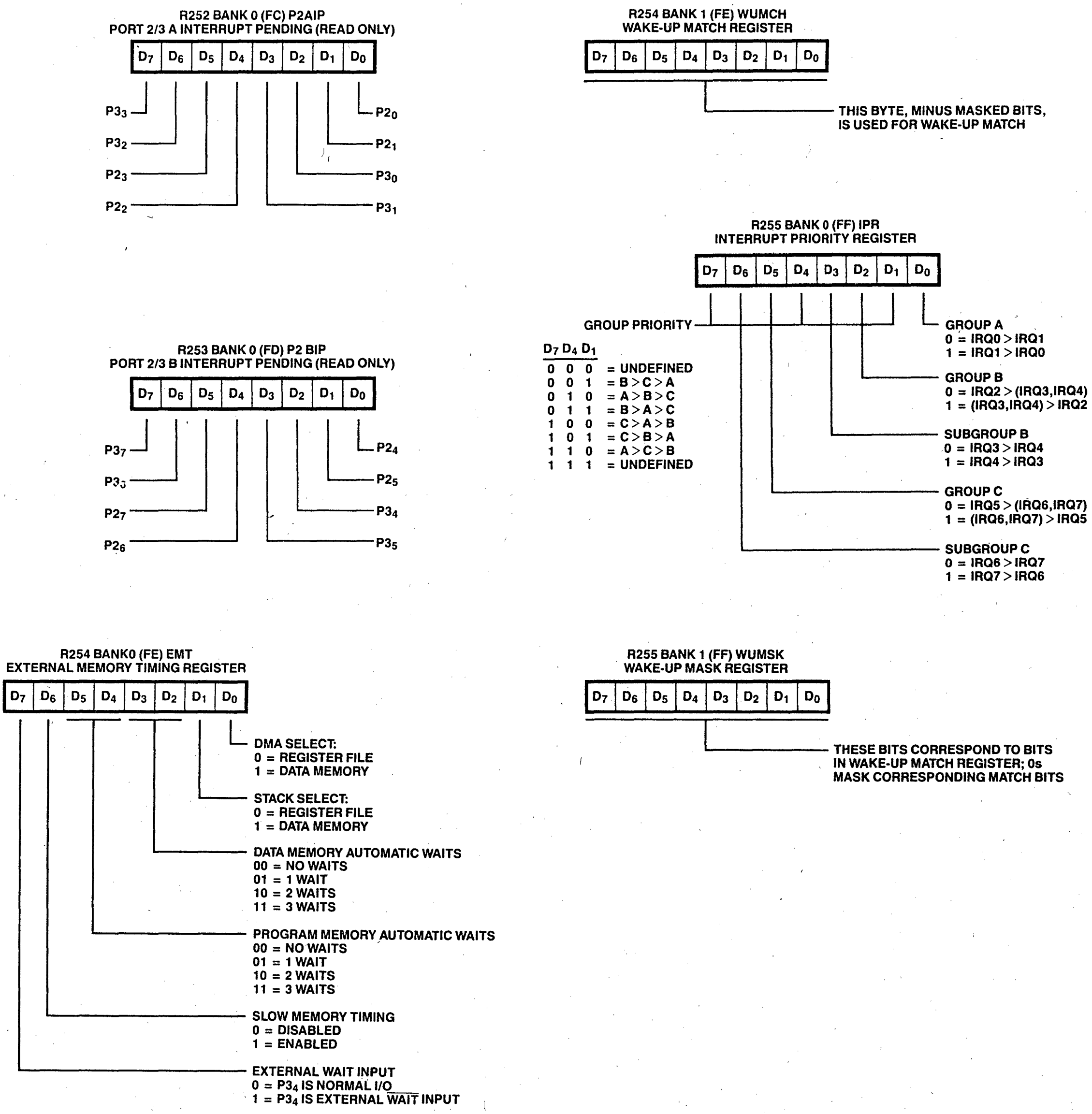


Figure 8. Mode and Control Registers (Continued)

MODE AND CONTROL REGISTERS (Continued)



R252 BANK 0 (FC) P2AIP
PORT 2/3 A INTERRUPT PENDING (READ ONLY)

R253 BANK 0 (FD) P2 BIP
PORT 2/3 B INTERRUPT PENDING (READ ONLY)

R254 BANK 1 (FE) WUMCH
WAKE-UP MATCH REGISTER

THIS BYTE, MINUS MASKED BITS, IS USED FOR WAKE-UP MATCH

R255 BANK 0 (FF) IPR
INTERRUPT PRIORITY REGISTER

D_7	D_4	D_1	
0	0	0	= UNDEFINED
0	0	1	= B>C>A
0	1	0	= A>B>C
0	1	1	= B>A>C
1	0	0	= C>A>B
1	0	1	= C>B>A
1	1	0	= A>C>B
1	1	1	= UNDEFINED

GROUP A
0 = IRQ0 > IRQ1
1 = IRQ1 > IRQ0

GROUP B
0 = IRQ2 > (IRQ3,IRQ4)
1 = (IRQ3,IRQ4) > IRQ2

SUBGROUP B
0 = IRQ3 > IRQ4
1 = IRQ4 > IRQ3

GROUP C
0 = IRQ5 > (IRQ6,IRQ7)
1 = (IRQ6,IRQ7) > IRQ5

SUBGROUP C
0 = IRQ6 > IRQ7
1 = IRQ7 > IRQ6

R254 BANKO (FE) EMT
EXTERNAL MEMORY TIMING REGISTER

DMA SELECT:
0 = REGISTER FILE
1 = DATA MEMORY

STACK SELECT:
0 = REGISTER FILE
1 = DATA MEMORY

DATA MEMORY AUTOMATIC WAITS
00 = NO WAITS
01 = 1 WAIT
10 = 2 WAITS
11 = 3 WAITS

PROGRAM MEMORY AUTOMATIC WAITS
00 = NO WAITS
01 = 1 WAIT
10 = 2 WAITS
11 = 3 WAITS

SLOW MEMORY TIMING
0 = DISABLED
1 = ENABLED

EXTERNAL WAIT INPUT
0 = $P3_4$ IS NORMAL I/O
1 = $P3_4$ IS EXTERNAL $\overline{\text{WAIT}}$ INPUT

R255 BANK 1 (FF) WUMSK
WAKE-UP MASK REGISTER

THESE BITS CORRESPOND TO BITS IN WAKE-UP MATCH REGISTER; 0s MASK CORRESPONDING MATCH BITS

Figure 8. Mode and Control Registers (Continued)

I/O PORTS

The Super8 has 40 I/O lines arranged into five 8-bit ports. These lines are all TTL-compatible, and can be configured as inputs or outputs. Some can also be configured as address/data lines.

Each port has an input register, an output register, and a register address. Data coming into the port is stored in the input register, and data to be written to a port is stored in the output register. Reading a port's register address returns the value in the input register; writing a port's register address loads the value in the output register. If the port is configured for an output, this value will appear on the external pins.

When the CPU reads the bits configured as outputs, the data on the external pins is returned. Under normal output loading, this has the same effect as reading the output register, unless the bits are configured as open-drain outputs.

The ports can be configured as shown in Table 2.

Table 2. Port Configuration

Port	Configuration Choices
0	Address outputs and/or general I/O
1	Multiplexed address/data(or I/O, only for ROM and Protopack)
2 and 3	Control I/O for UART, handshake channels, and counter/timers; also general I/O and external interrupts
4	General I/O

Port 0

Port 0 can be configured as an I/O port or an output for addressing external memory, or it can be divided and used as both. The bits configured as I/O can be either all outputs or all inputs; they cannot be mixed. If configured for outputs, they can be push-pull or open-drain type.

Any bits configured for I/O can be accessed via R208. To write to the port, specify R208 as the destination (dst) of an instruction; to read the port, specify R208 as the source (src).

Port 0 bits configured as I/O can be placed under handshake control of handshake channel 1.

Port 0 bits configured as address outputs cannot be accessed via the register.

In ROMless devices, initially the four lower bits are configured as address eight through twelve.

Port 1

In the ROMless device, Port 1 is configured as a byte-wide address/data port. It provides a byte-wide multiplexed address/data path. Additional address lines can be added by configuring Port 0.

The ROM and Protopack Port 1 can be configured as above or as an I/O port; it can be a byte-wide input, open-drain output, or push-pull output. It can be placed under handshake control or handshake channel 0.

Ports 2 and 3

Ports 2 and 3 provide external control inputs and outputs for the UART, handshake channels, and counter/timers. The pin assignments appear in Table 3.

Bits not used for control I/O can be configured as general-purpose I/O lines and/or external interrupt inputs.

Those bits configured for general I/O can be configured individually for input or output. Those configured for output can be individually configured for open-drain or push-pull output.

All Port 2 and 3 input pins are Schmitt-triggered.

The port address for Port 2 is R210, and for Port 3 is R211.

Table 3. Pin Assignments for Ports 2 and 3

Port 2		Port 3	
Bit	Function	Bit	Function
0	UART receive clock	0	UART receive data
1	UART transmit clock	1	UART transmit data
2	Reserved	2	Reserved
3	Reserved	3	Reserved
4	Handshake 0 input	4	Handshake 1 input/$\overline{\text{WAIT}}$
5	Handshake 0 output	5	Handshake 1 output/$\overline{\text{DM}}$
6	Counter 0 input	6	Counter 1 input
7	Counter 0 I/O	7	Counter 1 I/O

Port 4

Port 4 can be configured as I/O only. Each bit can be configured individually as input or output, with either push-pull or open-drain outputs. All Port 4 inputs are Schmitt-triggered.

Port 4 can be placed under handshake control of handshake channel 0. Its register address is R212.

UART

The UART is a full-duplex asynchronous channel. It transmits and receives independently with 5 to 8 bits per character, has options for even or odd bit parity, and a wake-up feature.

Data can be read into or out of the UART via R239, Bank 0. This single address is able to serve a full-duplex channel because it contains two complete 8-bit registers—one for the transmitter and the other for the receiver.

Pins

The UART uses the following Port 2 and 3 pins:

Port/Pin	UART Function
2/0	Receive Clock
3/0	Receive Data
2/1	Transmit Clock
3/1	Transmit Data

Transmitter

When the UART's register address is specified as the destination (dst) of an operation, the data is output on the UART, which automatically adds the start bit, the programmed parity bit, and the programmed number of stop bits. It can also add a wake-up bit if that option is selected.

If the UART is programmed for a 5-, 6-, or 7-bit character, the extra bits in R239 are ignored.

Serial data is transmitted at a rate equal to 1, 1/16, 1/32 or 1/64 of the transmitter clock rate, depending on the programmed data rate. All data is sent out on the falling edge of the clock input.

When the UART has no data to send, it holds the output marking (High). It may be programmed with the Send Break command to hold the output Low (Spacing), which it continues until the command is cleared.

Receiver

The UART begins receive operation when Receive Enable (URC, bit 0) is set High. After this, a Low on the receive input pin for longer than half a bit time is interpreted as a start bit. The UART samples the data on the input pin in the middle of each clock cycle until a complete byte is assembled. This is placed in the Receive Data register.

If the 1X clock mode is selected, external bit synchronization must be provided, and the input data is sampled on the rising edge of the clock.

For character lengths of less than eight bits, the UART inserts ones into the unused bits, and, if parity is enabled, the parity bit is not stripped. The data bits, extra ones, and the parity bit are placed in the UART Data register (UIO).

While the UART is assembling a byte in its input shift register, the CPU has time to service an interrupt and manipulate the data character in UIO.

Once a complete character is assembled, the UART checks it and performs the following:

- If it is an ASCII control character, the UART sets the Control Character status bit.
- It checks the wake-up settings and completes any indicated action.
- If parity is enabled, the UART checks to see if the calculated parity matches the programmed parity bit. If they do not match, it sets the Parity Error bit in URC (R236 Bank 0), which remains set until reset by software.
- It sets the Framing Error bit (URC, bit 4) if the character is assembled without any stop bits. This bit remains set until cleared by software.

Overrun errors occur when characters are received faster than they are read. That is, when the UART has assembled a complete character before the CPU has read the current character, the UART sets the Overrun Error bit (URC, bit 3), and the character currently in the receive buffer is lost.

The overrun bit remains set until cleared by software.

ADDRESS SPACE

The Super8 can access 64K bytes of program memory and 64K bytes of data memory. These spaces can be either combined or separate. If separate, they are controlled by the $\overline{DM}$ line (Port $P3_5$), which selects data memory when Low and program memory when High.

Figure 9 shows the system memory space.

CPU Program Memory

Program memory occupies addresses 0 to 64K. External program memory, if present, is accessed by configuring Ports 0 and 1 as a memory interface.

The address/data lines are controlled by $\overline{AS}$, $\overline{DS}$ and R/$\overline{W}$.

The first 32 program memory bytes are reserved for interrupt vectors; the lowest address available for user programs is 32 (decimal). This value is automatically loaded into the program counter after a hardware reset.

ROMless

Port 0 can be configured to provide from 0 to 8 additional address lines. Port 1 is always used as an 8-bit multiplexed address/data port.

ROM and Protopack

Port 1 is configured as multiplexed address/data or as I/O. When Port 1 is configured as address/data, Port 0 lines can be used as additional address lines, up to address 15. External program memory is mapped above internal program memory; that is, external program memory can occupy any space beginning at the top of the internal ROM space up to the 64K (16-bit address) limit.

CPU Data Memory

The external CPU data memory space, if separated from program memory by the $\overline{DM}$ optional output, can be mapped anywhere from 0 to 64K (full 16-bit address space). Data memory uses the same address/data bus (Port 1) and additional addresses (chosen from Port 0) as program memory. Data memory is distinguished from program memory by the DM pin ($P3_5$), and by the fact that data memory can begin at address 0000_H. This feature differs from the Z8.

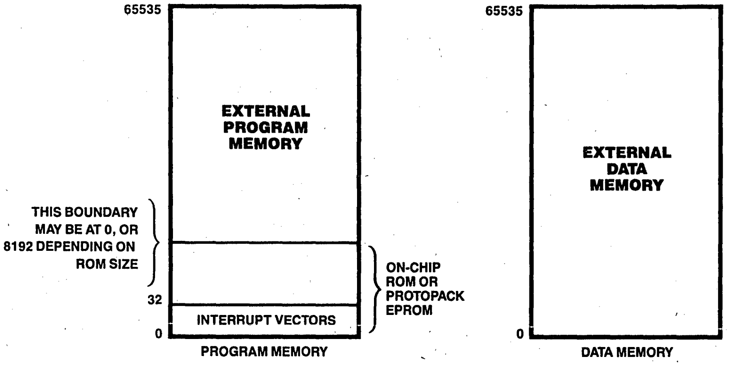


Figure 9. Program and Data Memory Address Spaces

INSTRUCTION SET

The Super8 instruction set is designed to handle its large register set. The instruction set provides a full complement of 8-bit arithmetic and logical operations, including multiply and divide. It supports BCD operations using a decimal adjustment of binary values, and it supports incrementing and decrementing 16-bit quantities for addresses and counters.

It provides extensive bit manipulation, and rotate and shift operations, and it requires no special I/O instructions—the I/O ports are mapped into the register file.

Instruction Pointer

A special register called the Instruction Pointer (IP) provides hardware support for threaded-code languages. It consists of register-pair R218 and R219, and it contains memory addresses. The MSB is R218.

Threaded-code languages deal with an imaginary higher-level machine within the existing hardware machine. The IP acts like the PC for that machine. The command NEXT passes control to or from the hardware machine to the imaginary machine, and the commands ENTER and EXIT are imaginary machine equivalents of (real machine) CALLS and RETURNS.

If the commands NEXT, ENTER, and EXIT are not used, the IP can be used by the fast interrupt processing, as described in the Interrupts section.

Flag Register

The Flag register (FLAGS) contains eight bits that describe the current status of the Super8. Four of these can be tested and used with conditional jump instructions; two others are used for BCD arithmetic. FLAGS also contains the Bank Address bit and the Fast Interrupt Status bit.

The flag bits can be set and reset by instructions.

CAUTION

Do not specify FLAGS as the destination of an instruction that normally affects the flag bits or the result will be unspecified.

The following paragraphs describe each flag bit:

Bank Address. This bit is used to select one of the register banks (0 or 1) between (decimal) addresses 224 and 255. It is cleared by the SB0 instruction and set by the SB1 instruction.

Fast Interrupt Status. This bit is set during a fast interrupt cycle and reset during the IRET following interrupt servicing. When set, this bit inhibits all interrupts and causes the fast interrupt return to be executed when the IRET instruction is fetched.

Half-Carry. This bit is set to 1 whenever an addition generates a carry out of bit 3, or when a subtraction borrows out of bit 4. This bit is used by the Decimal Adjust (DA) instruction to convert the binary result of a previous addition or subtraction into the correct decimal (BCD) result. This flag, and the Decimal Adjust flag, are not usually accessed by users.

Decimal Adjust. This bit is used to specify what type of instruction was executed last during BCD operations, so a subsequent Decimal Adjust operation can function correctly. This bit is not usually accessible to programmers, and cannot be used as a test condition.

Overflow Flag. This flag is set to 1 when the result of a twos-complement operation was greater than 127 or less than -128. It is also cleared to 0 during logical operations.

Sign Flag. Following arithmetic, logical, rotate, or shift operations, this bit identifies the state of the MSB of the result. A 0 indicates a positive number and a 1 indicates a negative number.

Zero Flag. For arithmetic and logical operations, this flag is set to 1 if the result of the operation is zero.

For operations that test bits in a register, the zero bit is set to 1 if the result is zero.

For rotate and shift operations, this bit is set to 1 if the result is zero.

Carry Flag. This flag is set to 1 if the result from an arithmetic operation generates a carry out of, or a borrow into, bit 7.

After rotate and shift operations, it contains the last value shifted out of the specified register.

It can be set, cleared, or complemented by instructions.

Condition Codes

The flags C, Z, S, and V are used to control the operation of conditional jump instructions.

The opcode of a conditional jump contains a 4-bit field called the condition code (cc). This specifies under which conditions it is to execute the jump. For example, a conditional jump with the condition code for "equal" after a compare operation only jumps if the two operands are equal.

The condition codes and their meanings are given in Table 4.

Addressing Modes

All operands except for immediate data and condition codes are expressed as register addresses, program memory addresses, or data memory addresses. The addressing modes and their designations are:

Register (R)
Indirect Register (IR)
Indexed (X)
Direct (DA)
Relative (RA)
Immediate (IM)
Indirect (IA)

Table 4. Condition Codes and Meanings

Binary	Mnemonic	Flags	Meaning
0000	F	—	Always false
1000	—	—	Always true
0111*	C	C=1	Carry
1111*	NC	C=0	No carry
0110*	Z	Z=1	Zero
1110*	NZ	Z=0	Not zero
1101	PL	S=0	Plus
0101	MI	S=1	Minus
0100	OV	V=1	Overflow
1100	NOV	V=0	No overflow
0110*	EQ	Z=1	Equal
1110*	NE	Z=0	Not equal
1001	GE	(S XOR V)=0	Greater than or equal
0001	LT	(S XOR V)=1	Less than
1010	GT	(Z OR (S XOR V))=0	Greater than
0010	LE	(Z OR (S XOR V))=1	Less than or equal
1111*	UGE	C=0	Unsigned greater than or equal
0111*	ULT	C=1	Unsigned less than
1011	UGT	(C=0 AND Z=0)=1	Unsigned greater than
0011	ULE	(C OR Z)=1	Unsigned less than or equal

NOTE: Asterisks (*) indicate condition codes that relate to two different mnemonics but test the same flags. For example, Z and EQ are both True if the Zero flag is set, but after an ADD instruction, Z would probably be used, while after a CP instruction, EQ would probably be used.

Registers can be addressed by an 8-bit address in the range of 0 to 255. Working registers can also be addressed using 4-bit addresses, where five bits contained in a register pointer (R218 or R219) are concatenated with three bits from the 4-bit address to form an 8-bit address.

Registers can be used in pairs to generate 16-bit program or data memory addresses.

Notation and Encoding

The instruction set notations are described in Table 5.

Functional Summary of Commands

Figure 10 shows the formats followed by a quick reference guide to the commands.

Table 5. Instruction Set Notations

Notation	Meaning	Notation	Meaning
cc	Condition code (see Table 4)	DA	Direct address (between 0 and 65535)
r	Working register (between 0 and 15)	RA	Relative address
rb	Bit of working register	IM	Immediate
r0	Bit 0 of working register	IML	Immediate long
R	Register or working register	dst	Destination operand
RR	Register pair or working register pair (Register pairs always start on an even-number boundary)	src	Source operand
		@	Indirect address prefix
IA	Indirect address	SP	Stack pointer
Ir	Indirect working register	PC	Program counter
IR	Indirect register or indirect working register	IP	Instruction pointer
Irr	Indirect working register pair	FLAGS	Flags register
IRR	Indirect register pair or indirect working register pair	RP	Register pointer
X	Indexed	#	Immediate operand prefix
XS	Indexed, short offset	%	Hexadecimal number prefix
XL	Indexed, long offset	OPC	Opcode

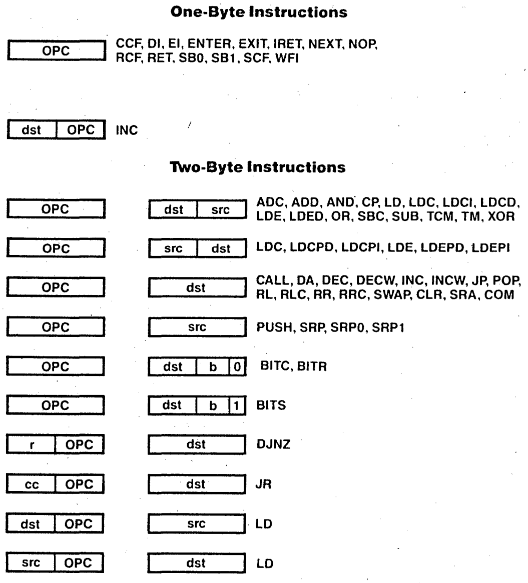

One-Byte Instructions
OPC
CCF, DI, EI, ENTER, EXIT, IRET, NEXT, NOP, RCF, RET, SB0, SB1, SCF, WFI
dst OPC
INC
Two-Byte Instructions
OPC
dst src
ADC, ADD, AND, CP, LD, LDC, LDCI, LDCD, LDE, LDED, OR, SBC, SUB, TCM, TM, XOR
OPC
src dst
LDC, LDCPD, LDCPI, LDE, LDEPD, LDEPI
OPC
dst
CALL, DA, DEC, DECW, INC, INCW, JP, POP, RL, RLC, RR, RRC, SWAP, CLR, SRA, COM
OPC
src
PUSH, SRP, SRP0, SRP1
OPC
dst b 0
BITC, BITR
OPC
dst b 1
BITS
r OPC
dst
DJNZ
cc OPC
dst
JR
dst OPC
src
LD
src OPC
dst
LD

Figure 10. Instruction Formats

Three-Byte Instructions

Four-Byte Instructions

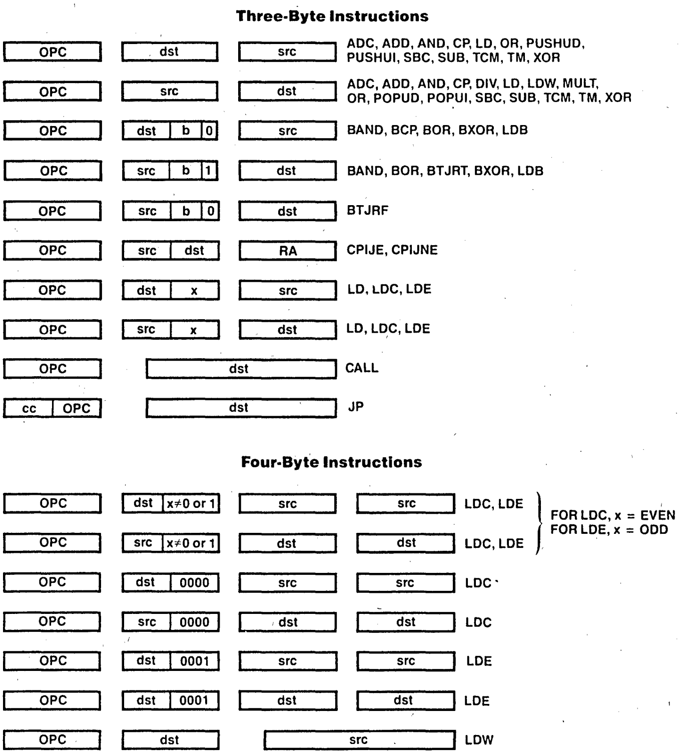

OPC | dst | src — ADC, ADD, AND, CP, LD, OR, PUSHUD, PUSHUI, SBC, SUB, TCM, TM, XOR
OPC | src | dst — ADC, ADD, AND, CP, DIV, LD, LDW, MULT, OR, POPUD, POPUI, SBC, SUB, TCM, TM, XOR
OPC | dst b 0 | src — BAND, BCP, BOR, BXOR, LDB
OPC | src b 1 | dst — BAND, BOR, BTJRT, BXOR, LDB
OPC | src b 0 | dst — BTJRF
OPC | src dst | RA — CPIJE, CPIJNE
OPC | dst x | src — LD, LDC, LDE
OPC | src x | dst — LD, LDC, LDE
OPC | dst — CALL
cc OPC | dst — JP
OPC | dst x≠0 or 1 | src | src — LDC, LDE
OPC | src x≠0 or 1 | dst | dst — LDC, LDE
FOR LDC, x = EVEN
FOR LDE, x = ODD
OPC | dst 0000 | src | src — LDC
OPC | src 0000 | dst | dst — LDC
OPC | dst 0001 | src | src — LDE
OPC | dst 0001 | dst | dst — LDE
OPC | dst | src — LDW

Figure 10. Instruction Formats (Continued)

INSTRUCTION SUMMARY

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
ADC dst,src dst ← dst + src + C	(Note 1)		1□	*	*	*	—	0	*
ADD dst,src dst ← dst + src	(Note 1)		0□	*	*	*	*	0	*
AND dst,src dst ← dst AND src	(Note 1)		5□	—	*	*	0	—	—
BAND dst,src dst ← dst AND src	r0 Rb	Rb r0	67 67	—	*	0	U	—	—
BCP dst, src dst − src	r0	Rb	17	—	*	0	U	—	—
BITC dst dst ← NOT dst	rb		57	—	*	0	U	—	—
BITR dst dst ← 0	rb		77	—	—	—	—	—	—
BITS dst dst ← 1	rb		77	—	—	—	—	—	—

Instruction and Operation	Addr Mode dst	Addr Mode src	Opcode Byte (Hex)	C	Z	S	V	D	H
BOR dst, src dst ← dst OR src	r0 Rb	rB r0	07	—	*	0	U	—	—
BTJRF if src = 0, PC = PC + dst	RA	rb	37	—	—	—	—	—	—
BTJRT if src = 1, PC = PC + dst	RA	rb	37	—	—	—	—	—	—
BXOR dst, src dst ← dst XOR src	r0 Rb	Rb r0	27 27	—	*	0	U	—	—
CALL dst SP ← SP − 2 @SP ← PC PC ← dst	DA IRR IA		F6 F4 D4	—	—	—	—	—	—
CCF C = NOT C			EF	*	—	—	—	—	—
CLR dst dst ← 0	R IR		B0 B1	—	—	—	—	—	—

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
COM dst	R		60	—	*	*	0	—	—
dst ← NOT dst	IR		61						
CP dst,src	(Note 1)		A□	*	*	*	*	—	—
dst − src									
CPIJE	r	Ir	C2	—	—	—	—	—	—
if dst − src = 0,then									
PC ← PC + RA									
Ir ← Ir + 1									
CPIJNE	r	Ir	D2	—	—	—	—	—	—
if dst − src = 0,then									
PC ← PC + RA									
Ir ← Ir + 1									
DA dst	R		40	*	*	*	U	—	—
dst ← DA dst	IR		41						
DEC dst	R		00	—	*	*	*	—	—
dst ← dst − 1	IR		01						
DECW dst	RR		80	—	*	*	*	—	—
dst ← dst − 1	IR		81						
DI			8F	—	—	—	—	—	—
SMR (0) ← 0									
DIV dst, src									
dst ÷ src	RR	R	94	*	*	*	*	—	—
dst (Upper) ←	RR	IR	95						
Quotient									
dst (Lower) ←	RR	IM	96						
Remainder									
DJNZ r,dst	RA	r	rA	—	—	—	—	—	—
r ← r − 1			(r = 0 to F)						
if r = 0									
PC ← PC + dst									
EI			9F	—	—	—	—	—	—
SMR (0) ← 1									
ENTER			1F	—	—	—	—	—	—
SP ← SP − 2									
@ SP ← IP									
IP ← PC									
PC ← @ IP									
IP ← IP + 2									
EXIT			2F	—	—	—	—	—	—
IP ← @SP									
SP ← SP + 2									
PC ← @IP									
IP ← IP + 2									
INC dst	r		rE	—	*	*	*	—	—
dst ← dst + 1			(r = 0 to F)						
	R		20						
	IR		21						

Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
INCW dst	RR		A0	—	*	*	*	—	—
dst ← 1 + dst	IR		A1						
IRET (Fast)			BF	Restored to before interrupt					
PC ↔ IP									
FLAG ← FLAG′									
FIS ← 0									
IRET (Normal)			BF	Restored to before interrupt					
FLAGS ← @SP; SP ← SP + 1									
PC ← @SP; SP ← SP + 2; SMR (0) ← 1									
JP cc,dst	DA		ccD	—	—	—	—	—	—
if cc is true,			(cc = 0 to F)						
PC ← dst	IRR		30						
JR cc,dst	RA		ccB	—	—	—	—	—	—
if cc is true,			(cc = 0 to F)						
PC ← PC + d									
LD dst,src	r	IM	rC	—	—	—	—	—	—
dst ← src	r	R	r8						
	R	r	r9						
			(r = 0 to F)						
	r	IR	C7						
	IR	r	D7						
	R	R	E4						
	R	IR	E5						
	R	IM	E6						
	IR	IM	D6						
	IR	R	F5						
	r	x	87						
	x	r	97						
LDB dst, src	r0	Rb	47	—	—	—	—	—	—
dst ← src	Rb	r0	47						
LDC/LDE	r	Irr	C3	—	—	—	—	—	—
dst ← src	Irr	r	D3						
	r	xs	E7						
	xs	r	F7						
	r	x1	A7						
	x1	r	B7						
	r	DA	A7						
	DA	r	B7						
LDCD/LDED dst, src	r	Irr	E2	—	—	—	—	—	—
dst ← src									
rr ← rr − 1									
LDEI/LDCI dst, src	r	Irr	E3	—	—	—	—	—	—
dst ← src									
rr ← rr + 1									
LDCPD/LDEPD dst,src									
rr ← rr − 1	Irr	r	F2	—	—	—	—	—	—
dst ← src									

Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	**dst**	**src**	**(Hex)**	**C**	**Z**	**S**	**V**	**D**	**H**
LDCPI/LDEPI dst, src									
rr ← rr + 1	Irr	r	F3	—	—	—	—	—	—
dst ← src									
LDW dst, src	RR	RR	C4	—	—	—	—	—	—
dst ← src	RR	IR	C5						
	RR	IMM	C6						
MULT dst, src	RR	R	84	*	0	*	*	—	—
	RR	IR	85						
	RR	IM	86						
NEXT			0F	—	—	—	—	—	—
PC ← @IP									
IP ← IP + 2									
NOP			FF	—	—	—	—	—	—
OR dst,src	(Note 1)		4□	—	*	*	0	—	—
dst ← dst OR src									
POP dst		R	50	—	—	—	—	—	—
dst ← @SP;		IR	51						
SP ← SP + 1									
POPUD dst, src	R	IR	92	—	—	—	—	—	—
dst ← src									
IR ← IR − 1									
POPUI dst, src	R	IR	93	—	—	—	—	—	—
dst ← src									
IR ← IR + 1									
PUSH src		R	70	—	—	—	—	—	—
SP ← SP − 1; @SP ← src		IR	71						
PUSHUD dst, src	IR	R	82	—	—	—	—	—	—
IR ← IR − 1									
dst ← src									
PUSHUI dst, src	IR	R	83	—	—	—	—	—	—
IR ← IR + 1									
dst ← src									
RCF			CF	0	—	—	—	—	—
C ← 0									
RET			AF	—	—	—	—	—	—
PC ← @SP; SP ← SP + 2									
RL dst	R		90	*	*	*	*	—	—
C ← dst (7)	IR		91						
dst (0) ← dst (7)									
dst (N + 1) ← dst (N)									
N = 0 to 6									

Instruction	Addr Mode		Opcode Byte	Flags Affected					
and Operation	**dst**	**src**	**(Hex)**	**C**	**Z**	**S**	**V**	**D**	**H**
RLC dst	R		10	*	*	*	*	—	—
dst (0) ← C	IR		11						
C ← dst (7)									
dst (N + 1) ← dst (N)									
N = 0 to 6									
RR dst	R		E0	*	*	*	*	—	—
C ← dst (0)	IR		E1						
dst (7) ← dst (0)									
dst (N) ← dst (N + 1)									
N = 0 to 6									
RRC dst	R		C0	*	*	*	*	—	—
C ← dst (0)	IR		C1						
dst (7) ← C									
dst (N) ← dst (N + 1)									
N = 0 to 6									
SB0			4F	—	—	—	—	—	—
BANK ← 0									
SB1			5F	—	—	—	—	—	—
BANK ← 1									
SBC dst,src	(Note 1)		3□	*	*	*	*	1	*
dst ← dst − src − C									
SCF			DF	1	—	—	—	—	—
C ← 1									
SRA dst	R		D0	*	*	*	0	—	—
dst (7) ← dst (7)	IR		D1						
C ← dst (0)									
dst (N) ← dst (N + 1)									
N = 0 to 6									
SRP src		IM	31	—	—	—	—	—	—
RP0 ← IM									
RP1 ← IM + 8									
SRP0		IM	3l	—	—	—	—	—	—
RP0 ← IM									
SRP1		IM	3l	—	—	—	—	—	—
RP1 ← IM									
SUB dst,src	(Note 1)		2□	*	*	*	*	1	*
dst ← dst − src									

Instruction	Addr Mode		Opcode	Flags Affected					
			Byte						
and Operation	dst	src	(Hex)	C	Z	S	V	D	H
SWAP dst dst (0-3) ↔ dst (4-7)	R IR		F0 F1	—	*	*	U	—	—
TCM dst,src (NOT dst) AND src	(Note 1)		6□	—	*	*	0	—	—
TM dst,src dst AND src	(Note 1)		7□	—	*	*	0	—	—
WFI			3F	—	—	—	—	—	—
XOR dst,src dst ← dst XOR src	(Note 1)		B□	—	*	*	0	—	—

NOTE 1: These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble identifies the command, and is found in the table above. The second nibble, represented by a □, defines the addressing mode as shown in Table 6.:

Table 6. Second Nibble

Addr Mode		Lower
dst	src	Opcode Nibble
r	r	$\boxed{2}$
r	Ir	$\boxed{3}$
R	R	$\boxed{4}$
R	IR	$\boxed{5}$
R	IM	$\boxed{6}$

For example, to use an opcode represented as x□ with an "RR" addressing mode, use the opcode "x4."

0	= Cleared to Zero
1	= Set to One
—	= Unaffected
*	= Set or reset, depending on result of operation.
U	= Undefined

SUPER-8 OPCODE MAP

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6 **DEC** R_1	6 **DEC** IR_1	6 **ADD** r_1,r_2	6 **ADD** r_1,Ir_2	10 **ADD** R_2,R_1	10 **ADD** IR_2,R_1	10 **ADD** R_1,IM	10 **BOR*** r_0-R_b	6 **LD** r_1,R_2	6 **LD** r_2,R_1	12/10 **DJNZ** r_1,RA	12/10 **JR** cc,RA	6 **LD** r_1,IM	12/10 **JP** cc,DA	6 **INC** r1	14 **NEXT**
1	6 **RLC** R_1	6 **RLC** IR_1	6 **ADC** r_1,r_2	6 **ADC** r_1,Ir_2	10 **ADC** R_2,R_1	10 **ADC** IR_2,R_1	10 **ADC** R_1,IM	10 **BCP** r_1,b,R_2	↓	↓	↓	↓	↓	↓	↓	20 **ENTER**
2	6 **INC** R_1	6 **INC** IR_1	6 **SUB** r_1,r_2	6 **SUB** r_1,Ir_2	10 **SUB** R_2,R_1	10 **SUB** IR_2,R_1	10 **SUB** R_1,IM	10 **BXOR*** r_0-R_b	↓	↓	↓	↓	↓	↓	↓	22 **EXIT**
3	10 **JP** IRR_1	**NOTE** **C**	6 **SBC** r_1,r_2	6 **SBC** r_1,Ir_2	10 **SBC** R_2,R_1	10 **SBC** IR_2,R_1	10 **SBC** R_1,IM	**NOTE** **A**	↓	↓	↓	↓	↓	↓	↓	6 **WFI**
4	6 **DA** R_1	6 **DA** IR_1	6 **OR** r_1,r_2	6 **OR** r_1,Ir_2	10 **OR** R_2,R_1	10 **OR** IR_2,R_1	10 **OR** R_1,IM	10 **LDB*** r_0-R_b	↓	↓	↓	↓	↓	↓	↓	6 **SBO**
5	10 **POP** R_1	10 **POP** IR_1	6 **AND** r_1,r_2	6 **AND** r_1,Ir_2	10 **AND** R_2,R_1	10 **AND** IR_2,R_1	10 **AND** R_1,IM	8 **BITC** r_1,b	↓	↓	↓	↓	↓	↓	↓	6 **SBI**
6	6 **COM** R_1	6 **COM** IR_1	6 **TCM** r_1,r_2	6 **TCM** r_1,Ir_2	10 **TCM** R_2,R_1	10 **TCM** IR_2,R_1	10 **TCM** R_1,IM	10 **BAND*** r_0-R_b	↓	↓	↓	↓	↓	↓	↓	
7	10/12 **PUSH** R_2	12/14 **PUSH** IR_2	6 **TM** r_1,r_2	6 **TM** r_1,Ir_2	10 **TM** R_2,R_1	10 **TM** IR_2,R_1	10 **TM** R_1,IM	**NOTE** **B**	↓	↓	↓	↓	↓	↓	↓	
8	10 **DECW** RR_1	10 **DECW** IR_1	10 **PUSHUD** IR_1,R_2	10 **PUSHUI** IR_1,R_2	24 **MULT** R_2,RR_1	24 **MULT** IR_2,RR_1	24 **MULT** IM,RR_1	10 **LD** r_1,x,r_2	↓	↓	↓	↓	↓	↓	↓	6 **DI**
9	6 **RL** R_1	6 **RL** IR_1	10 **POPUD** IR_2,R_1	10 **POPUI** IR_2,R_1	28/12 **DIV** R_2,RR_1	28/12 **DIV** IR_2,RR_1	28/12 **DIV** IM,RR_1	10 **LD** r_2,x,r_1	↓	↓	↓	↓	↓	↓	↓	6 **EI**
A	10 **INCW** RR_1	10 **INCW** IR_1	6 **CP** r_1,r_2	6 **CP** r_1,Ir_2	10 **CP** R_2,R_1	10 **CP** IR_2,R_1	10 **CP** R_1,IM	**NOTE** **D**	↓	↓	↓	↓	↓	↓	↓	14 **RET**
B	6 **CLR** R_1	6 **CLR** IR_1	6 **XOR** r_1,r_2	6 **XOR** r_1,Ir_2	10 **XOR** R_2,R_1	10 **XOR** IR_2,R_1	10 **XOR** R_1,IM	**NOTE** **E**	↓	↓	↓	↓	↓	↓	↓	16/6 **IRET**
C	6 **RRC** R_1	6 **RRC** IR_1	16/18 **CPIJE** Ir,r_2,RA	12 **LDC*** r_1,Irr_2	10 **LDW** RR_2,RR_1	10 **LDW** IR_2,RR_1	12 **LDW** RR_1,IML	6 **LD** r_1,Ir_2	↓	↓	↓	↓	↓	↓	↓	6 **RCF**
D	6 **SRA** R_1	6 **SRA** IR_1	16/18 **CPIJNE** Ir_1,r_2,RA	12 **LDC*** r_2,Irr_1	20 **CALL** IA_1		10 **LD** IR_1,IM	6 **LD** Ir_1,r_2	↓	↓	↓	↓	↓	↓	↓	6 **SCF**
E	6 **RR** R_1	6 **RR** IR_1	16 **LDCD*** r_1,Irr_2	16 **LDCI*** r_1,Irr_2	10 **LD** R_2,R_1	10 **LD** IR_2,R_1	10 **LD** R_1,IM	18 **LDC*** r_1,Irr_2,xs	↓	↓	↓	↓	↓	↓	↓	6 **CCF**
F	8 **SWAP** R_1	8 **SWAP** IR_1	16 **LDCPD*** r_2,Irr_1	16 **LDCPI*** r_2,Irr_1	18 **CALL** IRR_1	10 **LD** R_2,IR_1	18 **CALL** DA_1	18 **LDC*** r_2,Irr_1,xs	↓	↓	↓	↓	↓	↓	↓	6 **NOP**

NOTE A		
	16/18 **BTJRF** r_2,b,RA	16/18 **BTJRT** r_2,b,RA

NOTE B		
	8 **BITR** r_1,b	8 **BITS** r_1,b

NOTE C			
	6 **SRP** IM	6 **SRP0** IM	6 **SRP1** IM

NOTE D		
	20 **LDC*** r_1,Irr_2,xL	20 **LDC*** r_1,DA_2

NOTE E		
	20 **LDC*** r_2,Irr_2,xL	20 **LDC*** r_2,DA_1

Legend:
r = 4-bit address
R = 8-bit address
b = bit number
R_1 or r_1 = dst address
R_2 or r_2 = src address

***Examples:**
BOR r_0-R_2
is BOR r_1,b,R_2
or BOR r_2,b,R_1
LDC r_1,Irr_2
is LDC r_1,Irr_2 = program
or LDE r_1,Irr_2 = data

Sequence:
Opcode, first, second, third operands

NOTE: The blank areas are not defined.

Figure 11. Opcode Map

INSTRUCTIONS

Table 7. Super8 Instructions

Mnemonic	Operands	Instruction
Load Instructions		
CLR	dst	Clear
LD	dst, src	Load
LDB	dst, src	Load bit
LDC	dst, src	Load program memory
LDE	dst, src	Load data memory
LDCD	dst, src	Load program memory and decrement
LDED	dst, src	Load data memory and decrement
LDCI	dst, src	Load program memory and increment
LDEI	dst, src	Load data memory and increment
LDCPD	dst, src	Load program memory with pre-decrement
LDEPD	dst, src	Load data memory with pre-decrement
LDCPI	dst, src	Load program memory with pre-increment
LDEPI	dst, src	Load data memory with pre-increment
LDW	dst, src	Load word
POP	dst	Pop stack
POPUD	dst, src	Pop user stack (decrement)
POPUI	dst, src	Pop user stack (increment)
PUSH	src	Push stack
PUSHUD	dst, src	Push user stack (decrement)
PUSHUI	dst, src	Push user stack (increment)
Arithmetic Instructions		
ADC	dst, src	Add with carry
ADD	dst, src	Add
CP	dst, src	Compare
DA	dst	Decimal adjust
DEC	dst	Decrement
DECW	dst	Decrement word
DIV	dst, src	Divide
INC	dst	Increment
INCW	dst	Increment word
MULT	dst, src	Multiply
SBC	dst, src	Subtract with carry
SUB	dst, src	Subtract
Logical Instructions		
AND	dst, src	Logical AND
COM	dst	Complement
OR	dst, src	Logical OR
XOR	dst, src	Logical exclusive

Mnemonic	Operands	Instruction
Program Control Instructions		
BTJRT	dst, src	Bit test jump relative on True
BTJRF	dst, src	Bit test jump relative on False
CALL	dst	Call procedure
CPIJE	dst, src	Compare, increment and jump on equal
CPIJNE	dst, src	Compare, increment and jump on non-equal
DJNZ	r, dst	Decrement and jump on non-zero
ENTER		Enter
EXIT		Exit
IRET		Return from interrupt
JP	cc, dst	Jump on condition code
JP	dst	Jump unconditional
JR	cc, dst	Jump relative on condition code
JR	dst	Jump relative unconditional
NEXT		Next
RET		Return
WFI		Wait for interrupt
Bit Manipulation Instructions		
BAND	dst, src	Bit AND
BCP	dst, src	Bit compare
BITC	dst	Bit complement
BITR	dst	Bit reset
BITS	dst	Bit set
BOR	dst, src	Bit OR
BXOR	dst, src	Bit exclusive OR
TCM	dst, src	Test complement under mask
TM	dst, src	Test under mask
Rotate and Shift Instructions		
RL	dst	Rotate left
RLC	dst	Rotate left through carry
RR	dst	Rotate right
RRC	dst	Rotate right through carry
SRA	dst	Shift right arithmetic
SWAP	dst	Swap nibbles
CPU Control Instructions		
CCF		Complement carry flag
DI		Disable interrupts
EI		Enable interrupts
NOP		Do nothing
RCF		Reset carry flag
SB0		Set bank 0
SB1		Set bank 1
SCF		Set carry flag
SRP	src	Set register pointers
SRP0	src	Set register pointer zero
SRP1	src	Set register pointer one

INTERRUPTS

The Super8 interrupt structure contains 8 levels of interrupt, 16 vectors, and 27 sources.

Interrupt priority is assigned by level, controlled by the Interrupt Priority register (IPR). Each level is masked (or enabled) according to the bits in the Interrupt Mask register (IMR), and the entire interrupt structure can be disabled by clearing a bit in the System Mode register (R222).

The three major components of the interrupt structure are sources, vectors, and levels. These are shown in Figure 10 and discussed in the following paragraphs.

Sources

A source is anything that generates an interrupt. This can be internal or external to the Super8 MCU. Internal sources are hardwired to a particular vector and level, while external sources can be assigned to various external events.

Vectors

The 16 vectors are divided unequally among the eight levels. For example, vector 12 belongs to level 2, while level 3 contains vectors 0, 2, 4, and 6.

The vector number is used to generate the address of a particular interrupt servicing routine; therefore all interrupts using the same vector must use the same interrupt handling routine.

Levels

Levels provide the top level of priority assignment. While the sources and vectors are hardwired within each level, the priorities of the levels can be changed by using the Interrupt Priority register (see Figure 8 for bit details).

If more than one interrupt source is active, the source from the highest priority level will be serviced first. If both sources are from the same level, the source with the lowest vector will have priority. For example, if the UART Receive Data bit and UART Parity Error bit are both active, the UART Parity Error bit will be serviced first because it is vector 16, and UART receive data is vector 20.

The levels are shown in Figure 12.

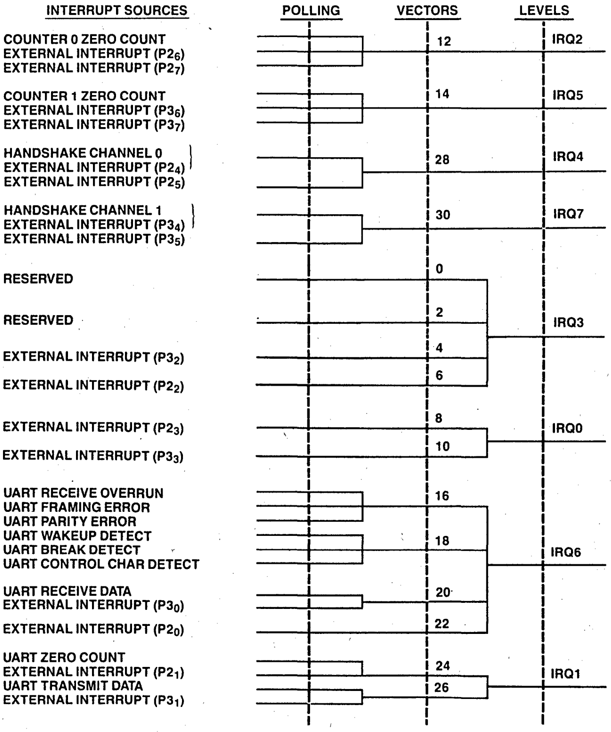

INTERRUPT SOURCES
POLLING
VECTORS
LEVELS
COUNTER 0 ZERO COUNT
EXTERNAL INTERRUPT (P2$_6$)
EXTERNAL INTERRUPT (P2$_7$)
12
IRQ2
COUNTER 1 ZERO COUNT
EXTERNAL INTERRUPT (P3$_6$)
EXTERNAL INTERRUPT (P3$_7$)
14
IRQ5
HANDSHAKE CHANNEL 0
EXTERNAL INTERRUPT (P2$_4$)
EXTERNAL INTERRUPT (P2$_5$)
28
IRQ4
HANDSHAKE CHANNEL 1
EXTERNAL INTERRUPT (P3$_4$)
EXTERNAL INTERRUPT (P3$_5$)
30
IRQ7
RESERVED
0
RESERVED
2
IRQ3
EXTERNAL INTERRUPT (P3$_2$)
4
EXTERNAL INTERRUPT (P2$_2$)
6
EXTERNAL INTERRUPT (P2$_3$)
8
IRQ0
EXTERNAL INTERRUPT (P3$_3$)
10
UART RECEIVE OVERRUN
UART FRAMING ERROR
UART PARITY ERROR
16
UART WAKEUP DETECT
UART BREAK DETECT
UART CONTROL CHAR DETECT
18
IRQ6
UART RECEIVE DATA
EXTERNAL INTERRUPT (P3$_0$)
20
EXTERNAL INTERRUPT (P2$_0$)
22
UART ZERO COUNT
EXTERNAL INTERRUPT (P2$_1$)
24
UART TRANSMIT DATA
EXTERNAL INTERRUPT (P3$_1$)
26
IRQ1

Figure 12. Interrupt Levels and Vectors

Enables

Interrupts can be enabled or disabled as follows:

- Interrupt enable/disable. The entire interrupt structure can be enabled or disabled by setting bit 0 in the System Mode register (R222).
- Level enable. Each level can be enabled or disabled by setting the appropriate bit in the Interrupt Mask register (R221).
- Level priority. The priority of each level can be controlled by the values in the Interrupt Priority register (R255, Bank 0).
- Source enable/disable. Each interrupt source can be enabled or disabled in the sources' Mode and Control register.

Service Routines

Before an interrupt request can be granted, a) interrupts must be enabled, b) the level must be enabled, c) it must be the highest priority interrupting level, d) it must be enabled at the interrupting source, and e) it must have the highest priority within the level.

If all this occurs, an interrupt request is granted.

The Super8 then enters an interrupt machine cycle that completes the following sequence:

- It resets the Interrupt Enable bit to disable all subsequent interrupts.
- It saves the Program Counter and status flags on the stack.
- It branches to the address contained within the vector location for the interrupt.
- It passes control to the interrupt servicing routine.

When the interrupt servicing routine has serviced the interrupt, it should issue an interrupt return (IRET) instruction. This restores the Program Counter and status flags and sets the Interrupt Enable bit in the System Mode register.

Fast Interrupt Processing

The Super8 provides a feature called fast interrupt processing, which completes the interrupt servicing in 6 clock periods instead of the usual 22.

Two hardware registers support fast interrupts. The Instruction Pointer (IP) holds the starting address of the service routine, and saves the PC value when a fast interrupt occurs. A dedicated register, FLAG', saves the contents of the FLAGS register when a fast interrupt occurs.

To use this feature, load the address of the service routine in the Instruction Pointer, load the level number into the Fast Interrupt Select field, and turn on the Fast Interrupt Enable bit in the System Mode register.

When an interrupt occurs in the level selected for fast interrupt processing, the following occurs:

- The contents of the Instruction Pointer and Program Counter are swapped.
- The contents of the Flag register are copied into FLAG'.
- The Fast Interrupt Status Bit in FLAGS is set.
- The interrupt is serviced.
- When IRET is issued after the interrupt service outline is completed, the Instruction Pointer and Program Counter are swapped again.
- The contents of FLAG' are copied back into the Flag register.
- The Fast Interrupt Status bit in FLAGS is cleared.

The interrupt servicing routine selected for fast processing should be written so that the location after the IRET instruction is the entry point the next time the (same) routine is used.

Level or Edge Triggered

Because internal interrupt requests are levels and interrupt requests from the outside are (usually) edges, the hardware for external interrupts uses edge-triggered flip-flops to convert the edges to levels.

The level-activated system requires that interrupt-serving software perform some action to remove the interrupting source. The action involved in serving the interrupt may remove the source, or the software may have to actually reset the flip-flops by writing to the corresponding Interrupt Pending register.

STACK OPERATION

The Super8 architecture supports stack operations in the register file or in data memory. Bit 1 in the external Memory Timing register (R254 bank 0) selects between the two.

Register pair 216-217 forms the Stack Pointer used for all stack operations. R216 is the MSB and R217 is the LSB.

The Stack Pointer always points to data stored on the top of the stack. The address is decremented prior to a PUSH and incremented after a POP.

The stack is also used as a return stack for CALLs and interrupts. During a CALL, the contents of the PC are saved on the stack, to be restored later. Interrupts cause the contents of the PC and FLAGS to be saved on the stack, for recovery by IRET when the interrupt is finished.

When the Super8 is configured for an internal stack (using the register file), R217 contains the Stack Pointer. R216 may be used as a general-purpose register, but its contents will be changed if an overflow or underflow occurs as the result of incrementing or decrementing the stack address during normal stack operations.

User-Defined Stacks

The Super8 provides for user-defined stacks in both the register file and program or data memory. These can be made to increment or decrement on a push by the choice of opcodes. For example, to implement a stack that grows from low addresses to high addresses in the register file, use PUSHUI and POPUD. For a stack that grows from high addresses to low addresses in data memory, use LDEI for pop and LDEPD for push.

COUNTER/TIMERS

The Super8 has two identical independently programmable 16-bit counter/timers that can be cascaded to produce a single 32-bit counter. They can be used to count external events, or they can obtain their input internally. The internal input is obtained by dividing the crystal frequency by four.

The counter/timers can be set to count up or down, by software or external events. They can be set for single or continuous cycle counting, and they can be set with a bi-value option, where two preset time constants alternate in loading the counter each time it reaches zero. This can be used to produce an output pulse train with a variable duty cycle.

The counter/timers can also be programmed to capture the count value at an external event or generate an interrupt whenever the count reaches zero. They can be turned on and off in response to external events by using a gate and/or a trigger option. The gate option enables counts only when the gate line is Low; the trigger option turns on the counter after a transient High. The gate and trigger options used together cause the counter/timer to work in gate mode after initially being triggered.

The control and status register bits for the counter/timers are shown in Figure 5.

DMA

The Super8 features an on-chip Direct Memory Access (DMA) channel to provide high bandwidth data transmission capabilities. The DMA channel can be used by the UART receiver, UART transmitter, or handshake channel 0. Data can be transferred between the peripheral and contiguous locations in either the register file or external data memory. A 16-bit count register determines the number of transactions to be performed; an interrupt can be generated when the count is exhausted. DMA transfers to or from the register file require six CPU clock cycles; DMA transfers to or from external memory take ten CPU clock cycles, excluding wait states.

ABSOLUTE MAXIMUM RATINGS

Voltage on all pins with respect
to ground . . . . $-0.3V$ to $+7.0V$
Ambient Operating
Temperature . . . . .See Ordering Information
Storage Temperature . . . . –65°C to +150°C

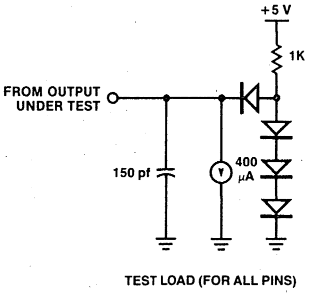
Stresses greater than these may cause permanent damage to the device. This is a stress rating only; operation of the device under conditions more severe than those listed for operating conditions may cause permanent damage to the device. Exposure to absolute maximum ratings for extended periods may also cause permanent damage.

STANDARD TEST CONDITIONS

Figure 14 shows the setup for standard test conditions. All voltages are referenced to ground, and positive current flows into the reference pin.

Standard conditions are:

- $+4.75V \leqslant V_{CC} \leqslant +5.25V$
- GND = 0V
- $0°C \leqslant T_A \leqslant +70°C$

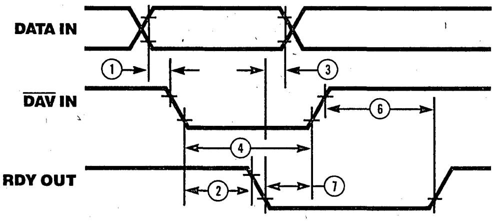


Standard Test Load

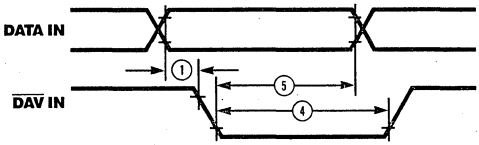
DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8	V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	–0.3	0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.2	V_{CC}	V	
V_{IL}	Input Low Voltage	–0.3	0.8	V	
V_{RH}	Reset Input High Voltage	3.8	V_{CC}	V	
V_{RL}	Reset Input Low Voltage	–0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -400\,\mu A$
V_{OL}	Output Low Voltage		0.4	V	$I_{OL} = +4.0\,mA$
I_{IL}	Input Leakage	–10	10	μA	
I_{OL}	Output Leakage	–10	10	μA	
I_{IR}	Reset Input Current		–50	μA	
I_{CC}	V_{CC} Supply Current		320	mA	

INPUT HANDSHAKE TIMING



Fully Interlocked Mode



Strobed Mode

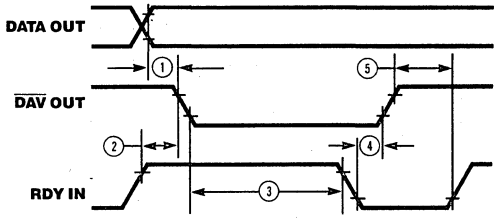
AC CHARACTERISTICS (20 MHz)

Input Handshake

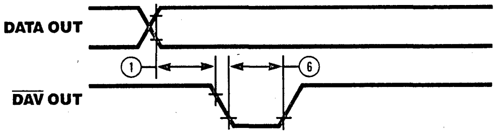
Number	Symbol	Parameter	Min	Max	Notes*‡
1	TsDI(DAV)	Data In to Setup Time	0		
2	TdDAVIf(RDY)	$\overline{\text{DAV}}$ ↓ Input to RDY ↓ Delay		200	1
3	ThDI(RDY)	Data In Hold Time from RDY ↓	0		
4	TwDAV	$\overline{\text{DAV}}$ In Width	45		
5	ThDI(DAV)	Data In Hold Time from $\overline{\text{DAV}}$ ↓	130		
6	TdDAV(RDY)	$\overline{\text{DAV}}$ ↑ Input to RDY ↑ Delay		100	2
7	TdRDYf(DAV)	RDY ↓ Output to $\overline{\text{DAV}}$ ↑ Delay	0		

NOTES:
1. Standard Test Load
2. This time assumes user program reads data before $\overline{\text{DAV}}$ Input goes high. RDY will not go high before data is read.
‡Times given are in ns.
*Times are preliminary and subject to change.

OUTPUT HANDSHAKE TIMING



Fully Interlocked Mode



Strobed Mode

AC CHARACTERISTICS (12 MHz, 20 MHz)

Output Handshake

Number	Symbol	Parameter	Min	Max	Notes*‡
1	TdDO(DAV)	Data Out to $\overline{\text{DAV}}$ ↓ Delay	90		1,2
2	TdRDYr(DAV)	RDY ↑ Input to $\overline{\text{DAV}}$ ↓ Delay	0	110	1
3	TdDAVOf(RDY)	$\overline{\text{DAV}}$ ↓ Output to RDY ↓ Delay	0		
4	TdRDYf(DAV)	RDY ↓ Input to $\overline{\text{DAV}}$ ↑ Delay	0	110	1
5	TdDAVOr(RDY)	$\overline{\text{DAV}}$ ↑ Output to RDY ↑ Delay	0		
6	TwDAVO	$\overline{\text{DAV}}$ Output Width	150		2

NOTES:
1. Standard Test Load
2. Time given is for zero value in Deskew Counter. For nonzero value of n where n = 1, 2,. . . 15 add 2 × n × TpC to the given time.
‡Times given are in ns.
*Times are preliminary and subject to change.

AC CHARACTERISTICS (12 MHz)

Read/Write

			Normal Timing		Extended Timing		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes‡*
1	TdA(AS)	Address Valid to $\overline{\text{AS}}$ ↑ Delay	35		115		
2	TdAS(A)	$\overline{\text{AS}}$ ↑ to Address Float Delay	65		150		
3	TdAS(DR)	$\overline{\text{AS}}$ ↑ to Read Data Required Valid		270		600	1
4	TwAS	$\overline{\text{AS}}$ Low Width	65		150		
5	TdA(DS)	Address Float to $\overline{\text{DS}}$ ↓	20		20		
6a	TwDS(Read)	$\overline{\text{DS}}$ (Read) Low Width	225		470		1
6b	TwDS(Write)	$\overline{\text{DS}}$ (Write) Low Width	130		295		1
7	TdDS(DR)	$\overline{\text{DS}}$ ↓ to Read Data Required Valid		180		420	1
8	ThDS(DR)	Read Data to $\overline{\text{DS}}$ ↑ Hold Time	0		0		
9	TdDS(A)	$\overline{\text{DS}}$ ↑ to Address Active Delay	50		135		
10	TdDS(AS)	$\overline{\text{DS}}$ ↑ to $\overline{\text{AS}}$ ↓ Delay	60		145		
11	TdDO(DS)	Write Data Valid to $\overline{\text{DS}}$ (Write) ↓ Delay	35		115		
12	TdAS(W)	$\overline{\text{AS}}$ ↑ to Wait Delay		220		600	2
13	ThDS(W)	$\overline{\text{DS}}$ ↑ to Wait Hold Time	0		0		
14	TdRW(AS)	R/$\overline{\text{W}}$ Valid to $\overline{\text{AS}}$ ↑ Delay	50		135		

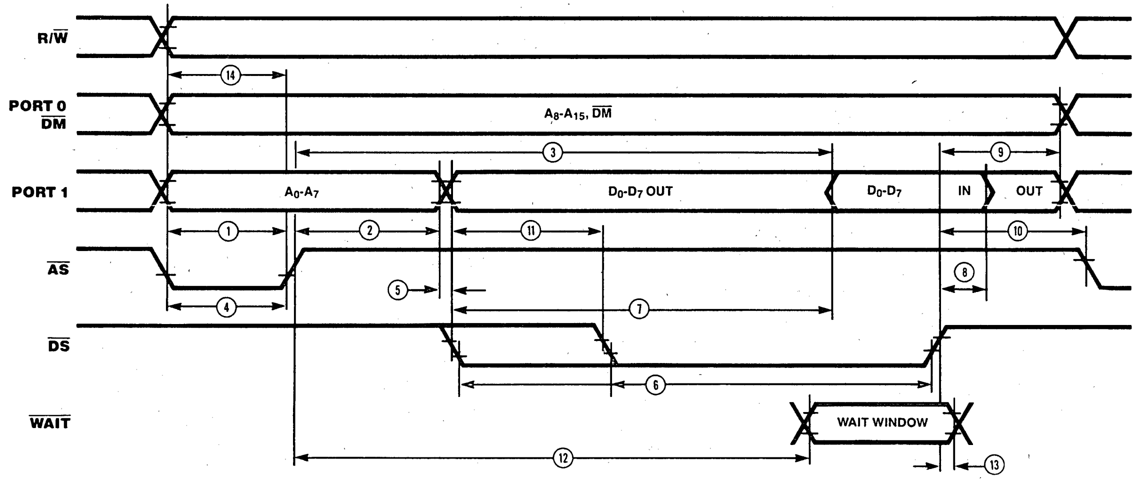
NOTES:
1. WAIT states add 167 ns to these times.
2. Auto-wait states add 167 ns to this time.
‡ All times are in ns and are for 12 MHz input frequency.
* Timings are preliminary and subject to change.

AC CHARACTERISTICS (20 MHz)

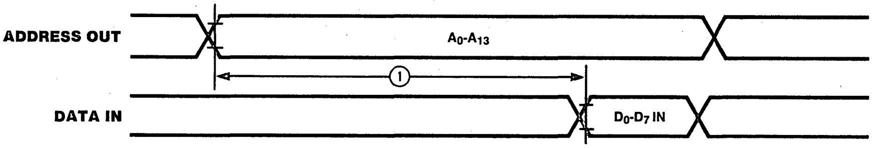
Read/Write

			Normal Timing		Extended Timing		
Number	Symbol	Parameter	Min	Max	Min	Max	Notes‡*
1	TdA(AS)	Address Valid to $\overline{\text{AS}}$ ↑ Delay	20		50		
2	TdAS(A)	$\overline{\text{AS}}$ ↑ to Address Float Delay	35		85		
3	TdAS(DR)	$\overline{\text{AS}}$ ↑ to Read Data Required Valid		150		335	1
4	TwAS	$\overline{\text{AS}}$ Low Width	35		85		
5	TdA(DS)	Address Float to $\overline{\text{DS}}$ ↓	0		0		
6a	TwDS(Read)	$\overline{\text{DS}}$ (Read) Low Width	125		275		1
6b	TwDS(Write)	$\overline{\text{DS}}$ (Write) Low Width	65		165		1
7	TdDS(DR)	$\overline{\text{DS}}$ ↓ to Read Data Required Valid		80		225	1
8	ThDS(DR)	Read Data to $\overline{\text{DS}}$ ↑ Hold Time	0		0		
9	TdDS(A)	$\overline{\text{DS}}$ ↑ to Address Active Delay	20		70		
10	TdDS(AS)	$\overline{\text{DS}}$ ↑ to $\overline{\text{AS}}$ ↓ Delay	30		80		
11	TdDO(DS)	Write Data Valid to $\overline{\text{DS}}$ (Write) ↓ Delay	10		50		
12	TdAS(W)	$\overline{\text{AS}}$ ↑ to Wait Delay		90		335	2
13	ThDS(W)	$\overline{\text{DS}}$ ↑ to Wait Hold Time	0		0		
14	TdRW(AS)	R/$\overline{\text{W}}$ Valid to $\overline{\text{AS}}$ ↑ Delay	20		70		

NOTES:
1. WAIT states add 100 ns to these times.
2. Auto-wait states add 100 ns to this time.
‡ All times are in ns and are for 20 MHz input frequency.
* Timings are preliminary and subject to change.



External Memory Read and Write Timing



EPROM Read Timing

AC CHARACTERISTICS (20 MHz)

EPROM Read Cycle

Number	Symbol	Parameter	Min	Max	Notes‡*
1	TdA(DR)	Address Valid to Read Data Required Valid		170	1

NOTES:
1. WAIT states add 167 ns to these times.
‡All times are in ns and are for 12 MHz input frequency.
*Timings are preliminary and subject to change.

August 1987

GETTING STARTED WITH THE ZILOG SUPER8

by Charles M. Link, II

Any time an engineer switches to a new processor, he usually begins the time consuming process of learning the quirks of the new part. This article is the first of a series of articles written to speed that transition time from any other processor to the Zilog Super8.

Getting started is the most difficult part of switching to a strange new processor and development tools. Weeks can be spent just getting the first lines of initialization code written and successfully assembled. Testing the code becomes another problem. The soft ɿre from this article series has been tested and it should be possible to copy most of the software directly to a user's application. All of the software is available in machine readable form as noted at the end of the article.

This first article demonstrates the proper initialization of the Zilog Super8 microcontroller. It sets up a Z8800 ROMLESS for 64K bytes of external program memory, although most typical applications probably do not require more than maybe 4K or 8K bytes. Ports 2 and 3, which are bit mappable as inputs or outputs, are set into the output mode. Port 4, also bit mappable, is set into the input mode. A hardware schematic has been included as an example.

The hardware schematic shown defines a simple Super8 implementation that was used to test the code in this series of articles. This example defines a simple evaluation board that contains 32K bytes of programable EPROM, and up to 32K bytes of RAM. The design contains a simple RS-232 interface that is used in future articles of the series. The entire board, including the RS-232 interface, is powered from 5 volts. The RAM battery option allows the software to be downloaded into the RAM and saved if power fails. Additional logic on the design allows a user to protect the lower half of RAM with a simple jumper change. This prevents the processor from destroying executable code if it goes off into space on a power failure.

Specifically, the ROMLESS Super8 is used as the core. The Super8 requires a latch to demultiplex the address from the data bus. A 74LS373 fits nicely here, requiring only an inverter to correct for the address strobe. The 'LS373 with inverter is preferred here rather than a single 'LS374 because the 'LS373 is a transparent latch and will present the address earlier than the 'LS374. JU1 selects the EPROM size, correcting for the /PGM pin on 2764 and 27128 EPROMs. It is necessary to use pull down resistors on the upper 4 bits of the address bus because on reset, the ROMLESS Super8 defines only 12 bits for address; the other 4 are set as inputs. Since LS-TTL devices require more current to pull down the inputs, this pull down trick will only work for MOS and CMOS inputs, hence the requirement for the logic chips in this design to be HCT type devices.

The remaining logic is required to select the EPROM or RAM. JU2 selects the half-RAM protect mode. JU3 is set to determine what size ram to protect. This circuit allows the lower half of CMOS battery backed RAM to be read only, and removes chip select on any writes to that address space. Of course, that exact circuitry and the battery is optional, and might be replaced by a power threshold detector. On the other front, a Maxim MAX 232 provides the RS-232 interface requiring only 5 volts.

To make the software initialization more interesting, a few other typical initialization tasks are demonstrated. The entire block of registers (user ram) is cleared to zero, and one of the counter timer units is initialized to provide a periodic interrupt to form the heart of a real time clock function.

The program shows the typical pseudo-op usage demonstrated. This article series uses a cross assembler available from Zilog for either an IBM PC or a VAX operating under VMS. The program begins by defining the registers used as general purpose storage. This is done so the user does not have to refer to register numbers, but may refer to a name equated to the register.

The first 32 bytes of every program (beginning at 0000H) always contain the interrupt vectors for the different sources. Using the Zilog assembler, the .WORD pseudo-op defines a pair of bytes for each of the 16 sources. Program execution begins at location 0020H. Since copyright requirements usually require the notice as close to the beginning as possible, it becomes necessary to jump around an ASCII string. The .ASCII pseudo-op generates the necessary string for this notice.

The source code describes almost completely, without further explaination, the entire initialization. Once initialized, the processor loops in a WAIT loop waiting on the periodic interrupt generated by the counter/timer. The counter timer interrupts 60 times per second, and the interrupt bumps ram storage locations representing seconds, minutes, and hours. Each time a location is bumped, an external port line is toggled so that those without emulators can see some activity with an oscilloscope.

One point of notice, is the interrupt service routine for the timer. One must reset the end of count interrupt bit (the source of interrupt) before exiting the interrupt service routine.

In the next article of this series, we will take the same basic initialization routine and modify it to support the serial UART. That article will demonstrate polled serial communications using the Zilog Super 8.

[Editors note: The sofware for this series is available on an IBM PC diskette and is included with the Super 8 Emulator package available from Creative Technology Corporation, 5144 Peachtree Road, Suite 301, Atlanta, GA 30341. (404) 455-8255. Any Zilog Field Application engineer should also be able to provide copies of the software on a user provided diskette.]

```
;
        .TITLE  Sample Zilog Super 8 Initialization
;
;=============================================================
;=      TITLE:          INIT.S8                             =
;=      DATE:           JUNE 17, 1986                       =
;=      PURPOSE:        TO DEMONSTRATE INITIALIZATION       =
;=                      OF THE ZILOG SUPER 8 USING THE      =
;=                      ZILOG ASMS8 ASSEMBLER               =
;=      PROGRAMMER:     CHARLES M. LINK, II                 =
;=============================================================
;
;
;
        .PAGE   55      ;set maximum page size to 55 lines
;
;*************************************************************
;*                                                           *
;*                  REGISTER EQUATE TABLE                    *
;*                                                           *
;*************************************************************
;
period: .equ    0       ;period timer
second: .equ    1       ;seconds timer
minute: .equ    2       ;minutes timer
hours:  .equ    3       ;hours timer
;
;*************************************************************
;*                                                           *
;*                  INTERRUPT VECTOR TABLE                   *
;*                                                           *
;*************************************************************
;
INTR0:  .WORD   INTRET          ;this area should always be defined
INTR1:  .WORD   INTRET          ;as it reserves the lower 32 bytes
INTR2:  .WORD   INTRET          ;for the interrupt table.  the name
INTR3:  .WORD   INTRET          ;of the subroutine for each particular
INTR4:  .WORD   INTRET          ;interrupt service would normally be
INTR5:  .WORD   INTRET          ;named here.
INTR6:  .WORD   TIMERO
INTR7:  .WORD   INTRET
INTR8:  .WORD   INTRET
INTR9:  .WORD   INTRET
INTR10: .WORD   INTRET
INTR11: .WORD   INTRET
INTR12: .WORD   INTRET
INTR13: .WORD   INTRET
INTR14: .WORD   INTRET
INTR15: .WORD   INTRET
;
;*************************************************************
;*                                                           *
;*              START OF PROGRAM EXECUTION                   *
;*                                                           *
;*************************************************************
;
```

```
START:  jr      START1          ;program execution unconditionally
                                ;begins at this location after reset
                                ;and power up.
        .ASCII  'REL 0 6/16/86' ;jump around optional ascii string
                                ;containing release info, copyright, etc.
START1: di                      ;begin
        sb0                     ;select register bank 0
        ld      EMT,#00000000B  ;external memory timing=no wait input, normal
                                ;memory timing, no wait states, stack internal,
                                ;and DMA internal
        ld      P0,#00H         ;address begins at 0000h, set upper byte
        ld      P0M,#11111111B  ;select all lines as address
        ld      PM,#00110000B   ;enable port 0 as upper 8 bits address
        ld      H1C,#00000000B  ;handshake not enabled port 0
;
;port 1 is defined in romless part as address/data.  it is not necessary
;here to initialize that port
;
        ld      P2,#00H         ;port 2 outputs low
        ld      P3,#00H         ;port 3 outputs low
        ld      P2AM,#10101010B ;p30,31,20,21 as output
        ld      P2BM,#10101010B ;p32,33,22,23 as output
        ld      P2CM,#10101010B ;p34,35,24,25 as output
        ld      P2DM,#10101010B ;p36,37,26,27 as output
;
        ld      P4,#00000000B   ;clear port 4 register
        ld      P4D,#11111111B  ;set all bits of P4 as inputs
        ld      P4OD,#00000000B ;active push/pull [not necessary since all
                                ; bits are inputs
;
;basic Super 8 I/O is initialized, now internal registers
;
        ld      RP0,#0C0H       ;set working register low to lower 8 bytes
        ld      RP1,#0C8H       ;set working register high to upper 8 bytes
        ld      SPL,#0FFH       ;set stack pointer to start at top of set two
                                ;note here that only lower 8 bits are used
                                ;for stack pointer.  location 0FFH is wasted
                                ;as stack operation.  SPH is general purpose
                                ;storage.
;
;now clear the internal memory and stack area
;
        ld      SPH,#0FFH       ;point to top of general purpose register
ZERO:   clr     @SPH            ;zero it
        dec     SPH
        jr      nz,ZERO         ;do it until register set is all cleared
        clr     @SPH            ;zero last register
;
;now everything except working registers is cleared
;
;cpu and memory now initialized, set up timer for real time clock
;
        ld      SYM,#00000000B  ;disable fast interrupt response
        ld      IPR,#00000010B  ;interrupt priority
                                ;IRQ2>IRQ3>IRQ4>IRQ5>IRQ6>IRQ7>IRQ0>IRQ1
        ld      IMR,#00000100B  ;enable only interrupt 2
        sb1                     ;select bank 1
        ld      C0TCH,#^HB(50000)       ;high byte of time constant
        ld      C0TCL,#^LB(50000)       ;low byte of time constant
                                ;12,000,000 hertz / 4 / 50,000 = 60 hertz
                                ;12 Mhz is xtal freq, 4 is internal divider
        ld      C0M,#00000100B  ;p27,37 is I/O, programmed up/down, no capture
                                ;timer mode is selected
        sb0                     ;select bank 0
        ld      C0CT,#10100101B ;continuous, count down, load counter,
                                ;zero count interrupt enable, enable counter
;
;timer is initialized, now lets enable interrupts and wait
;
        ei                      ;enable interrupts
WAIT:   nop
        nop
        nop
        nop
        jr      WAIT            ;loop back
;
```

```
        nop
        nop
        nop
TIMER0: inc     period          ;bump periodic counter (60 hertz)
        cp      period,#60      ;one second yet?
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000001B   ;complement the second bit
        clr     period          ;start it over again
        inc     second          ;bump the seconds timer
        cp      second,#60      ;reached maximum
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000010B   ;complement the minute bit
        clr     second          ;start it over again
        inc     minute          ;bump the minutes timer
        cp      minute,#60      ;reached maximum
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000100B   ;complement the hour bit
        clr     minute          ;start it over again
        inc     hours           ;bump the hours timer
        cp      hours,#24       ;reached maximum
        jr      ne,NOROLL       ;no rollover
        clr     hours           ;start it over again
NOROLL: or      C0CT,#00000010B ;reset end of count interrupt
        nop
        nop
INTRET: iret                    ;and return from interrupt
;
;
;
        .END
```

Zilog Application Note

August 1987

POLLED ASYNCHRONOUS SERIAL OPERATION WITH THE ZILOG SUPER8

by Charles M. Link, II

The transition from one processor to another often involves many hours of trial-and-error software development to determine the quirks (manufacturers call it features) of the part. Once the real features are discovered, programming the processor to perform as described can be hazardous to one's health. This article, the second in a series of eight, attempts to introduce the Zilog Super8 user to the serial communications port, and its initialization in a polled serial environment.

The universal asynchronous receiver/transmitter (UART) on the Super8 is a fairly unique implementation among single chip microcomputers in that it supports all of the functions generally available only on chip level UARTs. The UART is a close approximation of the Z80 DART device in one channel. It supports independent receiver/transmitter clocking, 5 to 8 bits per character, plus optional odd or even parity, and even an optional wake-up bit. The UART can serve full duplex communications via polled, interrupt, or DMA modes of operation. Auto-echo and internal loopback can be programmed as options. The most unique of the UART features is the character match and interrupt option.

The following article describes the initialization and use of the UART in a polled environment. This software has been tested and provides several routines that may be copied into a user's software. Although the demonstration software does not do much, it is fully functional as a stand-alone program, and may be "burned" into eprom as a test.

The basic software is almost the same general purpose initialization software from the first article in the series. Routines set-up counter/timer 0 for a real time clock option. Note, however, the change to configuration register P2AM. It is necessary to configure port 30 as input for receive data and p31 as output for transmit data.

The UART initialization sequence begins by setting the functions in the UART MODE A register. Since the UMA register is in the alternate bank, the instruction SB1 must be executed to gain access to the following registers. The loaded data selects a X16 clock, 8 bits per character, no parity, and no wake up values. Note that the clock options are X1, X16, X32, and X64. For true asynchronous operation, a clock multiplier option of at least X16 is required. The X1 mode could be used for externally syncing the received data to the UART. The transmitter is not affected.

Next, the baud rate generator must be loaded. The formula for determining the baud rate is shown below:

TIME CONSTANT = (XTAL FREQ / 8 / CLOCK MULT / DESIRED RATE) - 1

where TIME CONSTANT is a 16 bit value, XTAL FREQ is the crystal ifrequency in hertz, CLOCK MULT is the clock rate loaded into UART MODE A register (as above X1, X16, X32, and X64), and DESIRED rate is the desired bit rate in bits per second. Note that the baud rate generator may be used as an additional counter, and may be loaded with any value permitting just about any crystal frequency to operate the Super8.

The cross-assembler permitted a single 16-bit decimal number to be loaded into the UART BAUD RATE GENERATOR, high and low byte, without unnecessary figuring using the high/low byte pseudo-op.

The initialization sequence continues, with the UART MODE B register next. This example sends port 21 data to the port 21 pin. An option allows different clocks to be sent out from this pin. It could be used for clocking external logic, or for diagnostic purposes to make sure the baud rate generator is running. Auto-echo is not selected in this application, as that is primarily what the example software does. The receive and transmit clock input is the baud rate generator and the generator source is the internal clock; the crystal divided by four. Since the baud rate generator has been loaded, it is enabled, and the UART is set for normal operation (without loopback). Loopback operation permits transmitting and receiving data without any external logic in front of the Super8.

The UART TRANSMIT CONTROL register is initialized next in the sequence. Select transmit data out on port 31 and transmit enable. The stop bits are optional, and the DMA and WAKE-UP enables are for features discussed in future application articles. At this point, the transmitter is operational, and except for housekeeping, is usable. The housekeeping is in reference to selecting the bank 0 by executing the SB0 instruction.

Since polled mode communications are desired, all of the UART interrupts are disabled by loading the UART INTERRUPT ENABLE with all zeros. Lastly, the receiver must be enabled by setting bit 0 of the UART RECEIVE CONTROL register.

This program primarily sends a message to the console and then accepts input from the console and echos it upon receiving a carriage return. It is necessary to delay sending data to the console after initialization because the transmit data line is in the SPACE state when idle. Alternately, add a pull-up resistor to the output, and while idle and before initialized, it would exibit the MARK state.

The transmit character routine "SENDC" monitors the TRANSMIT BUFFER EMPTY bit of the UART TRANSMIT CONTROL register. When this bit is a "1", the transmit buffer is empty and may be loaded with a new character for transmission. To transmit a character, load the character into the UART data register (UIO).

The receive character routine "GETC" monitors the RECEIVE CHARACTER AVAILABLE bit of the UART RECEIVE CONTROL register. When this bit is a "1", a new character has been received by the UART.

The polled mode of UART operation is simple. Making the UART operate in an interrupt mode requires a few minor modifications, and DMA mode requires a few more modifications. Those modes are the subject of future application articles in this series.

```
;
        .TITLE  Sample Zilog Super 8 Serial Port Initialization
;
;
;===========================================================
;=      TITLE:          UART1.S                            =
;=      DATE:           JULY 17, 1986                      =
;=      PURPOSE:        TO DEMONSTRATE INITIALIZATION      =
;=                      AND USAGE OF SERIAL PORT IN        =
;=                      POLLED MODE.                       =
;=      ASSEMBLER:      ZILOG ASMS8 ASSEMBLER              =
;=      PROGRAMMER:     CHARLES M. LINK, II                =
;===========================================================
;
;
;
        .PAGE   55      ;set maximum page size to 55 lines
;***********************************************************
;*                                                         *
;*                      GENERAL EQUATES                    *
;*                                                         *
;***********************************************************
;
CR:     .equ    0dH     ;carriage return
LF:     .equ    0aH     ;line feed
;
;
;***********************************************************
;*                                                         *
;*                    REGISTER EQUATE TABLE                *
;*                                                         *
;***********************************************************
;
period: .equ    0       ;period timer
second: .equ    1       ;seconds timer
minute: .equ    2       ;minutes timer
hours:  .equ    3       ;hours timer
;working register equates
MPTR:   .equ    RR8     ;message pointer for external memory
;
;***********************************************************
;*                                                         *
;*                    INTERRUPT VECTOR TABLE               *
;*                                                         *
;***********************************************************
;
INTR0:  .WORD   INTRET          ;this area should always be defined
INTR1:  .WORD   INTRET          ;as it reserves the lower 32 bytes
INTR2:  .WORD   INTRET          ;for the interrupt table.  the name
INTR3:  .WORD   INTRET          ;of the subroutine for each particular
INTR4:  .WORD   INTRET          ;interrupt service would normally be
INTR5:  .WORD   INTRET          ;named here.
INTR6:  .WORD   TIMER0
INTR7:  .WORD   INTRET
INTR8:  .WORD   INTRET
INTR9:  .WORD   INTRET
INTR10: .WORD   INTRET
INTR11: .WORD   INTRET
INTR12: .WORD   INTRET
```

```
INTR13: .WORD   INTRET
INTR14: .WORD   INTRET
INTR15: .WORD   INTRET
;
;***********************************************************
;*                                                         *
;*              START OF PROGRAM EXECUTION                 *
;*                                                         *
;***********************************************************
;
START:  jr      START1          ;program execution unconditionally
                                ;begins at this location after reset
                                ;and power up.
        .ASCII  'REL 0 7/17/86' ;jump around optional ascii string
                                ;containing release info, copyright, etc.
START1: di                      ;begin
        sb0                     ;select register bank 0
        ld      EMT,#00000000B  ;external memory timing=no wait input, normal
                                ;memory timing, no wait states, stack internal,
                                ;and DMA internal
        ld      P0,#00H         ;address begins at 0000h, set upper byte
        ld      POM,#11111111B  ;select all lines as address
        ld      PM,#00110000B   ;enable port 0 as upper 8 bits address
        ld      H1C,#00000000B  ;handshake not enabled port 0
;
;port 1 is defined in romless part as address/data.  it is not necessary
;here to initialize that port
;
        ld      P2,#00H         ;port 2 outputs low
        ld      P3,#00H         ;port 3 outputs low
        ld      P2AM,#10001010B ;p31,20,21 as output,p30 input
                                ;it is necessary here to configure p30 as input
                                ;for the receive data, and p31 as output for
                                ;transmit data for UART
        ld      P2BM,#10101010B ;p32,33,22,23 as output
        ld      P2CM,#10101010B ;p34,35,24,25 as output
        ld      P2DM,#10101010B ;p36,37,26,27 as output
;
        ld      P4,#00000000B   ;clear port 4 register
        ld      P4D,#11111111B  ;set all bits of P4 as inputs
        ld      P4OD,#00000000B ;active push/pull [not necessary since all
                                ; bits are inputs
;
;basic Super 8 I/O is initialized, now internal registers
;
        ld      RPO,#0COH       ;set working register low to lower 8 bytes
        ld      RP1,#0C8H       ;set working register high to upper 8 bytes
        ld      SPL,#0FFH       ;set stack pointer to start at top of set two
                                ;note here that only lower 8 bits are used
                                ;for stack pointer.  location 0FFH is wasted
                                ;as stack operation.  SPH is general purpose
                                ;storage.
;
;now clear the internal memory and stack area
;
        ld      SPH,#0FFH       ;point to top of general purpose register
ZERO:   clr     @SPH            ;zero it
        dec     SPH
        jr      nz,ZERO         ;do it until register set is all cleared
        clr     @SPH            ;zero last register
;
;now everything except working registers is cleared
;
;cpu and memory now initialized, set up timer for real time clock
;
        ld      SYM,#00000000B  ;disable fast interrupt response
        ld      IPR,#00000010B  ;interrupt priority
                                ;IRQ2>IRQ3>IRQ4>IRQ5>IRQ6>IRQ7>IRQ0>IRQ1
        ld      IMR,#00000100B  ;enable only interrupt 2
        sb1                     ;select bank 1
        ld      COTCH,#^HB(50000)       ;high byte of time constant
        ld      COTCL,#^LB(50000)       ;low byte of time constant
                                ;12,000,000 hertz / 4 / 50,000 = 60 hertz
                                ;12 Mhz is xtal freq, 4 is internal divider
        ld      COM,#00000100B  ;p27,37 is I/O, programmed up/down, no capture
                                ;timer mode is selected
        sb0                     ;select bank 0
        ld      COCT,#10100101B ;continuous, count down, load counter,
                                ;zero count interrupt enable, enable counter
;
;timer is set, now lets initialize the UART for polled operation
;
        sb1                     ;bank 1
        ld      UMA,#01110000B
                                ;time constant  = (12,000,000/4/16/9600/2)-1=
                                ;8.76 rounded to 9.
                                ;note that a 12 Mhz does not make a very
                                ;accurate baud rate source.  error is large
        ld      UBGH,#^HB(00009)        ;high byte of time constant
        ld      UBGL,#^LB(00009)        ;low byte of time constant
        ld      UMB,#00011110B  ;p21=p21data,auto-echo is off, transmit and
                                ;receive clock is baud rate generator output,
                                ;baud rate generator input is system clock / 2,
                                ;baud rate generator is enabled, loopback
                                ;is disabled
        sb0                     ;select bank 0
        ld      UTC,#10001000B  ;select p31 as transmit data out, 1 stop bit
                                ;and transmit enable
        ld      UIE,#00000000B  ;disable all interrupts, no DMA
        ld      URC,#00000010B  ;enable receive

;UART is initialized, enable interrupts for real time clock
;
        ei                      ;enable interrupts
;
;wait 1 full second for serial line to mark before sending anything
;
WAIT:   cp      second,#1       ;wait 1 second
        jr      ne,WAIT
;
;display the logon message
;
LOGON:  ldw     MPTR,#MSG       ;load the address of MSG into word reg MPTR
        call    SENDM           ;send the message
;
;logon message displayed, get response from console
;and move to upper register memory
;
GET:    ld      r1,#80          ;maximum character count
        ld      r2,#80H         ;point to first location in upper register bank
GETN:   call    GETC            ;get input from console
        and     r0,#7fH         ;remove upper parity bit
        call    SENDC           ;echo to console
        ld      @r2,r0          ;move to upper internal ram in Super8
        cp      r0,#CR          ;was the received character a carriage return
        jr      eq,ECHO         ;if so, echo it to console
        inc     r2              ;bump pointer
        djnz    r1,GETN         ;get next character if not done
;
;if carriage return typed, or 80 characters exceeded, echo message
;
ECHO:   ldw     MPTR,#MSG1      ;load the address of MSG1 in word reg MPTR
        call    SENDM           ;send the message
        ld      r1,#80          ;maximum character count
        ld      r2,#80H         ;first location of character buffer
ECHO1:  ld      r0,@r2          ;get character from buffer
        call    SENDC           ;send the character to console
        cp      r0,#CR          ;carriage return?
        jr      eq,LOGON        ;if so, end message display
        inc     r2              ;bump pointer
        djnz    r1,ECHO1        ;display next character if not done
        jr      LOGON
;
;subroutines
;
;send message at MPTR until '$' character found
SENDM:  ldci    r0,@MPTR        ;get the character
        call    SENDC           ;otherwise send character
        cp      r0,#'$'         ;last character?
        jr      ne,SENDM        ;and loop back to send next one
        ret
```

```
;send character in r0
SENDC:  tm      UTC,#00000010B  ;transmit buffer empty yet
        jr      z,SENDC         ;if not, wait until it is
        ld      UIO,r0          ;load the character into the transmitter
        ret
;get a character from the uart, return in r0
GETC:   tm      URC,#00000001B  ;character available
        jr      z,GETC          ;if not, wait until it is
        ld      r0,UIO          ;get the character from the receiver
        ret
;
;real time interrupt running in background
;
TIMER0: inc     period          ;bump periodic counter (60 hertz)
        cp      period,#60      ;one second yet?
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000001B   ;complement the second bit
        clr     period          ;start it over again
        inc     second          ;bump the seconds timer
        cp      second,#60      ;reached maximum
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000010B   ;complement the minute bit
        clr     second          ;start it over again
        inc     minute          ;bump the minutes timer
        cp      minute,#60      ;reached maximum
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000100B   ;complement the hour bit
        clr     minute          ;start it over again
        inc     hours           ;bump the hours timer
        cp      hours,#24       ;reached maximum
        jr      ne,NOROLL       ;no rollover
        clr     hours           ;start it over again
NOROLL: or      COCT,#00000010B ;reset end of count
        nop
        nop
INTRET: iret                    ;and return from interrupt
;
;
;
MSG:    .ASCII  CR,LF,'Super8 Uart test program.',CR,LF
        .ASCII  'Enter up to one full line followed by return',CR,LF,'$'
MSG1:   .ASCII  CR,LF,'Echoed back, your line was...',CR,LF,'$'


        .END
```

Zilog Application Note

August 1987

USING THE ZILOG SUPER8 IN INTERRUPT DRIVEN COMMUNICATIONS

by Charles M. Link, II

The power of the Super8 microcomputer lies in its on board peripherals. One of those peripherals is the full duplex UART. The UART can operate under program control in polled mode, or under interrupt control, and in a DMA mode. This article, the third in a series, discusses using the UART in a fully interrupt driven system. Since it is assumed that the reader has access to the eariler article discussing the UART and the polled mode of operation, this article will only discuss the differences.

The Zilog Super8 contains an on board interrupt controller that is tightly linked to the other on-board peripherals. The UART, being on-board, can be operated in an interrupt mode permitting very little execution overhead time while monitoring the UART for incomming characters and waiting for the UART to send outgoing characters.

Operation of an interrupt driven system demands more software logic to control the interrupt. Although more software is present, less time is spent executing it, because most of the overhead is in the setup for interrupt transfers. Generally, interrupt driven serial I/O overlaps some other process or processes, and therefore enhances total system speed and operation. Interrupt driven I/O has no advantages in a system that must wait on the serial port. In the example program, no real advantage has been gained by interrupt operation. The program displays a simple message to the console, and accepts input responses and echos them. For program simplicity, the main program waits on the interrupt to complete before starting the next phase of the program.

In any interrupt driven system, the central processor must know what to do when an interrupt occurs. The Super8 is no exeception. An interrupt vector table directs the processor to begin execution at certain addresses for particular interrupt inputs. The UART can be the source for up to five different interrupts and therefore up to five of the sixteen vectors can be designated for it. This sample program ignores errors and special condition interrupts, and therefore only two vectors are used; one for transmit buffer empty and one for receive character available. These vectors are programmed into the vector table by setting interrupt vector 10 (zero reference) to the address for the receive data service routine, and setting interrupt vector 13 to the address for the transmit data service routine.

The setup of the Super8 is essentially the same as that of the serial port in a polled mode of operation. The

proper priority for the interrupts are assigned arbitrarily. The real time clock as highest priority, the receive character available as second priority, and transmit character buffer empty as the lowest priority. Generally, the transmit interrupt should be the lowest in an asynchronous system because if it does not get serviced immediately, no major problems occur. If the real time interrupt took more time in relationship to the time required to transmit a single character, then maybe the receive should be put higher. If the receiver is not serviced, that character would be lost.

Enabling the interrupts is a two stage process. First the mask in the INTERRUPT MASK REGISTER must be enabled for each level of the interrupts used. Next, it is necessary to enable the individual transmit and receive interrupts. In the example program, a character is loaded into the transmit buffer and then the interrupt is enabled by setting bit 2 in the UART INTERRUPT ENABLE (UIE) register. Each successive transmit interrupt indicates an empty buffer, and the next character is loaded into the buffer. When the last character is loaded into the buffer, the transmit interrupt is disabled to prevent further interruptions by clearing bit 2 of the UIE register.

The receiver interrupt is enabled to allow the processor to accept incoming characters by setting bit 0 of the UIE register. Once set, any received character will cause the processor to transfer control to the "RXDATI" routine. In this example, the receive service routine reads, echos, and stores each received character until a carriage routine is received. The input is then repeated.

The example program does not fully utilize the interrupt system, as it waits for each routine to complete before moving to the next. However, it does however work, and demonstrates interrupt service routines. Serial interrupt software is not complex, and could lead to very powerful user programs. With the addition of the on board DMA to automaticly transfer characters, the Super8 can complete many tasks that previously would require complex hardware and software. The next article in the series demonstrates using the DMA controller with the serial port.

```
;
        .TITLE  Sample Zilog Super 8 Serial Interrupt Mode Operation
;
;
;===========================================================
;=      TITLE:          UART2.S                            =
;=      DATE:           JULY 17, 1986                      =
;=      PURPOSE:        TO DEMONSTRATE INTERRUPT           =
;=                      DRIVEN SERIAL PORT                 =
;=                      COMMUNICATIONS                     =
;=      ASSEMBLER:      ZILOG ASMS8 ASSEMBLER              =
;=      PROGRAMMER:     CHARLES M. LINK, II                =
;===========================================================
;
;
;
        .PAGE   55      ;set maximum page size to 55 lines
;**********************************************************
;*                                                        *
;*                    GENERAL EQUATES                     *
;*                                                        *
;**********************************************************
;
CR:     .equ    0dH     ;carriage return
LF:     .equ    0aH     ;line feed
;
;
;**********************************************************
;*                                                        *
;*                  REGISTER EQUATE TABLE                 *
;*                                                        *
;**********************************************************
;
period: .equ    0       ;period timer
second: .equ    1       ;seconds timer
minute: .equ    2       ;minutes timer
hours:  .equ    3       ;hours timer
;working register equates
MPTR:   .equ    RR8     ;message pointer for external memory
;
;**********************************************************
;*                                                        *
;*                  INTERRUPT VECTOR TABLE                *
;*                                                        *
;**********************************************************
;
INTR0:  .WORD   INTRET          ;this area should always be defined
INTR1:  .WORD   INTRET          ;as it reserves the lower 32 bytes
INTR2:  .WORD   INTRET          ;for the interrupt table.  the name
INTR3:  .WORD   INTRET          ;of the subroutine for each particular
INTR4:  .WORD   INTRET          ;interrupt service would normally be
INTR5:  .WORD   INTRET          ;named here.
INTR6:  .WORD   TIMER0
INTR7:  .WORD   INTRET
INTR8:  .WORD   INTRET
INTR9:  .WORD   INTRET
INTR10: .WORD   RXDATI
INTR11: .WORD   INTRET
INTR12: .WORD   INTRET
INTR13: .WORD   TXDATI
INTR14: .WORD   INTRET
INTR15: .WORD   INTRET
;
;**********************************************************
;*                                                        *
;*              START OF PROGRAM EXECUTION                *
;*                                                        *
;**********************************************************
;
START:  jr      START1          ;program execution unconditionally
                                ;begins at this location after reset
                                ;and power up.
        .ASCII  'REL 0 7/17/86' ;jump around optional ascii string
                                ;containing release info, copyright, etc.
START1: di                      ;begin
        sb0                     ;select register bank 0
        ld      EMT,#00000000B  ;external memory timing=no wait input, normal
                                ;memory timing, no wait states, stack internal,
                                ;and DMA internal
        ld      P0,#00H         ;address begins at 0000h, set upper byte
        ld      P0M,#11111111B  ;select all lines as address
        ld      PM,#00110000B   ;enable port 0 as upper 8 bits address
        ld      H1C,#00000000B  ;handshake not enabled port 0
;
;port 1 is defined in romless part as address/data.  it is not necessary
;here to initialize that port
;
        ld      P2,#00H         ;port 2 outputs low
        ld      P3,#00H         ;port 3 outputs low
        ld      P2AM,#10001010B ;p31,20,21 as output,p30 input
                                ;it is necessary here to configure p30 as input
                                ;for the receive data, and p31 as output for
                                ;transmit data for UART
        ld      P2BM,#10101010B ;p32,33,22,23 as output
        ld      P2CM,#10101010B ;p34,35,24,25 as output
        ld      P2DM,#10101010B ;p36,37,26,27 as output
;
        ld      P4,#00000000B   ;clear port 4 register
        ld      P4D,#11111111B  ;set all bits of P4 as inputs
        ld      P4OD,#00000000B ;active push/pull [not necessary since all
                                ; bits are inputs
;
;basic Super 8 I/O is initialized, now internal registers
;
        ld      RP0,#0C0H       ;set working register low to lower 8 bytes
        ld      RP1,#0C8H       ;set working register high to upper 8 bytes
        ld      SPL,#0FFH       ;set stack pointer to start at top of set two
                                ;note here that only lower 8 bits are used
                                ;for stack pointer.  location 0FFH is wasted
                                ;as stack operation.  SPH is general purpose
                                ;storage.
;
;now clear the internal memory and stack area
;
        ld      SPH,#0FFH       ;point to top of general purpose register
ZERO:   clr     @SPH            ;zero it
        dec     SPH
        jr      nz,ZERO         ;do it until register set is all cleared
        clr     @SPH            ;zero last register
;
;now everything except working registers is cleared
;
;cpu and memory now initialized, set up timer for real time clock
;
        ld      SYM,#00000000B  ;disable fast interrupt response
        ld      IPR,#00000010B  ;interrupt priority
                                ;IRQ2>IRQ3>IRQ4>IRQ5>IRQ6>IRQ7>IRQ0>IRQ1
        ld      IMR,#01000110B  ;enable counter, rx and tx interrupts
        sb1                     ;select bank 1
        ld      C0TCH,#^HB(50000)       ;high byte of time constant
        ld      C0TCL,#^LB(50000)       ;low byte of time constant
                                ;12,000,000 hertz / 4 / 50,000 = 60 hertz
                                ;12 Mhz is xtal freq, 4 is internal divider
        ld      C0M,#00000100B  ;p27,37 is I/O, programmed up/down, no capture
                                ;timer mode is selected
        sb0                     ;select bank 0
        ld      C0CT,#10100101B ;continuous, count down, load counter,
                                ;zero count interrupt enable, enable counter
;
;timer is set, now lets initialize the UART for polled operation
;
        sb1                     ;bank 1
        ld      UMA,#01110000B
                                ;time constant  = (12,000,000/4/16/9600/2)-1=
                                ;8.76 rounded to 9.
                                ;note that a 12 Mhz does not make a very
                                ;accurate baud rate source.  error is large
        ld      UBGH,#^HB(00009)        ;high byte of time constant
        ld      UBGL,#^LB(00009)        ;low byte of time constant
        ld      UMB,#00011110B  ;p21=p21data,auto-echo is off, transmit and
                                ;receive clock is baud rate generator output,
                                ;baud rate generator input is system clock / 2,
                                ;baud rate generator is enabled, loopback
                                ;is disabled
        sb0                     ;select bank 0
        ld      UTC,#10001000B  ;select p31 as transmit data out, 1 stop bit
                                ;and transmit enable
        ld      UIE,#00000000B  ;no interrupts, no DMA
        ld      URC,#00000010B  ;enable receive

;UART is initialized, enable interrupts for real time clock
;
        ei                      ;enable interrupts
;
;wait 1 full second of serial line mark before sending anything
;
WAIT:   cp      second,#1       ;wait 1 second
        jr      ne,WAIT
;
;display the logon message
;
LOGON:  ldw     MPTR,#MSG       ;load the address of MSG into word reg MPTR
        call    SENDM           ;send the message
        call    TXWAT           ;wait for transmitter to complete
;
;logon message displayed, get response from console
;and move to upper register memory
;
GET:    ld      r1,#80          ;maximum character count
        ld      r2,#80H         ;point to first location in upper register bank
        di                      ;stop interrupts
        or      UIE,#00000001B  ;receive character enable
        ei
;now wait for input to be completed
GW:     tm      UIE,#00000001B  ;wait for interrupt to be disabled
        jr      nz,GW           ;if interrupt still enabled

;
;if carriage return typed, or 80 characters exceeded, echo message
;
ECHO:   ldw     MPTR,#MSG1      ;load the address of MSG1 in word reg MPTR
        call    SENDM           ;send the message
;
;since messages are interrupt driven, we must wait for message to
;complete before transmitting next message
;
        call    TXWAT           ;wait on transmitter
        ld      r1,#80          ;maximum character count
        ld      r2,#80H         ;first location of character buffer
ECHO1:  ld      r0,@r2          ;get character from buffer
        call    SENDC           ;send the character to console
        cp      r0,#CR          ;carriage return?
        jr      eq,LOGON        ;if so, end message display
        inc     r2              ;bump pointer
        djnz    r1,ECHO1        ;display next character if not done
        jr      LOGON
;
;subroutines
;
;send message at MPTR until '$' character found
SENDM:  ldci    r0,@MPTR        ;get the character
        call    SENDC           ;start UART transmitting
        di                      ;no interrupts
        or      UIE,#00000100B  ;enable transmit interrupts
        ei
        ret
;send character in r0
SENDC:  tm      UTC,#00000010B  ;transmit buffer empty yet
        jr      z,SENDC         ;if not, wait until it is
        ld      UIO,r0          ;load the character into the transmitter
        ret
;transmit buffer available interrupt
TXDATI: ldci    r0,@MPTR        ;get next character to transmit
        ld      UIO,r0          ;load the character in transmitter
        cp      r0,#'$'         ;last character
        jr      eq,LASTT        ;if last transmit character
        iret
LASTT:  and     UIE,#11111011B  ;disable transmit interrupts
        iret                    ;ignore it if no character to transmit
;transmitter wait routine
TXWAT:  tm      UIE,#00000100B  ;wait until interrupts disabled
        jr      nz,TXWAT        ;wait if bit set
        ret
```

```
;receive character available interrupt
RXDATI: ld      r0,UIO          ;get input from console
        and     r0,#7fH         ;remove upper parity bit
        call    SENDC           ;echo to console
        ld      @r2,r0          ;move to upper internal ram in Super8
        cp      r0,#CR          ;was the received character a carriage return
        jr      eq,LASTR        ;if so, disable interrupts
        inc     r2              ;bump pointer
        djnz    r1,RXR          ;exit if not last
LASTR:  and     UIE,#11111110B  ;disable the receive interrupts
RXR:    iret
;
;real time interrupt running in background
;
TIMER0: inc     period          ;bump periodic counter (60 hertz)
        cp      period,#60      ;one second yet?
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000001B   ;complement the second bit
        clr     period          ;start it over again
        inc     second          ;bump the seconds timer
        cp      second,#60      ;reached maximum
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000010B   ;complement the minute bit
        clr     second          ;start it over again
        inc     minute          ;bump the minutes timer
        cp      minute,#60      ;reached maximum
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000100B   ;complement the hour bit
        clr     minute          ;start it over again
        inc     hours           ;bump the hours timer
        cp      hours,#24       ;reached maximum
        jr      ne,NOROLL       ;no rollover
        clr     hours           ;start it over again
NOROLL: or      COCT,#00000010B ;reset end of count
        nop
        nop
INTRET: iret                    ;and return from interrupt
;
;
;
MSG:    .ASCII  CR,LF,'Super8 Uart test program.',CR,LF
        .ASCII  'Enter up to one full line followed by return',CR,LF,'$'
MSG1:   .ASCII  CR,LF,'Echoed back, your line was...',CR,LF,'$'
.END
```

August 1987

USING THE SUPER8 SERIAL PORT WITH DMA

by Charles M. Link, II

With the increasing integration available today, microprocessor manufacturers are incorporating new peripherals that typically were off board in previous products, and sometimes required a large amount of external logic to utilize. The direct memory access function is a good example. Zilog has incorporated a very powerful DMA in the new Super8 microcontroller. It has the capability of linking to several on board peripherals, including the serial port, and can control data transfers to the different memory mediums.

The Super8, with its on-board DMA can reduce processor overhead in data transfer tasks. It allows direct transfer of serial input characters to either internal register memory (256 bytes) or external ram memory. For example, this transfer can be set to transfer a specific number of input characters, then interrupt the processor. Processor program service overhead is minimal. Serial output characters can be transfered from external EPROM or ram memory, or the internal register memory.

The required setup for the DMA transfers are much the same as that of interrupt or polled operation. This program example uses the DMA to interrupt upon termination of data transfers so that approopriate vectors and routines are required. Since the program links to the serial port, the DMA uses the serial port receive and transmit interrupt vectors 10 and 13, respectively. Upon completion of a receive DMA transfer, the service routine defined by the receive vector is executed. Upon completion of the transmit DMA transfer, service routine defined by the transmit vector is executed.

It is necessary to define the memory source/destination by setting the appropriate state of bit 0 in the EXTERNAL MEMORY TIMING (EMT) register. Initially, the example program selects external memory as the source/destination. A special note: read the fine print in the technical manual. Many hours were spent debugging the DMA mode of operation, with the final realization that internal rom does not qualify as external memory. Only that memory that would be selected if the /DM line was true would be a valid source/destination. Since this article uses the hardware defined from the first of the series, and uses a Z8800 with external EPROM, it will work perfectly. ROM and PIGGYBACK or prototype type parts will not work. Neither will emulators.

This sample uses the DMA mode to transmit a few lines of ASCII data to a console. The DMA requires a total byte count to properly transfer the data and terminate. Be careful to recognize that the ASCIL pseudo-op in the Zilog assembler, or many other assemblers, is not an easy way to generate the byte count. Warning! The Zilog assembler generates a length for each subgroup, e.g., "MSG" generates a separate length for each group separated by commas, not one total length.

Initially, the DMA transfers from EPROM. The address from which to transfer is C0 and C1 as defined by the working register pointers. It is necessary to set RP0 to C0 to access the register, and it is accessed as R0 and R1 or RR0. The count for the transfer is taken from DMA COUNT HIGH and DMA COUNT LOW. For each transfer, initialize the address and count values. Upon completion of the DMA transmit process, when the count goes to -1, a transmit interrupt is generated. The example program disables transmit interrupts and DMA, and returns. The main line program was polling the interrupt enable bit for completion.

Next, the DMA is set up to transfer 25 characters into the internal register memory. One must select internal memory in the EMT register by clearing bit 0. The address for transfer requires only one byte, so that working register 1 (R1), when RP0 equals C0, is the address pointer. The DMA count must also be loaded, in this case with 25. For demonstration purposes, the autoecho bit of the UART MODE B register is selected. This causes any characters received to be automatically looped back to the transmit port. Finally, the receive interrupt and DMA enable bits (BITS 0 and 1) are set to enable and begin DMA operation. When 25 characters have been input to the Super8, a receive interrupt will be generated, and control will be transfered to the "RXDATI" routine, where interrupts and DMA are disabled.

The last routine in the example software sends another message from EPROM to the console and then sends the characters from the internal memory buffer that were previously entered. The prime consideration is to remember to select the source/destination memory in the EMT register.

In this DMA example, the code is simple for DMA operation. It is important to note that this example does not fully utilize the functionality of the DMA transfer. The example purposely waits in a software loop while the DMA transfer occurs. This prevents the supporting code from becoming too complex to follow for an example. Normal operation might have the UART receiving characters under DMA controls and transmitting characters under interrupt control with processing occurring somewhere in the middle.

```
;
        .TITLE  Sample Zilog Super 8 Serial DMA Mode Operation
;
;
;============================================================
;=      TITLE:          UART3.S                             =
;=      DATE:           JULY 17, 1986                       =
;=      PURPOSE:        TO DEMONSTRATE DMA                  =
;=                      DRIVEN SERIAL PORT                  =
;=                      COMMUNICATIONS                      =
;=      ASSEMBLER:      ZILOG ASMS8 ASSEMBLER               =
;=      PROGRAMMER:     CHARLES M. LINK, II                 =
;============================================================
;
;
;
        .PAGE   55      ;set maximum page size to 55 lines
;************************************************************
;*                                                          *
;*                      GENERAL EQUATES                     *
;*                                                          *
;************************************************************
;
CR:     .equ    0dH     ;carriage return
LF:     .equ    0aH     ;line feed
;
;
;************************************************************
;*                                                          *
;*                    REGISTER EQUATE TABLE                 *
;*                                                          *
;************************************************************
;
period: .equ    0       ;period timer
second: .equ    1       ;seconds timer
minute: .equ    2       ;minutes timer
hours:  .equ    3       ;hours timer
;working register equates
MPTR:   .equ    RR0     ;message pointer for external memory
;
;************************************************************
;*                                                          *
;*                    INTERRUPT VECTOR TABLE                *
;*                                                          *
;************************************************************
;
INTR0:  .WORD   INTRET          ;this area should always be defined
INTR1:  .WORD   INTRET          ;as it reserves the lower 32 bytes
INTR2:  .WORD   INTRET          ;for the interrupt table.  the name
INTR3:  .WORD   INTRET          ;of the subroutine for each particular
INTR4:  .WORD   INTRET          ;interrupt service would normally be
INTR5:  .WORD   INTRET          ;named here.
INTR6:  .WORD   TIMER0
INTR7:  .WORD   INTRET
INTR8:  .WORD   INTRET
INTR9:  .WORD   INTRET
INTR10: .WORD   RXDATI
INTR11: .WORD   INTRET
INTR12: .WORD   INTRET
INTR13: .WORD   TXDATI
INTR14: .WORD   INTRET
INTR15: .WORD   INTRET
;
;************************************************************
;*                                                          *
;*                START OF PROGRAM EXECUTION                *
;*                                                          *
;************************************************************
;
START:  jr      START1          ;program execution unconditionally
```

```
                                ;begins at this location after reset
                                ;and power up.
        .ASCII  'REL 0 7/17/86' ;jump around optional ascii string
                                ;containing release info, copyright, etc.
START1: di                      ;begin
        sb0                     ;select register bank 0
        ld      EMT,#00000001B  ;external memory timing=no wait input, normal
                                ;memory timing, no wait states, stack internal,
                                ;and DMA external
        ld      P0,#00H         ;address begins at 0000h, set upper byte
        ld      P0M,#11111111B  ;select all lines as address
        ld      PM,#00110000B   ;enable port 0 as upper 8 bits address
        ld      H1C,#00000000B  ;handshake not enabled port 0
;
;port 1 is defined in romless part as address/data.  it is not necessary
;here to initialize that port
;
        ld      P2,#00H         ;port 2 outputs low
        ld      P3,#00H         ;port 3 outputs low
        ld      P2AM,#10001010B ;p31,20,21 as output,p30 input
                                ;it is necessary here to configure p30 as input
                                ;for the receive data, and p31 as output for
                                ;transmit data for UART
        ld      P2BM,#10101010B ;p32,33,22,23 as output
        ld      P2CM,#10101010B ;p34,35,24,25 as output
        ld      P2DM,#10101010B ;p36,37,26,27 as output
;
        ld      P4,#00000000B   ;clear port 4 register
        ld      P4D,#11111111B  ;set all bits of P4 as inputs
        ld      P4OD,#00000000B ;active push/pull [not necessary since all
                                ; bits are inputs
;
;basic Super 8 I/O is initialized, now internal registers
;
        ld      RP0,#0C0H       ;set working register low to lower 8 bytes
        ld      RP1,#0C8H       ;set working register high to upper 8 bytes
        ld      SPL,#0FFH       ;set stack pointer to start at top of set two
                                ;note here that only lower 8 bits are used
                                ;for stack pointer.  location 0FFH is wasted
                                ;as stack operation.  SPH is general purpose
                                ;storage.
;
;now clear the internal memory and stack area
;
        ld      SPH,#0FFH       ;point to top of general purpose register
ZERO:   clr     @SPH            ;zero it
        dec     SPH
        jr      nz,ZERO         ;do it until register set is all cleared
        clr     @SPH            ;zero last register
;
;now everything except working registers is cleared
;
;cpu and memory now initialized, set up timer for real time clock
;
        ld      SYM,#00000000B  ;disable fast interrupt response
        ld      IPR,#00000010B  ;interrupt priority
                                ;IRQ2>IRQ3>IRQ4>IRQ5>IRQ6>IRQ7>IRQ0>IRQ1
        ld      IMR,#01000110B  ;enable counter, rx and tx interrupts
        sb1                     ;select bank 1
        ld      C0TCH,#^HB(50000)       ;high byte of time constant
        ld      C0TCL,#^LB(50000)       ;low byte of time constant
                                ;12,000,000 hertz / 4 / 50,000 = 60 hertz
                                ;12 Mhz is xtal freq, 4 is internal divider
        ld      C0M,#00000100B  ;p27,37 is I/O, programmed up/down, no capture
                                ;timer mode is selected
        sb0                     ;select bank 0
        ld      C0CT,#10100101B ;continuous, count down, load counter,
                                ;zero count interrupt enable, enable counter
;
;timer is set, now lets initialize the UART for polled operation
;
        sb1                     ;bank 1
        ld      UMA,#01110000B
                                ;time constant  = (12,000,000/4/16/9600/2)-1=
                                ;8.76 rounded to 9.
                                ;note that a 12 Mhz does not make a very
                                ;accurate baud rate source.  error is large
        ld      UBGH,#^HB(00009)       ;high byte of time constant
        ld      UBGL,#^LB(00009)       ;low byte of time constant
        ld      UMB,#00011110B  ;p21=p21data,auto-echo is off, transmit and
                                ;receive clock is baud rate generator output,
                                ;baud rate generator input is system clock / 2,
                                ;baud rate generator is enabled, loopback
                                ;is disabled
        sb0                     ;select bank 0
        ld      UTC,#10001000B  ;select p31 as transmit data out, 1 stop bit
                                ;and transmit enable
        ld      UIE,#00000000B  ;no interrupts, no DMA
        ld      URC,#00000010B  ;enable receive

;UART is initialized, enable interrupts for real time clock
;
        ei                      ;enable interrupts
;
;because uart was just enabled, allow data line to mark for at least 1 second
;
WAIT:   cp      second,#1
        jr      ne,WAIT         ;wait 1 second
;
;display the logon message
;
LOGON:  ldw     MPTR,#MSG       ;load the address of MSG into word reg MPTR
        call    SENDM           ;send the message
        call    TXWAT           ;wait for transmitter to complete
;
;logon message displayed, get response from console
;and move to upper register memory
;
GET:    di                      ;no interrupts while setting up for DMA
        ldw     MPTR,#0080H     ;first character receive location
        and     EMT,#11111110B  ;select register file for receiving character
        sb1                     ;select bank one
        ld      DCH,#0          ;DMA count high byte
        ld      DCL,#25         ;DMA count low byte
        or      UMB,#00100000B  ;auto echo enable
        sb0                     ;restore to bank zero
        or      UIE,#00000011B  ;receive character DMA link, interrupt enable
        ei
        call    RXWAT           ;wait for receiver to complete receiving input
;
;receive characters in buffer, restore Super8 non DMA state
;
        di                      ;no interrupts while cleaning up
        sb1                     ;bank 1
        and     UMB,#11011111B  ;disable auto echo
        sb0                     ;restore bank 0
        or      EMT,#00000001B  ;select data memory for DMA transfers
        ei
;
;25 characters received via DMA, now display "ECHO" message
;
ECHO:   ldw     MPTR,#MSG1      ;load the address of MSG1 in word reg MPTR
        call    SENDM           ;send the message
        call    TXWAT           ;wait on transmitter
;
;message sent, now replay typed input
;
        di
        ldw     MPTR,#0080H     ;point to beginning of buffer
        and     EMT,#11111110B  ;select register bank for DMA transfer
        sb1                     ;select bank 1
        ld      DCH,#0          ;DMA count high byte
        ld      DCL,#25         ;DMA count low byte
        sb0                     ;select bank 0
        or      UIE,#00000100B  ;enable transmit interrupts
        or      UTC,#00000001B  ;transmit DMA enable
        ei                      ;enable interrupts
        call    TXWAT           ;wait on transmitter
        di
        or      EMT,#00000001B  ;select external data memory for DMA transfer
        ei
;
;replay complete, loop back and do it again
;
        jr      LOGON
```

```
;
;subroutines
;
;send message at MPTR for length in first byte
SENDM:  ldci    r7,@MPTR        ;get the character
        dec     r7              ;count actually should be n-1 for n bytes
        di                      ;no interrupts while setting up
        or      EMT,#00000001B  ;select external data memory for DMA transfer
        sb1                     ;select bank 1
        ld      DCH,#0          ;DMA count high byte is 0
        ld      DCL,r7          ;move the count DMA count low byte
        sb0                     ;select bank 0
        or      UIE,#00000100B  ;enable transmit interrupts
        or      UTC,#00000001B  ;transmit DMA enable
        ei
        ret
;transmit DMA complete
TXDATI: and     UIE,#11111011B  ;disable transmit interrupts
        and     UTC,#11111110B  ;disable transmit DMA
        iret                    ;ignore it if no character to transmit
;transmitter wait routine
TXWAT:  tm      UIE,#00000100B  ;wait until interrupts disabled
        jr      nz,TXWAT        ;wait if bit set
        ret
;receive character available interrupt
RXDATI: and     UIE,#11111100B  ;disable the receive interrupts
        iret
;receive wait routine
RXWAT:  tm      UIE,#00000001B  ;wait until interrupts disabled
        jr      nz,RXWAT        ;wait if bit still set
        ret
;
;real time interrupt running in background
;
TIMER0: inc     period          ;bump periodic counter (60 hertz)
        cp      period,#60      ;one second yet?
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000001B   ;complement the second bit
        clr     period          ;start it over again
        inc     second          ;bump the seconds timer
        cp      second,#60      ;reached maximum
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000010B   ;complement the minute bit
        clr     second          ;start it over again
        inc     minute          ;bump the minutes timer
        cp      minute,#60      ;reached maximum
        jr      ne,NOROLL       ;no rollover
        xor     P2,#00000100B   ;complement the hour bit
        clr     minute          ;start it over again
        inc     hours           ;bump the hours timer
        cp      hours,#24       ;reached maximum
        jr      ne,NOROLL       ;no rollover
        clr     hours           ;start it over again
NOROLL: or      COCT,#00000010B ;reset end of count
        nop
        nop
INTRET: iret                    ;and return from interrupt
;
;
;
MSG:    .BYTE   56
        .ASCII  CR,LF,'Super8 Uart DMA test program.',CR,LF
        .ASCII  'Enter 25 characters',CR,LF,'$'
MSG1:   .BYTE   34
        .ASCII  CR,LF,'Echoed back, your line was...',CR,LF,'$'


        .END
```

Zilog Application Note

August 1987

GENERATING SINE WAVES WITH THE ZILOG SUPER8

by Charles M. Link, II

Generally digital microprocessors are thought of as only being able to generate digital signals...that is either on or off. With the simple addition of a digital-to-analog converter (DAC), more complex waveforms may be generated. Since the advent of the microprocessor and the DAC, many methods have been used by hardware and software designers to generate sine waves, including some that involve precise instruction and clock cycle calculations. This example is different.

The Zilog Super8 microcomputer is a single chip device requiring only a latch and EPROM to operate in its ROMLESS state. Leaving 24 I/O lines for user configuration, it is extremely easy to interface with peripherals, including, in this case, the DAC- 08. The hardware in this application example is essentially the same base hardware as the previous application articles. Since it is assumed that the reader has access to those articles, detailed explaination of the base will not be made here. Only the additions to the base will be explained.

The base Super8 microprocessor has ports 2, 3 and 4 available for user connection. For this example, the DAC-08 is connected to port 4 (P4). The DAC-08 is tied, with the least significant bit tied to P40 and the most significant bit tied to P47. The other connections to the DAC-08 are mostly out of the test circuit description shown in the data manuals associated with it. The DAC requires -12 volts for proper operation. The output for this example is tied to a simple op- amp filter with a sharp roll off at about 3500 hertz. This type filter might be quite suitable for telecommunications applications, but may not be so good for many others. An oscilloscope displays the resultant waveform.

The software to operate the Super8 is in the original initialization software from eariler in this article series. Initialization is essentially the same. Port 4 must be set up as output, with active push-pull drivers. The main consideration for this program is the software "sample" rate. For this example, 8000 samples per second was chosen. Any other rate may be chosen, and the author has successfully used values up to 16000 samples per second without timing problems. Higher base clock rates are possible with the recently introducecd 20 megahertz Super8 chips available. With the sample method used, the sample rate does not vary with the different sine wave frequencies generated.

The sample method requires a sine wave table stored in ROM or EPROM. This example uses 256 values, although 64, 128 or more values are quite acceptable. The BASICA program that generated the sine table is included for user modification. Once the values were generated, they were manually typed into the program. Using the Zilog macro assembler would have signigicantly slowed assembling. Note that the comments in the BASICA program imust be removed before the PC can execute.

The values generated by the BASICA program are values ranging from 01H to 0FEH. Since the DAC represents 00H as zero volts and 0FFH as 5 volts, this table will product sine outputs from almost zero to almost five volts.

The principle of operation requires that a sixteen bit frequency increment be maintained. This increment is generated by the simple formula

FREQUENCY INCREMENT = (TABLESTEP X 256 X FREQUENCY) / SAMPLE

where FREQUENCY INCREMENT is a sixteen bit value saved in an increment register, TABLESTEP is the number of values in the sine wave table, FREQUENCY is the desired frequency of generation in hertz, and SAMPLE is the number of samples per second. In the example program, this increment is stored in "FINCR".

A current offset into the sine table is maintained in the register pair labeled "INCR". At each periodic interrupt, FINCR must be added to INCR and saved in INCR. This sixteen bit value remains the offset into the table. The upper byte of the offset is used to point to the value in the 256 byte sine table that is loaded into the DAC. In the sample program, the value loaded into the DAC is generated in the previous interrupt and saved until the first instruction of the next interrupt. This allows the interrupt to perform some other varying length transactions, without introducing bit jitter into the sine wave.

Changing the "FINCR" by program control causes different frequencies to be generated. In this case, the sine wave may be turned off by disabling the counter 0 interrupt. Depending upon the number of steps in the sine table and the sample frequency, very accurate sine frequencies may be generated. Calculate the actual error by using the following formula:

[ABS (REAL FREQI - INTEGER FREQI) / REAL FREQI] X 100 = % ERROR

where REAL FREQI is the actual calculated frequency increment, INTEGER FREQI is the nearest rounded integer of the calculated frequency increment, and the result is the actual percent error form the desired value.

With the addition of a filter with sharp cutoff just above the highest desired frequency, the Super8 serves quite well as a programmable sine wave generator. In addition to sine waves, complex waveforms may be easily generated by the Super8 with the addition of the low-cost DAC. The next article in this series will describe how to generate some of these more complex waveforms.

```
;
        .TITLE  Super8 Example Sine Wave Generation
;
;============================================================
;=      TITLE:          SINE.S                             =
;=      DATE:           JUNE 17, 1986                      =
;=      PURPOSE:        TO DEMONSTRATE USING SUPER8        =
;=                      TO GENERATE HIGH QUALITY SINE      =
;=                      WAVES.                             =
;=      HARDWARE:       DAC-08 ON PORT 4                   =
;=                      SEE DIAGRAM                        =
;=      ASSEMBLER:      ZILOG ASMS8 ASSEMBLER              =
;=      PROGRAMMER:     CHARLES M. LINK, II                =
;============================================================
;
;
;
        .PAGE   55      ;set maximum page size to 55 lines
;
;************************************************************
;*                                                          *
;*                   REGISTER EQUATE TABLE                  *
;*                                                          *
;************************************************************
;
INCR:   .equ    rr0             ;current increment in sine table
INCRH:  .equ    r0              ;high byte of current increment value
INCRL:  .equ    r1              ;low byte of current increment value
FINCR:  .equ    rr2             ;increment in sine table for frequency
FINCRH: .equ    r2              ;high byte of frequency increment value
FINCRL: .equ    r3              ;low byte of frequency increment value
POINT:  .equ    rr4             ;pointer into sine table
POINTH: .equ    r4              ;high byte of sine table pointer
POINTL: .equ    r5              ;low byte of sine table pointer
CVAL:   .equ    r6              ;current value to output to DAC-08
;
;************************************************************
;*                                                          *
;*                     GENERAL EQUATES                      *
;*                                                          *
;************************************************************
;
XTAL:   .equ    12000000                ;crystal freq in hertz
SAMPLE: .equ    8000                    ;sample frequency in hertz
CTVAL:  .equ    XTAL/4/SAMPLE           ;counter load value
TABSTP: .equ    256                     ;number of values in sine table
FREQ:   .equ    697                     ;desired sine wave frequency
FREQI:  .equ    (TABSTP*256*FREQ)/SAMPLE
;
;************************************************************
;*                                                          *
;*                  INTERRUPT VECTOR TABLE                  *
;*                                                          *
;************************************************************
;
INTR0:  .WORD   INTRET          ;this area should always be defined
INTR1:  .WORD   INTRET          ;as it reserves the lower 32 bytes
INTR2:  .WORD   INTRET          ;for the interrupt table.  the name
INTR3:  .WORD   INTRET          ;of the subroutine for each particular
INTR4:  .WORD   INTRET          ;interrupt service would normally be
INTR5:  .WORD   INTRET          ;named here.
INTR6:  .WORD   TIMER0
INTR7:  .WORD   INTRET
```

```
INTR8:  .WORD   INTRET
INTR9:  .WORD   INTRET
INTR10: .WORD   INTRET
INTR11: .WORD   INTRET
INTR12: .WORD   INTRET
INTR13: .WORD   INTRET
INTR14: .WORD   INTRET
INTR15: .WORD   INTRET
;
;***********************************************************
;*                                                         *
;*              START OF PROGRAM EXECUTION                 *
;*                                                         *
;***********************************************************
;
START:  jr      START1          ;program execution unconditionally
                                ;begins at this location after reset
                                ;and power up.
        .ASCII  'REL 0 6/16/86' ;jump around optional ascii string
                                ;containing release info, copyright, etc.
START1: di                      ;begin
        sb0                     ;select register bank 0
        ld      EMT,#00000000B  ;external memory timing=no wait input, normal
                                ;memory timing, no wait states, stack internal,
                                ;and DMA internal
        ld      P0,#00H         ;address begins at 0000h, set upper byte
        ld      P0M,#11111111B  ;select all lines as address
        ld      PM,#00110000B   ;enable port 0 as upper 8 bits address
        ld      H1C,#00000000B  ;handshake not enabled port 0
;
;port 1 is defined in romless part as address/data.  it is not necessary
;here to initialize that port
;
        ld      P2,#00H         ;port 2 outputs low
        ld      P3,#00H         ;port 3 outputs low
        ld      P2AM,#10101010B ;p30,31,20,21 as output
        ld      P2BM,#10101010B ;p32,33,22,23 as output
        ld      P2CM,#10101010B ;p34,35,24,25 as output
        ld      P2DM,#10101010B ;p36,37,26,27 as output
;
        ld      P4,#10000000B   ;set midpoint for DAC inputs
        ld      P4D,#00000000B  ;set all bits of P4 as output
        ld      P4OD,#00000000B ;active push/pull
;
;basic Super 8 I/O is initialized, now internal registers
;
        ld      RP0,#0C0H       ;set working register low to lower 8 bytes
        ld      RP1,#0C8H       ;set working register high to upper 8 bytes
        ld      SPL,#0FFH       ;set stack pointer to start at top of set two
                                ;note here that only lower 8 bits are used
                                ;for stack pointer.  location 0FFH is wasted
                                ;as stack operation.  SPH is general purpose
                                ;storage.
;
;now clear the internal memory and stack area
;
        ld      SPH,#0FFH       ;point to top of general purpose register
ZERO:   clr     @SPH            ;zero it
        dec     SPH
        jr      nz,ZERO         ;do it until register set is all cleared
        clr     @SPH            ;zero last register
;
;now everything except working registers is cleared
;
;cpu and memory now initialized, set up timer for real time clock
;
        ld      SYM,#00000000B  ;disable fast interrupt response
        ld      IPR,#00000010B  ;interrupt priority
                                ;IRQ2>IRQ3>IRQ4>IRQ5>IRQ6>IRQ7>IRQ0>IRQ1
        ld      IMR,#00000100B  ;enable only interrupt 2
        sb1                     ;select bank 1
        ld      C0TCH,#^HB(CTVAL)       ;high byte of time constant
        ld      C0TCL,#^LB(CTVAL)       ;low byte of time constant
        ld      C0M,#00000100B  ;p27,37 is I/O, programmed up/down, no capture
                                ;timer mode is selected
        sb0                     ;select bank 0
        ld      C0CT,#10100101B ;continuous, count down, load counter,
                                ;zero count interrupt enable, enable counter
;
;timer is initialized, now lets enable interrupts and wait
        ldw     INCR,#1         ;start at the beginning of sine table
        ldw     FINCR,#FREQI    ;load frequency of increment
        ldw     POINT,#SINTAB   ;pointer points to sine table
        ld      CVAL,#080H      ;initial value to prevent glitch at start
        ei                      ;enable interrupts
WAIT:   nop
        nop
        nop
        nop
        jr      WAIT            ;loop back
;
;Timer interrupt.  Occurs SAMPLE times per second
;interrupt outputs value to DAC-08 and then determines value for next
;interrupt.  This assures no bit jitter.
;
TIMER0: ld      p4,CVAL         ;write new value to DAC-08
        rcf                     ;clear carry flag
        add     INCRL,FINCRL    ;find next position in sine table
        adc     INCRH,FINCRH    ;by adding frequency offset to last position
        ld      POINTL,INCRH    ;set new pointer into sine table
                                ;upper byte ok since on boundary
        ldc     CVAL,@POINT     ;get value from sine table
        or      COCT,#00000010B ;reset end of count interrupt
INTRET: iret                    ;and return from interrupt
;
;************************************************************
;*                                                          *
;*                      SINE WAVE LOOKUP                    *
;*                                                          *
;************************************************************
;
;sine table for sine wave generation using DAC-08.  Table based upon
;case of waveform with minumum amplititude = 0 volts and maximum
;amplititude = 5 volts.  DAC-08 input for 0 volts = 00H
;5 volts = 0FFH.  Table generated using following BASICA program,
;then typed into program.
;
;       10 CLS                  ;clear screen
;       20 PI=3.141593          ;define PI
;       30 FOR I=0 TO 255       ;256 total values
;       40 C=360/256            ;define basic interval value
;       50 D=C*I                ;value from zero on sine wave
;       60 E=D*PI/180
;       70 F=SIN(E)             ;figure sine for interval from 0
;       80 G=F*127              ;sine range should be from -127 to 127
;       90 H=128+G              ;make result from 0 to 255
;       100 J=CINT(H)           ;round to nearest integer
;       110 A$=HEX$(J)          ;convert to hex
;       120 PRINT A$            ;on screen
;       130 LPRINT A$           ;on printer
;       140 NEXT                ;do next inverval
;       150 END
;
;*note-remove comments, BASICA will not accept ; as comment delimiter
;
;
SINTAB: .ORG    0400H           ;begin sine table on even byte boundary
        .byte   080H,083H,086H,089H,08CH,090H,093H,096H,099H,09CH,09FH,0A2H
        .byte   0A5H,0A8H,0ABH,0AEH,0B1H,0B3H,0B6H,0B9H,0BCH,0BFH,0C1H,0C4H
        .byte   0C7H,0C9H,0CCH,0CEH,0D1H,0D3H,0D5H,0D8H,0DAH,0DCH,0DEH,0E0H
        .byte   0E2H,0E4H,0E6H,0E8H,0EAH,0EBH,0EDH,0EFH,0F0H,0F1H,0F3H,0F4H
        .byte   0F5H,0F6H,0F8H,0F9H,0FAH,0FAH,0FBH,0FCH,0FDH,0FDH,0FEH,0FEH
        .byte   0FEH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FEH,0FEH,0FEH,0FDH
        .byte   0FDH,0FCH,0FBH,0FAH,0FAH,0F9H,0F8H,0F6H,0F5H,0F4H,0F3H,0F1H
        .byte   0F0H,0EFH,0EDH,0EBH,0EAH,0E8H,0E6H,0E4H,0E2H,0E0H,0DEH,0DCH
        .byte   0DAH,0D8H,0D5H,0D3H,0D1H,0CEH,0CCH,0C9H,0C7H,0C4H,0C1H,0BFH
        .byte   0BCH,0B9H,0B6H,0B3H,0B1H,0AEH,0ABH,0A8H,0A5H,0A2H,09FH,09CH
```

```
.byte   099H,096H,093H,090H,08CH,089H,086H,083H,080H,07DH,07AH,077H
.byte   074H,070H,06DH,06AH,067H,064H,061H,05EH,05BH,058H,055H,052H
.byte   04FH,04DH,04AH,047H,044H,041H,03FH,03CH,039H,037H,034H,032H
.byte   02FH,02DH,02BH,028H,026H,024H,022H,020H,01EH,01CH,01AH,018H
.byte   016H,015H,013H,011H,010H,00FH,00DH,00CH,00BH,00AH,008H,007H
.byte   006H,006H,005H,004H,003H,003H,002H,002H,002H,001H,001H,001H
.byte   001H,001H,001H,001H,002H,002H,002H,003H,003H,004H,005H,006H
.byte   006H,007H,008H,00AH,00BH,00CH,00DH,00FH,010H,011H,013H,015H
.byte   016H,018H,01AH,01CH,01EH,020H,022H,024H,026H,028H,02BH,02DH
.byte   02FH,032H,034H,037H,039H,03CH,03FH,041H,044H,047H,04AH,04DH
.byte   04FH,052H,055H,058H,05BH,05EH,061H,064H,067H,06AH,06DH,070H
.byte   074H,077H,07AH,07DH

.END
```

August 1987

GENERATING DTMF TONES WITH THE ZILOG SUPER8

by Charles M. Link, II

In the previous article, a sine wave generation example was demonstrated. Sine waves are great, but, sometimes, more complex waveforms must be generated. One of the most widely used complex waveforms is the DTMF tone. The DTMF tone is used on millions of telephones under the AT&T registered name "TOUCH TONE". Generally, telecommunications designers purchase one of the many DTMF encoder chips and hang it beside a microprocessor. This application article contains an example of a DTMF generation scheme that produces nearly as pure and probably as accurate a tone as the external chip method.

Generating sine waves requires some type of digital-to-analog converter to interface to the microprocessor. For this application, a DAC-08 is used. This DAC-08 is tied to port 4 of the Super8. Since it is assumed that the reader has access to the previous article, a detailed description of the hardware will be left to that article. Why not use the DTMF generator chip, when it might be just as inexpensive as the DAC- 08? The answer is that the DTMF generator chip requires an external crystal or clock, and it might not be convenient to pick a processor frequency that is a direct multiple of the one required by the generator. The second and more important reason is that the DAC-08 can be used to generate other call progress tones such as ringback and busy, or any other complex waveform.

Since the previous article discussed the method for generating sine wave tones, this article will only discuss how to turn that into the DTMF tone. The DTMF tone is actually a combination of two tones, hence, the name DUAL TONE MULTI-FREQUENCY. The tones are arranged such that each row and each column has a corresponding single frequency tone assigned. An additional, normally unseen column, contains an eighth tone frequency. A simple diagram below shows the arrangement.

DTMF TONE ASSIGNMENT

	1209	1336	1477	1633
697	1	2	3	A
770	4	5	6	B
852	7	8	9	C
941	*	0	#	D

The method used to combine the two tones into one single complex waveform is simple: add the two individual tones together. Adding the tones together is usually what happens when analog circuitry produces the DTMF tone. In fact, most of the DTMF encoder chips usually add the tones together either internally or externally to produce the single waveform.

Generating the two tones is no task for the Super8 microcomputer. Just set up two current table offset values and two different frequency increments. At each periodic interrupt the 16 bit frequency increment is added to the current table offset producing a new current table offset. The upper byte of each current table offset (one for the row frequency and one for the column) is used as a pointer into a 256 byte table. The sine values retrieved from the table are then added together and loaded into the DAC-08.

Since the DAC input of 00H corresponds to an output of 0 volts and the input of 0FFH corresponds to an output of 5 volts, adding two values that could possibly be 0FFH presents a problem. Since two sines must add to no more 5 volts, the maximum for one single sine value must be one half of 5 volts, or 80H. The sine table has been adjusted so that the 2.5 volt value is mid-range. The maximum or mimumum for the sine wave is plus or minus 1.25 volts.

The interrupt service routine is almost exactly the same as the interrupt routine for the sine wave, except that two sine waves are calculated. The final values are added together and stored for the first instruction of the next interrupt. In order to change tones, or disable the tone generation, additional software logic could enable or disable the interrupt, and modify the two values "CINCR", and "RINCR".

It is clear from the example, that ringback, busy, MF, and other signaling tones can be easily generated without additional hardware. Increased sampling rates could be used to generate tones of much higher frequencies and accuracies. The accuracy, using the above method and sampling frequencies, is much less than one percent, totally suitable for telecommunications needs.

```
;
        .TITLE  Super8 Example DTMF Generation
;
;===========================================================
;=      TITLE:          DTMF.S                             =
;=      DATE:           JUNE 17, 1986                      =
;=      PURPOSE:        TO DEMONSTRATE USING SUPER8        =
;=                      TO GENERATE HIGH QUALITY DTMF      =
;=                      WAVES.                             =
;=      HARDWARE:       DAC-08 ON PORT 4                   =
;=                      SEE DIAGRAM                        =
;=      ASSEMBLER:      ZILOG ASMS8 ASSEMBLER              =
;=      PROGRAMMER:     CHARLES M. LINK, II                =
;===========================================================
;
;
;
        .PAGE   55      ;set maximum page size to 55 lines
;
;***********************************************************
;*                                                         *
;*                 REGISTER EQUATE TABLE                   *
;*                                                         *
;***********************************************************
;
;column tone equates
CINCR:  .equ    rr0             ;current increment in sine table
CINCRH: .equ    r0              ;high byte of current increment value
CINCRL: .equ    r1              ;low byte of current increment value
CFINCR: .equ    rr2             ;increment in sine table for frequency
CFINCH: .equ    r2              ;high byte of frequency increment value
CFINCL: .equ    r3              ;low byte of frequency increment value
POINT:  .equ    rr4             ;pointer into sine table
POINTH: .equ    r4              ;high byte of sine table pointer
POINTL: .equ    r5              ;low byte of sine table pointer
;row tone equates
RINCR:  .equ    rr6             ;current increment in sine table
RINCRH: .equ    r6              ;high byte of current increment value
RINCRL: .equ    r7              ;low byte of current increment value
RFINCR: .equ    rr8             ;increment in sine table for frequency
RFINCH: .equ    r8              ;high byte of frequency increment value
RFINCL: .equ    r9              ;low byte of frequency increment value
CVAL:   .equ    r10             ;current value to output to DAC-08
RVAL:   .equ    r11             ;current row value
;
;***********************************************************
;*                                                         *
;*                   GENERAL EQUATES                       *
;*                                                         *
;***********************************************************
;
XTAL:   .equ    12000000                ;crystal freq in hertz
SAMPLE: .equ    8000                    ;sample frequency in hertz
CTVAL:  .equ    XTAL/4/SAMPLE           ;counter load value
TABSTP: .equ    256                     ;number of values in sine table
CFREQ:  .equ    1209                    ;desired column frequency
RFREQ:  .equ    697                     ;desired row frequency
CFREQI: .equ    (TABSTP*256*CFREQ)/SAMPLE
RFREQI: .equ    (TABSTP*256*RFREQ)/SAMPLE
;note dtmf frequencies are 697,770,852,941,1209,1336,1477,1633
;
;***********************************************************
;*                                                         *
;*                 INTERRUPT VECTOR TABLE                  *
;*                                                         *
;***********************************************************
;
INTR0:  .WORD   INTRET          ;this area should always be defined
INTR1:  .WORD   INTRET          ;as it reserves the lower 32 bytes
INTR2:  .WORD   INTRET          ;for the interrupt table.  the name
INTR3:  .WORD   INTRET          ;of the subroutine for each particular
INTR4:  .WORD   INTRET          ;interrupt service would normally be
INTR5:  .WORD   INTRET          ;named here.
INTR6:  .WORD   TIMER0
INTR7:  .WORD   INTRET
INTR8:  .WORD   INTRET
INTR9:  .WORD   INTRET
INTR10: .WORD   INTRET
```

```
INTR11: .WORD   INTRET
INTR12: .WORD   INTRET
INTR13: .WORD   INTRET
INTR14: .WORD   INTRET
INTR15: .WORD   INTRET
;
;***********************************************************
;*                                                         *
;*              START OF PROGRAM EXECUTION                 *
;*                                                         *
;***********************************************************
;
START:  jr      START1          ;program execution unconditionally
                                ;begins at this location after reset
                                ;and power up.
        .ASCII  'REL 0 6/16/86' ;jump around optional ascii string
                                ;containing release info, copyright, etc.
START1: di                      ;begin
        sb0                     ;select register bank 0
        ld      EMT,#00000000B  ;external memory timing=no wait input, normal
                                ;memory timing, no wait states, stack internal,
                                ;and DMA internal
        ld      P0,#00H         ;address begins at 0000h, set upper byte
        ld      P0M,#11111111B  ;select all lines as address
        ld      PM,#00110000B   ;enable port 0 as upper 8 bits address
        ld      H1C,#00000000B  ;handshake not enabled port 0
;
;port 1 is defined in romless part as address/data.  it is not necessary
;here to initialize that port
;
        ld     P2,#00H          ;port 2 outputs low
        ld     P3,#00H          ;port 3 outputs low
        ld     P2AM,#10101010B ;p30,31,20,21 as output
        ld     P2BM,#10101010B ;p32,33,22,23 as output
        ld     P2CM,#10101010B ;p34,35,24,25 as output
        ld     P2DM,#10101010B ;p36,37,26,27 as output
;
        ld     P4,#10000000B    ;set midpoint for DAC inputs
        ld     P4D,#00000000B   ;set all bits of P4 as output
        ld     P4OD,#00000000B ;active push/pull
;
;basic Super 8 I/O is initialized, now internal registers
;
        ld     RP0,#0C0H        ;set working register low to lower 8 bytes
        ld     RP1,#0C8H        ;set working register high to upper 8 bytes
        ld     SPL,#0FFH        ;set stack pointer to start at top of set two
                                ;note here that only lower 8 bits are used
                                ;for stack pointer.  location 0FFH is wasted
                                ;as stack operation.  SPH is general purpose
                                ;storage.
;
;now clear the internal memory and stack area
;
        ld     SPH,#0FFH        ;point to top of general purpose register
ZERO:   clr    @SPH             ;zero it
        dec    SPH
        jr     nz,ZERO          ;do it until register set is all cleared
        clr    @SPH             ;zero last register
;
;now everything except working registers is cleared
;
;cpu and memory now initialized, set up timer for real time clock
;
        ld     SYM,#00000000B  ;disable fast interrupt response
        ld     IPR,#00000010B  ;interrupt priority
                               ;IRQ2>IRQ3>IRQ4>IRQ5>IRQ6>IRQ7>IRQ0>IRQ1
        ld     IMR,#00000100B  ;enable only interrupt 2
        sb1                    ;select bank 1
        ld     C0TCH,#^HB(CTVAL)       ;high byte of time constant
        ld     C0TCL,#^LB(CTVAL)       ;low byte of time constant
        ld     C0M,#00000100B  ;p27,37 is I/O, programmed up/down, no capture
                               ;timer mode is selected
        sb0                    ;select bank 0
        ld     C0CT,#10100101B ;continuous, count down, load counter,
                               ;zero count interrupt enable, enable counter
;
;timer is initialized, now lets enable interrupts and wait
        ldw     CINCR,#1           ;start column at beginning of sine table
        ldw     RINCR,#1           ;start row at beginning of sine table
```

```
;
;this example loads the tones for digit '1'
;user software would, of course have to manipulate these registers for
;proper tone control
;
        ldw     CFINCR,#CFREQI  ;load column frequency increment
        ldw     RFINCR,#RFREQI  ;load row frequency increment
        ldw     POINT,#SINTAB   ;pointer points to sine table
        ld      CVAL,#080H      ;initial value to prevent glitch at start
        ei                      ;enable interrupts
WAIT:   nop
        nop
        nop
        nop
        jr      WAIT            ;loop back
;
;Timer interrupt.  Occurs SAMPLE times per second
;interrupt outputs value to DAC-08 and then determines value for next
;interrupt.  This assures no bit jitter.
;
TIMER0: ld      p4,CVAL         ;write new value to DAC-08
        rcf                     ;clear carry flag
        add     CINCRL,CFINCL   ;find next position in sine table
        adc     CINCRH,CFINCH   ;by adding frequency offset to last position
        ld      POINTL,CINCRH   ;set new pointer into sine table
        ldc     CVAL,@POINT     ;get value from sine table
        add     RINCRL,RFINCL   ;find next position in sine table
        adc     RINCRH,RFINCH   ;by adding frequencty offset to last position
        ld      POINTL,RINCRH   ;set new pointer into sine table
        ldc     RVAL,@POINT     ;get second value from sine table
        add     CVAL,RVAL       ;form a complex waveform from two sine values
        or      COCT,#00000010B ;reset end of count interrupt
INTRET: iret                    ;and return from interrupt
;
;*************************************************************
;*                                                           *
;*                      SINE WAVE LOOKUP                     *
;*                                                           *
;*************************************************************
;
;sine table for DTMF generation using DAC-08.  Table based upon
;case of waveform consisting of two sine waves summed to provide a single
;complex waveform with minumum amplititude = 0 volts and maximum
;amplititude = 5 volts.  DAC-08 input for 0 volts = 00H
;5 volts = 0FFH.  Both waves must total no more than 0FFH, therefore
;maximum for one wave must be 1/2 5 volts or 080H.
;Table generated using following BASICA program,
;then typed into program.
;
;       10 CLS                  ;clear screen
;       20 PI=3.141593          ;define PI
;       30 FOR I=0 TO 255       ;256 total values
;       40 C=360/256            ;define basic interval value
;       50 D=C*I                ;value from zero on sine wave
;       60 E=D*PI/180
;       70 F=SIN(E)             ;figure sine for interval from 0
;       80 G=F*63               ;sine range should be from -63 to 63
;       90 H=64+G               ;make result from 0 to 127
;       100 J=CINT(H)           ;round to nearest integer
;       110 A$=HEX$(J)          ;convert to hex
;       120 PRINT A$            ;on screen
;       130 LPRINT A$           ;on printer
;       140 NEXT                ;do next inverval
;       150 END
;
;*note-remove comments, BASICA will not accept ; as comment delimiter
;
;
SINTAB: .ORG    0400H           ;begin sine table on even byte boundary
        .byte   040H,042H,043H,045H,046H,048H,049H,04BH,04CH,04EH,04FH,051H
        .byte   052H,054H,055H,057H,058H,05AH,05BH,05CH,05EH,05FH,060H,062H
        .byte   063H,064H,066H,067H,068H,069H,06AH,06BH,06DH,06EH,06FH,070H
        .byte   071H,072H,073H,074H,074H,075H,076H,077H,078H,078H,079H,07AH
        .byte   07AH,07BH,07BH,07CH,07CH,07DH,07DH,07DH,07EH,07EH,07EH,07FH
        .byte   07FH,07FH,07FH,07FH,07FH,07FH,07FH,07FH,07FH,07FH,07EH,07EH
        .byte   07EH,07DH,07DH,07DH,07CH,07CH,07BH,07BH,07AH,07AH,079H,078H
        .byte   078H,077H,076H,075H,074H,074H,073H,072H,071H,070H,06FH,06EH
        .byte   06DH,06BH,06AH,069H,068H,067H,066H,064H,063H,062H,060H,05FH
        .byte   05EH,05CH,05BH,05AH,058H,057H,055H,054H,052H,051H,04FH,04EH
        .byte   04CH,04BH,049H,048H,046H,045H,043H,042H,040H,03EH,03DH,03BH
        .byte   03AH,038H,037H,035H,034H,032H,031H,02FH,02EH,02CH,02BH,029H
        .byte   028H,026H,025H,024H,022H,021H,020H,01EH,01DH,01CH,01AH,019H
        .byte   018H,017H,016H,015H,013H,012H,011H,010H,00FH,00EH,00DH,00CH
        .byte   00CH,00BH,00AH,009H,008H,008H,007H,006H,006H,005H,005H,004H
        .byte   004H,003H,003H,003H,002H,002H,002H,001H,001H,001H,001H,001H
        .byte   001H,001H,001H,001H,001H,001H,002H,002H,002H,003H,003H,003H
        .byte   004H,004H,005H,005H,006H,006H,007H,008H,008H,009H,00AH,00BH
        .byte   00CH,00CH,00DH,00EH,00FH,010H,011H,012H,013H,015H,016H,017H
        .byte   018H,019H,01AH,01CH,01DH,01EH,020H,021H,022H,024H,025H,026H
        .byte   028H,029H,02BH,02CH,02EH,02FH,031H,032H,034H,035H,037H,038H
        .byte   03AH,03BH,03DH,03EH
        .END
```

Zilog Application Note

August 1987

A SIMPLE SERIAL TO PARALLEL CONVERTER USING THE ZILOG SUPER8

by Charles M. Link, II

The Zilog Super8 has many on-board peripherals that provide multiple user applications. Earlier articles have demonstrated simple application "stubs" or short test programs. This article and the next article demonstrate a useful application for the Super8. Although it underutilizes the Super8's power, the simple serial to parallel converter in this application and the print buffer in the next application demonstrate the ease at which applications are developed with the Super8.

The Zilog Super8 has several features that enhance its use as a communication controller. The interrupt or DMA driven serial port are helpful, but the handshaking parallel ports finish the job. In the serial to parallel converter, the 256 byte internal register memory is used as a small circular queue.

Hardware for this application is fairly simple. Port 4 is buffered and hooked to the data lines, as shown, to interface to a centronics type printer connector. The strobe from P25 provides the strobe (pin 1) to the printer. The acknowledge line from the printer is inverted and tied to P24 of the Super8. The busy signal from the printer is buffered and tied to P23 of the Super8. The design was tested on an Okidata printer and is not guaranteed to work on all printers.

Software is fairly straightforward. The serial port is initialized just like it was in the application article on the interrupt driven serial port. Port 4 must be set-up as outputs with active push-pull drivers. Port 2, bits 3 and 4, are set up as input with P24 set to enable interrupts. P25 is set as output and handshake 0 is set in H0C to provide a strobe of 16 clock periods in length.

```
;
        .TITLE  Sample Zilog Super 8 Serial to Parallel Converter
;
;
;=============================================================
;=      TITLE:          SERPAR.S                             =
;=      DATE:           JULY 17, 1986                        =
;=      PURPOSE:        TO DEMONSTRATE INTERRUPT             =
;=                      DRIVEN SERIAL PORT IN A              =
;=                      REALISTIC APPLICATION.               =
;=                      THIS APPLICATION RECEIVES            =
;=                      SIMPLE SERIAL DATA A SENDS IT        =
;=                      OUT THE PARALLEL PORT TO A           =
;=                      PRINTER.                             =
;=      ASSEMBLER:      ZILOG ASMS8 ASSEMBLER                =
;=      PROGRAMMER:     CHARLES M. LINK, II                  =
;=============================================================
;
;
;
        .PAGE   55      ;set maximum page size to 55 lines
;************************************************************
;*                                                          *
;*                   GENERAL EQUATES                        *
;*                                                          *
;************************************************************
;
CR:     .equ    0dH     ;carriage return
LF:     .equ    0aH     ;line feed
;
;
;************************************************************
;*                                                          *
;*                REGISTER EQUATE TABLE                     *
;*                                                          *
;************************************************************
;
;working register equates
INPNT:  .equ    R3      ;input character pointer
OUTPNT: .equ    R4      ;output character pointer
```

```
MPTR:   .equ    RR6     ;message pointer for external memory
ACKB:   .equ    R5      ;byte containing acknowledge bit
ACKBIT: .equ    0       ;bit set = no acknowledge yet
                        ;bit clear = not waiting on acknowledge
;
;***********************************************************
;*                                                         *
;*                  INTERRUPT VECTOR TABLE                 *
;*                                                         *
;***********************************************************
;
INTR0:  .WORD   INTRET          ;this area should always be defined
INTR1:  .WORD   INTRET          ;as it reserves the lower 32 bytes
INTR2:  .WORD   INTRET          ;for the interrupt table.  the name
INTR3:  .WORD   INTRET          ;of the subroutine for each particular
INTR4:  .WORD   INTRET          ;interrupt service would normally be
INTR5:  .WORD   INTRET          ;named here.
INTR6:  .WORD   INTRET
INTR7:  .WORD   INTRET
INTR8:  .WORD   INTRET
INTR9:  .WORD   INTRET
INTR10: .WORD   RXDATI          ;receive data interrupt
INTR11: .WORD   INTRET
INTR12: .WORD   INTRET
INTR13: .WORD   INTRET
INTR14: .WORD   ACKSTB          ;acknowledge strobe interrupt
INTR15: .WORD   INTRET
;
;***********************************************************
;*                                                         *
;*              START OF PROGRAM EXECUTION                 *
;*                                                         *
;***********************************************************
;
START:  jr      START1          ;program execution unconditionally
                                ;begins at this location after reset
                                ;and power up.
        .ASCII  'REL 0 7/17/86' ;jump around optional ascii string
                                ;containing release info, copyright, etc.
START1: di                      ;begin
        sb0                     ;select register bank 0
        ld      EMT,#00000000B  ;external memory timing=no wait input, normal
                                ;memory timing, no wait states, stack internal,
                                ;and DMA internal
        ld      P0,#00H         ;address begins at 0000h, set upper byte
        ld      P0M,#11111111B  ;select all lines as address
        ld      PM,#00110000B   ;enable port 0 as upper 8 bits address
        ld      H1C,#00000000B  ;handshake not enabled port 0
;
;port 1 is defined in romless part as address/data.  it is not necessary
;here to initialize that port
;
        ld      P2,#00100000B   ;port 2 outputs low, except strobe bit
        ld      P3,#00H         ;port 3 outputs low
        ld      P2AM,#10001010B ;p31,20,21 as output,p30 input
                                ;it is necessary here to configure p30 as input
                                ;for the receive data, and p31 as output for
                                ;transmit data for UART
        ld      P2BM,#10100010B ;p32,33,22 as output, 23 as input
        ld      P2CM,#10101001B ;p34,35,25 as output, 24 as input, interrupt en
        ld      P2DM,#10101010B ;p36,37,26,27 as output
;
        ld      P4,#00000000B   ;clear port 4 register
        ld      P4D,#00000000B  ;set all bits of P4 as outputs
        ld      P4OD,#00000000B ;active push/pull
        ld      H0C,#11110001B  ;handshake enable for port 4, 16 clock pulse
;
;basic Super 8 I/O is initialized, now internal registers
;
        ld      RP0,#0C0H       ;set working register low to lower 8 bytes
        ld      RP1,#0C8H       ;set working register high to upper 8 bytes
        ld      SPL,#0FFH       ;set stack pointer to start at top of set two
                                ;note here that only lower 8 bits are used
                                ;for stack pointer.  location 0FFH is wasted
                                ;as stack operation.  SPH is general purpose
                                ;storage.
;
;now clear the internal memory and stack area
```

```
;
        ld      SPH,#0FFH       ;point to top of general purpose register
ZERO:   clr     @SPH            ;zero it
        dec     SPH
        jr      nz,ZERO         ;do it until register set is all cleared
        clr     @SPH            ;zero last register
;
;now everything except working registers is cleared
;
;cpu and memory now initialized, set up timer for real time clock
;
        ld      SYM,#00000000B  ;disable fast interrupt response
        ld      IPR,#10111111B  ;interrupt priority
                                ;IRQ6>IRQ7>IRQ5>IRQ4>IRQ3>IRQ2>IRQ1>IRQ0
        ld      IMR,#01010000B  ;rx interrupts, acknowledge strobe
;
;timer is set, now lets initialize the UART for polled operation
;
        sb1                     ;bank 1
        ld      UMA,#01110000B
                                ;time constant  = (12,000,000/4/16/9600/2)-1=
                                ;8.76 rounded to 9.
                                ;note that a 12 Mhz does not make a very
                                ;accurate baud rate source.  error is large
        ld      UBGH,#^HB(00009)        ;high byte of time constant
        ld      UBGL,#^LB(00009)        ;low byte of time constant
        ld      UMB,#00011110B  ;p21=p21data,auto-echo is off, transmit and
                                ;receive clock is baud rate generator output,
                                ;baud rate generator input is system clock / 2,
                                ;baud rate generator is enabled, loopback
                                ;is disabled
        sb0                     ;select bank 0
        ld      UTC,#10001000B  ;select p31 as transmit data out, 1 stop bit
                                ;and transmit enable
        ld      UIE,#00000001B  ;receive interrupts, no DMA
        ld      URC,#00000010B  ;enable receiver
;
;UART is initialized, reset acknowledge bit and begin
;
        bitr    ACKB,#ACKBIT    ;reset acknowldege bit if set
        ld      P2BIP,#00000001B        ;reset interrupt input flip-flop
        ei                      ;enable interrupts
WAIT:   ldw     MPTR,#MSG       ;point to message
        call    SENDM           ;send the message
        ld      INPNT,#0        ;set input pointer to register 0
        ld      OUTPNT,#0       ;set output pointer to register 0
WAIT1:  call    SNDBUF          ;send any characters in buffer
        jr      WAIT1           ;loop back
;
;
;
SENDM:  tm      P2,#00001000B   ;printer busy
        jr      nz,SENDM        ;wait for printer unbusy
        btjrt   SENDM,ACKB,#ACKBIT      ;see if the acknowledge has occurred
                                        ;from possible last byte
        bits    ACKB,#ACKBIT    ;set acknowledge bit before writing to output
        ldci    r0,@MPTR        ;get the character
        ld      P4,r0           ;send to printer
        nop                     ;allow 18 clocks for strobe
        nop
        nop
        cp      r0,#'$'         ;last character?
        jr      ne,SENDM        ;loop back for next
        ret
;
;
SNDBUF: cp      INPNT,OUTPNT    ;compare inpointer to outpointer
        jr      ne,SC1          ;send character if any to send
        ret                     ;otherwise return
SC1:    tm      P2,#00001000B   ;printer busy?
        jr      nz,SC1          ;if so, wait until it is not busy
        btjrt   SC1,ACKB,#ACKBIT        ;see if acknowledge has occurred
                                ;from possible last byte

        di
        bits    ACKB,#ACKBIT    ;set acknowledge bit before writing to output
        ld      P4,@OUTPNT      ;send the character
        tm      P2,#00000001B
        jr      z,HON           ;if host is on
        ld      r0,OUTPNT       ;get the output pointer
        xor     r0,#10000000B   ;add 128 to it
        cp      INPNT,r0        ;turn host back on when 128 bytes left in buf
        jr      ne,HON          ;otherwise keep sending
        and     P2,#11111110B   ;host back on
HON:    nop
        inc     OUTPNT          ;bump pointer
        ei                      ;to make sure pointer not changed
        ret
;
;send character in r0
SENDC:  tm      UTC,#00000010B  ;transmit buffer empty yet
        jr      z,SENDC         ;if not, wait until it is
        ld      UIO,r0          ;load the character into the transmitter
        ret
;receive character available interrupt
RXDATI: ld      r0,UIO          ;get input from console
        and     r0,#7fH         ;remove upper parity bit
        call    SENDC           ;echo to console
        ld      @INPNT,r0       ;save the character
        inc     INPNT           ;bump input pointer
        cp      INPNT,OUTPNT    ;has the input made a complete loop?
        jr      ne,RXIT
;
;receive character buffer full, stop sending device
;
        or      P2,#00000001B   ;raise DTR to stop host sending
INTRET:
RXIT:   iret
;
ACKSTB: tm      P2,#00010000B   ;is line low or high now
        bitr    ACKB,#ACKBIT    ;reset acknowledge bit in register
;
ACKS1:  tm      P2,#00010000B           ;test ack bit
        jr      z,ACKS1         ;wait here till end of strobe
        ld      P2BIP,#00000001B        ;reset p24 interrupt pending register
        iret                    ;and return
;
MSG:    .ASCII  CR,LF,'Super8 serial/parallel test program.',CR,LF
        .ASCII  'Second line test data',CR,LF,'$'

        .END
```

```
;
        .TITLE  Sample Zilog Super 8 Serial to Parallel Converter with XON/XOFF
;
;
;===========================================================
;=      TITLE:          SERPAR1.S                          =
;=      DATE:           JULY 17, 1986                      =
;=      PURPOSE:        TO DEMONSTRATE INTERRUPT           =
;=                      DRIVEN SERIAL PORT IN A            =
;=                      REALISTIC APPLICATION.             =
;=                      THIS APPLICATION RECEIVES          =
;=                      SIMPLE SERIAL DATA A SENDS IT      =
;=                      OUT THE PARALLEL PORT TO A         =
;=                      PRINTER. FLOW CONTROL IS BY        =
;=                      XON/XOFF COMMANDS ON THE BACK      =
;=                      CHANNEL TO THE HOST                =
;=      ASSEMBLER:      ZILOG ASMS8 ASSEMBLER              =
;=      PROGRAMMER:     CHARLES M. LINK, II                =
;===========================================================
;
;
;
        .PAGE   55      ;set maximum page size to 55 lines
;***********************************************************
;*                                                         *
;*                    GENERAL EQUATES                      *
;*                                                         *
;***********************************************************
;
CR:     .equ    0dH     ;carriage return
LF:     .equ    0aH     ;line feed
```

```
XON:    .equ    11H     ;control-Q or DC1
XOFF:   .equ    13H     ;control-S or DC3
;
;
;************************************************************
;*                                                          *
;*                  REGISTER EQUATE TABLE                   *
;*                                                          *
;************************************************************
;
;working register equates
INPNT:  .equ    R3      ;input character pointer
OUTPNT: .equ    R4      ;output character pointer
MPTR:   .equ    RR6     ;message pointer for external memory
ACKB:   .equ    R5      ;byte containing acknowledge bit
ACKBIT: .equ    0       ;bit set = no acknowledge yet
                        ;bit clear = not waiting on acknowledge
XBIT:   .equ    1       ;XOFF send to host
;
;************************************************************
;*                                                          *
;*                  INTERRUPT VECTOR TABLE                  *
;*                                                          *
;************************************************************
;
INTR0:  .WORD   INTRET          ;this area should always be defined
INTR1:  .WORD   INTRET          ;as it reserves the lower 32 bytes
INTR2:  .WORD   INTRET          ;for the interrupt table.  the name
INTR3:  .WORD   INTRET          ;of the subroutine for each particular
INTR4:  .WORD   INTRET          ;interrupt service would normally be
INTR5:  .WORD   INTRET          ;named here.
INTR6:  .WORD   INTRET
INTR7:  .WORD   INTRET
INTR8:  .WORD   INTRET
INTR9:  .WORD   INTRET
INTR10: .WORD   RXDATI          ;receive data interrupt
INTR11: .WORD   INTRET
INTR12: .WORD   INTRET
INTR13: .WORD   INTRET
INTR14: .WORD   ACKSTB          ;acknowledge strobe interrupt
INTR15: .WORD   INTRET
;
;************************************************************
;*                                                          *
;*              START OF PROGRAM EXECUTION                  *
;*                                                          *
;************************************************************
;
START:  di                      ;for emulation if nothing else
        jr      START1          ;program execution unconditionally
                                ;begins at this location after reset
                                ;and power up.
        .ASCII  'REL 0 7/17/86' ;jump around optional ascii string
                                ;containing release info, copyright, etc.
START1: sb0                     ;select register bank 0
        ld      EMT,#00000000B  ;external memory timing=no wait input, normal
                                ;memory timing, no wait states, stack internal,
                                ;and DMA internal
        ld      P0,#00H         ;address begins at 0000h, set upper byte
        ld      P0M,#11111111B  ;select all lines as address
        ld      PM,#00110000B   ;enable port 0 as upper 8 bits address
        ld      H1C,#00000000B  ;handshake not enabled port 0
;
;port 1 is defined in romless part as address/data.  it is not necessary
;here to initialize that port
;
        ld      P2,#00100000B   ;port 2 outputs low, except strobe bit
        ld      P3,#00H         ;port 3 outputs low
        ld      P2AM,#10001010B ;p31,20,21 as output,p30 input
                                ;it is necessary here to configure p30 as input
                                ;for the receive data, and p31 as output for
                                ;transmit data for UART
        ld      P2BM,#10100010B ;p32,33,22 as output, 23 as input
        ld      P2CM,#10101001B ;p34,35,25 as output, 24 as input, interrupt en
        ld      P2DM,#10101010B ;p36,37,26,27 as output
;
        ld      P4,#00000000B   ;clear port 4 register
        ld      P4D,#00000000B  ;set all bits of P4 as outputs
        ld      P4OD,#00000000B ;active push/pull
        ld      HOC,#11110001B  ;handshake enable for port 4, 16 clock pulse
;
;basic Super 8 I/O is initialized, now internal registers
;
        ld      RP0,#0C0H       ;set working register low to lower 8 bytes
        ld      RP1,#0C8H       ;set working register high to upper 8 bytes
        ld      SPL,#0FFH       ;set stack pointer to start at top of set two
                                ;note here that only lower 8 bits are used
                                ;for stack pointer.  location 0FFH is wasted
                                ;as stack operation.  SPH is general purpose
                                ;storage.
;
;now clear the internal memory and stack area
;
        ld      SPH,#0FFH       ;point to top of general purpose register
ZERO:   clr     @SPH            ;zero it
        dec     SPH
        jr      nz,ZERO         ;do it until register set is all cleared
        clr     @SPH            ;zero last register
;
;now everything except working registers is cleared
;
;cpu and memory now initialized, set up timer for real time clock
;
        ld      SYM,#00000000B  ;disable fast interrupt response
        ld      IPR,#10111111B  ;interrupt priority
                                ;IRQ6>IRQ7>IRQ5>IRQ4>IRQ3>IRQ2>IRQ1>IRQ0
        ld      IMR,#01010000B  ;rx interrupts, acknowledge strobe
;
;timer is set, now lets initialize the UART for polled operation
;
        sb1                     ;bank 1
        ld      UMA,#01110000B
                                ;time constant  = (12,000,000/4/16/9600/2)-1=
                                ;8.76 rounded to 9.
                                ;note that a 12 Mhz does not make a very
                                ;accurate baud rate source.  error is large
        ld      UBGH,#^HB(00009)        ;high byte of time constant
        ld      UBGL,#^LB(00009)        ;low byte of time constant
        ld      UMB,#00011110B  ;p21=p21data,auto-echo is off, transmit and
                                ;receive clock is baud rate generator output,
                                ;baud rate generator input is system clock / 2,
                                ;baud rate generator is enabled, loopback
                                ;is disabled
        sb0                     ;select bank 0
        ld      UTC,#10001000B  ;select p31 as transmit data out, 1 stop bit
                                ;and transmit enable
        ld      UIE,#00000001B  ;receive interrupts, no DMA
        ld      URC,#00000010B  ;enable receiver
;
;UART is initialized, reset acknowledge bit and begin
;
        bitr    ACKB,#ACKBIT    ;reset acknowldege bit if set
        bitr    ACKB,#XBIT      ;reset XON/XOFF bit
        ld      P2BIP,#00000001B        ;reset interrupt input flip-flop
        ei                      ;enable interrupts
WAIT:   ldw     MPTR,#MSG       ;point to message
        call    SENDM           ;send the message
        ld      INPNT,#0        ;set input pointer to register 0
        ld      OUTPNT,#0       ;set output pointer to register 0
WAIT1:  call    SNDBUF          ;send any characters in buffer
        jr      WAIT1           ;loop back
;
;
;
SENDM:  tm      P2,#00001000B   ;printer busy
        jr      nz,SENDM        ;wait for printer unbusy
        btjrt   SENDM,ACKB,#ACKBIT      ;see if the acknowledge has occurred
                                        ;from possible last byte
        bits    ACKB,#ACKBIT    ;set acknowledge bit before writing to output
        ldci    r0,@MPTR        ;get the character
        ld      P4,r0           ;send to printer
        nop                     ;allow 18 clocks for strobe
        nop
        nop
        cp      r0,#'$'         ;last character?
        jr      ne,SENDM        ;loop back for next
        ret
```

```
;
;timer is initialized, now lets enable interrupts and wait
        ldw     CINCR,#1        ;start column at beginning of sine table
        ldw     RINCR,#1        ;start row at beginning of sine table
;
;this example loads the tones for digit '1'
;user software would, of course have to manipulate these registers for
;proper tone control
;
        ldw     CFINCR,#CFREQI  ;load column frequency increment
        ldw     RFINCR,#RFREQI  ;load row frequency increment
        ldw     POINT,#SINTAB   ;pointer points to sine table
        ld      CVAL,#080H      ;initial value to prevent glitch at start
        ei                      ;enable interrupts
WAIT:   nop
        nop
        nop
        nop
        jr      WAIT            ;loop back
;
;Timer interrupt.  Occurs SAMPLE times per second
;interrupt outputs value to DAC-08 and then determines value for next
;interrupt.  This assures no bit jitter.
;
TIMER0: ld      p4,CVAL         ;write new value to DAC-08
        rcf                     ;clear carry flag
        add     CINCRL,CFINCL   ;find next position in sine table
        adc     CINCRH,CFINCH   ;by adding frequency offset to last position
        ld      POINTL,CINCRH   ;set new pointer into sine table
        ldc     CVAL,@POINT     ;get value from sine table
        add     RINCRL,RFINCL   ;find next position in sine table
        adc     RINCRH,RFINCH   ;by adding frequencty offset to last position
        ld      POINTL,RINCRH   ;set new pointer into sine table
        ldc     RVAL,@POINT     ;get second value from sine table
        add     CVAL,RVAL       ;form a complex waveform from two sine values
        or      COCT,#00000010B ;reset end of count interrupt
INTRET: iret                    ;and return from interrupt
;
;***********************************************************
;*                                                         *
;*                      SINE WAVE LOOKUP                   *
;*                                                         *
;***********************************************************
;
;sine table for DTMF generation using DAC-08.  Table based upon
;case of waveform consisting of two sine waves summed to provide a single
;complex waveform with minumum amplititude = 0 volts and maximum
;amplititude = 5 volts.  DAC-08 input for 0 volts = 00H
;5 volts = 0FFH.  Both waves must total no more than 0FFH, therefore
;maximum for one wave must be 1/2 5 volts or 080H.
;Table generated using following BASICA program,
;then typed into program.
;
;       10 CLS                  ;clear screen
;       20 PI=3.141593          ;define PI
;       30 FOR I=0 TO 255       ;256 total values
;       40 C=360/256            ;define basic interval value
;       50 D=C*I                ;value from zero on sine wave
;       60 E=D*PI/180
;       70 F=SIN(E)             ;figure sine for interval from 0
;       80 G=F*63               ;sine range should be from -63 to 63
;       90 H=64+G               ;make result from 0 to 127
;       100 J=CINT(H)           ;round to nearest integer
;       110 A$=HEX$(J)          ;convert to hex
;       120 PRINT A$            ;on screen
;       130 LPRINT A$           ;on printer
;       140 NEXT                ;do next inverval
;       150 END
;
;*note-remove comments, BASICA will not accept ; as comment delimiter
;
;
SINTAB: .ORG    0400H           ;begin sine table on even byte boundary
        .byte   040H,042H,043H,045H,046H,048H,049H,04BH,04CH,04EH,04FH,051H
        .byte   052H,054H,055H,057H,058H,05AH,05BH,05CH,05EH,05FH,060H,062H
        .byte   063H,064H,066H,067H,068H,069H,06AH,06BH,06DH,06EH,06FH,070H
        .byte   071H,072H,073H,074H,074H,075H,076H,077H,078H,078H,079H,07AH
        .byte   07AH,07BH,07BH,07CH,07CH,07DH,07DH,07DH,07EH,07EH,07EH,07FH
```

```
;
;
SNDBUF: cp      INPNT,OUTPNT    ;compare inpointer to outpointer
        jr      ne,SC1          ;send character if any to send
        ret                     ;otherwise return
SC1:    tm      P2,#00001000B   ;printer busy?
        jr      nz,SC1          ;if so, wait until it is not busy
        btjrt   SC1,ACKB,#ACKBIT        ;see if acknowledge has occurred
                                ;from possible last byte

        di
        bits    ACKB,#ACKBIT    ;set acknowledge bit before writing to output
        ld      P4,@OUTPNT      ;send the character
        btjrf   HON,ACKB,#XBIT  ;host is still sending
        ld      r0,OUTPNT       ;get the output pointer
        xor     r0,#10000000B   ;add 128 to it
        cp      INPNT,r0        ;turn host back on when 128 bytes left in buf
        jr      ne,HON          ;otherwise keep sending
        ld      r0,XON          ;send XON to host to start it sending again
        call    SENDC
        bitr    ACKB,#XBIT      ;reset XOFF bit
HON:    nop
        inc     OUTPNT          ;bump pointer
        ei                      ;to make sure pointer not changed
        ret
;
;send character in r0
SENDC:  tm      UTC,#00000010B  ;transmit buffer empty yet
        jr      z,SENDC         ;if not, wait until it is
        ld      UIO,r0          ;load the character into the transmitter
        ret
;receive character available interrupt
RXDATI: ld      r0,UIO          ;get input from console
        and     r0,#7fH         ;remove upper parity bit
        call    SENDC           ;echo to console
        ld      @INPNT,r0       ;save the character
        inc     INPNT           ;bump input pointer
        ld      r0,INPNT        ;get the input pointer
        add     r0,#5           ;allow 5 characters after XOFF
        cp      r0,OUTPNT       ;has the input made a complete loop?
        jr      ne,RXIT
;
;receive character buffer full, stop sending device
;
        ld      r0,#XOFF        ;send XOFF to host
        call    SENDC           ;send it
        bits    ACKB,#XBIT      ;set the XOFF bit
INTRET:
RXIT:   iret
;
ACKSTB: tm      P2,#00010000B   ;is line low or high now
        bitr    ACKB,#ACKBIT    ;reset acknowledge bit in register
;
ACKS1:  tm      P2,#00010000B           ;test ack bit
        jr      z,ACKS1         ;wait here till end of strobe
        ld      P2BIP,#00000001B        ;reset p24 interrupt pending register
        iret                    ;and return
;
MSG:    .ASCII  CR,LF,'Super8 serial/parallel test program.',CR,LF
        .ASCII  'Second line test data',CR,LF,'$'

        .END
```

Super8™ Microcomputer

Contents

Chapter 1. Super8 Overview 1

Chapter 2. Architectural Overview 2

Chapter 3. Address Spaces 3

Chapter 4. Addressing Modes 4

Chapter 5. Instruction Set 5

Contents (Continued)

5

Chapter 6. Interrupts

6

Chapter 7. Reset and Clock

7

Chapter 8. I/O Ports

8

8

Chapter 9. Counter/Timers

9

Chapter 10. UART

10

Chapter 11. DMA Channel

11

Contents (Continued)

11

Chapter 12. External Interface

12

Chapter 1
Super8 Overview

1.1 INTRODUCTION

The Super8 family consists of basic microcomputers, protopack emulators, and ROMless microcomputers. The various family members differ in the amount of on-chip ROM and the physical packaging.

All of the Super8 family members offer a full-duplex universal asynchronous receiver/transmitter (UART) with an on-chip baud-rate generator, two 16-bit programmable counter/timers, a direct memory access (DMA) controller, and an on-chip oscillator.

1.2 FEATURES

Super8 microprocessor features include:

- 325 byte-wide registers, including 272 general-purpose registers and 53 mode and control registers
- Full-duplex UART with special features
- Up to 32 bit-programmable and 8 byte-programmable I/O lines, with 2 handshake channels
- Addressing of up to 128K byes of memory
- An interrupt structure that supports:
  - 27 interrupt sources
  - 16 interrupt vectors (2 reserved for future versions)
  - 8 interrupt levels
  - Servicing in 6 CPU clock cycles
- Two Register Pointers that allow use of short and fast instructions to access register groups within 600 ns.
- An instruction set that includes multiply and divide instructions, Boolean and BCD operations
- Additional instructions that support threaded-code languages, such as Forth

1.3 BASIC MICROCOMPUTERS

These parts are the core of the Super8 family of products. They have various amounts of mask-programmable on-chip ROM, are suitable for high volume applications, and require a single +5 Vdc power supply.

1.4 PROTOPACK MICROCOMPUTERS

These parts function as emulators for the basic microcomputer versions. They use the same package and pin-out as the basic microcomputer but also have a 28-pin "piggy back" socket on the top into which a ROM or EPROM can be installed, to replace the on-chip ROM of the basic microcomputer.

This package permits the protopack to be used in prototype and final PC boards while still permitting user program development. When a final program is developed, it can be mask-programmed into the production microcomputer device, directly replacing the emulator. The protopack parts are also useful in situations where the cost of mask-programming is prohibitive or where program flexibility is desired.

1.5 ROMLESS MICROCOMPUTERS

The ROMless microcomputers are similar to the basic microcomputer parts, but have no internal ROM. Port 1 is dedicated as an 8-bit address/data bus and $P0_0$-$P0_4$ are dedicated address lines. Up to 64K bytes of external memory can be addressed by configuring Port 0 as address bits. The address capability can be doubled to 128K bytes by programming $P3_5$ of Port 3 as the Data Memory select signal $\overline{DM}$. The two states of this signal can be used with the 16-bit address bus to address two separate banks of external memory, each with up to 64K bytes.

Chapter 2
Architectural Overview

2.1 INTRODUCTION

The Super8 is a versatile single-chip microcomputer that can be programmed for many different memory and I/O configurations. This flexibility has been achieved by merging a multiplexed address/data bus with several I/O-oriented ports. This provides the user with large amounts of external memory while maintaining many I/O lines. Figure 2-1 shows the Super8 block diagram.

2.2 ADDRESS SPACES

To provide for both I/O and memory intensive applications, the Super8 supports three basic address spaces:

- Program memory (internal and external)
- Data memory (external)
- Register file (internal)

A maximum of 64K bytes of program memory is directly addressable. When present, internal program memory normally consists of mask-programmed ROM. The data memory space is 64K bytes in size.

The ease of interfacing with external memory is enhanced with options for programmable wait states and half-speed memory timing, as well as an optional external wait input.

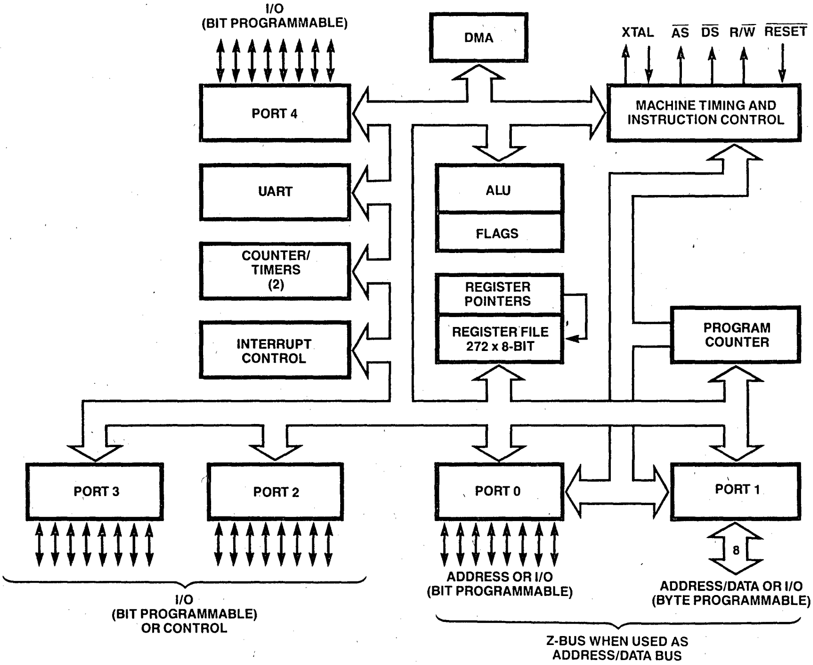


Figure 2-1. Functional Block Diagram

2.3 REGISTER FILE

The Super8 architecture centers around an internal register file composed of 325 registers. All registers are eight bits wide. Of the 272 general-purpose registers, 208 can be used as an accumulator, address pointer, index register, data register, or stack register. The 64 remaining general-purpose registers are limited to Indirect or Indexed addressing mode functions such as stacks, data buffers, and look-up tables. Fifty-three registers are dedicated to special control and status operations.

2.3.1 Register Pointer

The register file is logically divided into 32 working register groups of 8 registers each when using 4-bit register addressing. Two groups may be active at any one time and the two Register Pointers (RP0 and RP1) contain the base addresses of these two working register groups. This allows fast context switching and shorter instruction formats.

2.3.2 Instruction Pointer

The Super8 hardware includes features that facilitate the implementation of threaded-code languages such as Forth. These include a special 16-bit register called the Instruction Pointer (IP) and three special CPU instructions called NEXT, ENTER, and EXIT. The IP can also be used to support the fast interrupt processing mode.

2.4 INSTRUCTION SET

The CPU has an instruction set designed for its large register file. This includes a full complement of 8-bit arithmetic and logical operations, including multiply and divide. Binary-Coded Decimal (BCD) operations are supported using a decimal adjustment of binary values. Incrementing and decrementing 16-bit quantities for addresses and counters are also supported. Extensive bit manipulation, including Rotate and Shift instructions, round out the data manipulation capabilities of the Super8. No special I/O instructions are necessary since I/O is mapped into the register file.

2.4.1 Addressing Modes

The addressing modes of the Super8 Central Processing Unit (CPU) are:

- Register (R)
- Indirect Register (IR)
- Indirect Address (IA)
- Immediate (IM)
- Direct Address (DA)
- Indexed (X)
- Relative Address (RA)

Register, Indirect Register, and Immediate addressing modes are available for Load, Arithmetic, Logical, Shift, Rotate, and Stack instructions. Conditional jumps support both the Direct and Relative addressing modes, while Jump and Call instructions support the Direct, Indirect, and Indirect Register addressing modes. Only Load instructions support Indexed addressing.

2.4.2 Data Types

The Super8 CPU supports operations on bits, bytes, BCD digits, and 2-byte words.

Bits in the register file can be set, cleared, complemented, and tested. Bits within a byte are numbered from 0 to 7; bit 0 is the least significant (right-most) bit.

Bytes in the register file can be operated on by Arithmetic, Logical, Shift and Rotate, and Load instructions. Bytes in memory can be operated on only by load or stack instructions.

Manipulation of BCD digits, packed two to a byte, is accomplished by a Decimal Adjust instruction and a Swap instruction. Decimal Adjust is used after either a binary addition or subtraction on BCD digits.

Words in the register file can be loaded, incremented, and decremented with the 16-bit Load Word, Increment Word, and Decrement Word instructions.

2.5 I/O OPERATIONS

The Super8 has I/O lines grouped into five ports of eight lines each. Ports are configurable as input, output, or bidirectional. Under software control, the ports can provide timing, status signals, address outputs, and I/O ports with or without handshaking. Multiprocessor system configurations are also supported.

2.5.1 Interrupts

I/O operations can be interrupt-driven or polled. The Super8 supports 16 vectored interrupts on eight different levels from 27 interrupt sources. Each level can be masked and prioritized. Optional high-speed interrupt processing can be used on any one of the levels for minimum latency.

2.5.2 On-Chip Peripherals

To help cope with real-time problems such as counting/timing, the Super8 contains two counter/timers with a large number of user selectable modes. It also contains an on-chip universal asynchronous receiver/transmitter (UART) which has its own built-in baud-rate generator that can be used as a counter when not being used to generate baud rates.

A DMA channel is provided that allows high-speed data transfers between on-chip peripherals and the register file or external memory.

2.6 OSCILLATOR

In addition to these features, the Super8 offers an on-chip oscillator requiring only an external crystal for operation.

Chapter 3
Address Spaces

3.1 INTRODUCTION

The Super8 microprocessor supports the following address spaces:

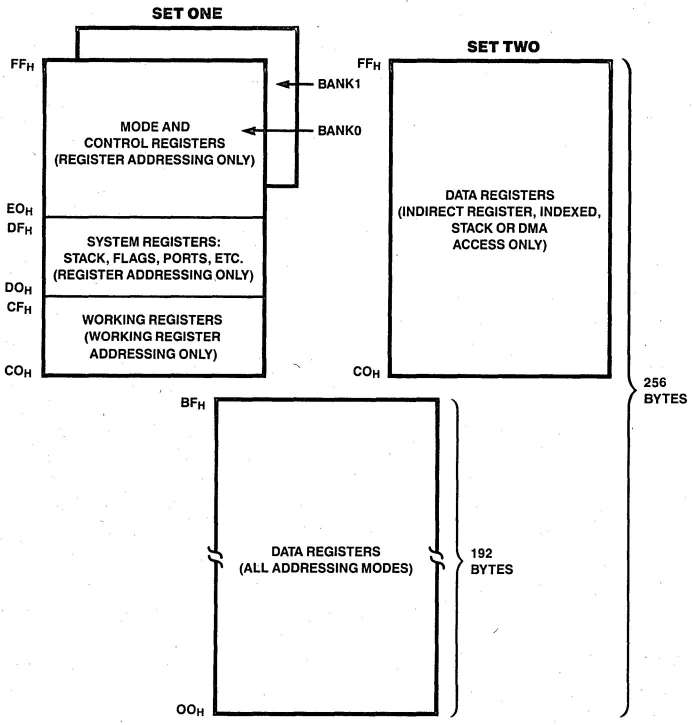
- CPU register file
- Program memory
- Data memory

3.2 CPU REGISTER FILE

Registers within the Super8 CPU's internal register file are identified with an 8-bit address, yielding 256 possible register addresses. However, the upper 64 addresses are used more than once, as described below. A total of 325 registers is available, including 272 general-purpose registers and 53 special control and status registers. Two of these registers are Register Pointers.

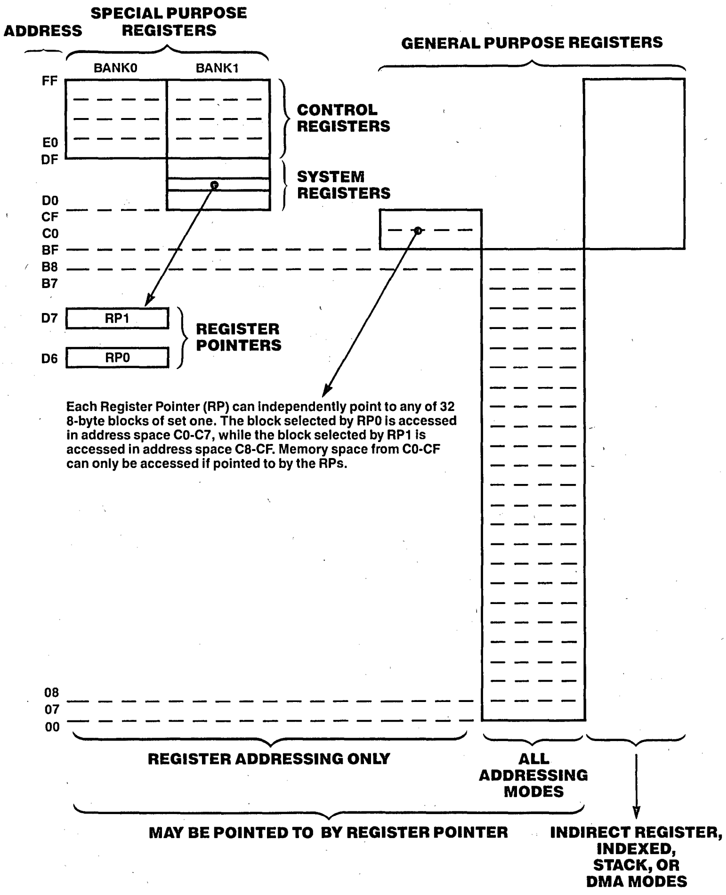
A total of 325 registers is accessible with 192 registers (00_H-BF_H) accessible in all addressing modes. These can be used as accumulators, working registers, data buffers, internal stack, and so forth. It is possible to set up a 256-byte data buffer and still have 16 registers remaining as accumulators and working registers.

Figures 3-1 and 3-2 show layouts of the register file address space. The upper 64 bytes of the address space ($C0_H$-FF_H) contain two sets of registers. The first set can be accessed only by the Register addressing mode; the second set can be accessed by the Indirect Register and Indexed addressing modes, stack operations, and DMA accesses. The registers in the second set are usable as data buffers or as an internal stack area.



SET ONE
FFH
BANK1
BANK0
MODE AND CONTROL REGISTERS (REGISTER ADDRESSING ONLY)
EOH
DFH
SYSTEM REGISTERS: STACK, FLAGS, PORTS, ETC. (REGISTER ADDRESSING ONLY)
DOH
CFH
WORKING REGISTERS (WORKING REGISTER ADDRESSING ONLY)
COH
SET TWO
FFH
DATA REGISTERS (INDIRECT REGISTER, INDEXED, STACK OR DMA ACCESS ONLY)
COH
256 BYTES
BFH
DATA REGISTERS (ALL ADDRESSING MODES)
192 BYTES
OOH

Figure 3-1. Super8 Registers



Each Register Pointer (RP) can independently point to any of 32 8-byte blocks of set one. The block selected by RP0 is accessed in address space C0-C7, while the block selected by RP1 is accessed in address space C8-CF. Memory space from C0-CF can only be accessed if pointed to by the RPs.

Figure 3-2. Super8 Register File Address Spaces

The first set consists of three subsets of registers. The bottom sixteen registers ($C0_H$-CF_H) are available for use as accumulators or working registers. The middle sixteen registers ($D0_H$-DF_H) are used for system registers--Stack Pointer, Flag register, I/O ports, and so forth. The upper 32 bytes ($E0_H$-FF_H) consist of two banks of registers. Each bank is selected by a bit located in the Flag register called the Bank Address bit. These two banks, a total of 64 bytes, are used for Mode and Control registers. Only 38 of these 64 bytes are currently used. The remaining 26 bytes are reserved for future expansion.

Registers can be accessed as either 8- or 16-bit registers using Register, Indirect Register, or Indexed addressing modes. For register addresses $C0_H$ to FF_H, the addressing mode used determines the actual register being accessed. Registers accessed as 16-bit registers are treated as even-odd register pairs, with the most significant byte of data stored in the even-numbered register and the least significant byte stored in the next higher odd-numbered register (Figure 3-3).

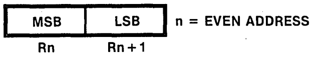


Figure 3-3. 16-Bit Register Addressing

With few exceptions, all instructions that reference or modify a register may do so to any of the 325 8-bit registers or 176 16-bit register pairs, regardless of the particular register, as long as the proper addressing mode is used. The instructions operate on I/O ports, system registers, mode and control registers, and general-purpose registers without the need for special-purpose instructions.

Usage and access are shown in Table 3-1.

Table 3-1. Super8 Register File

Registers	Usage	Access
00-BF	General-purpose registers	Register, Indirect Register, or Indexed modes, via on-chip DMA operations, or as part of internal stack
C0-FF Set Two	General-purpose registers	Indirect Register or Indexed modes, via on-chip DMA operations, or as part of internal stack
C0-FF Set One	Working registers only	Register mode
D0-DF Set One	System registers	Register mode
E0-FF Set One	Mode and control registers	Register mode

The instructions can access 8-bit registers or 16-bit register pairs using either 4-bit or 8-bit address fields. When using 4-bit register addressing, the register file is logically divided into 32 groups of 8 working registers, as shown in Figure 3-4. All the registers in a working register set have the same value for their five most-significant address bits. The two Register Pointers (RP0 and RP1) are system registers that contain the base addresses of two active working register groups.

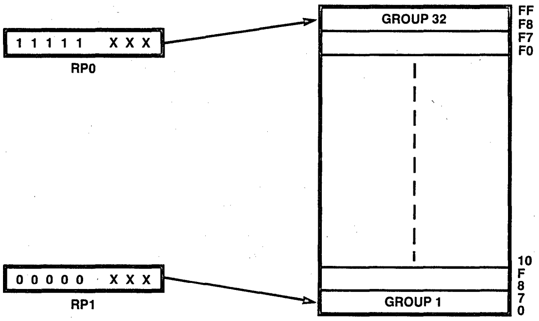


Figure 3-4. Working Register Groups

Note that 4-bit register addressing (Figure 3-5) is a Register addressing mode so that the registers accessible by this mode include the mode and control registers, system registers, and working register groups.

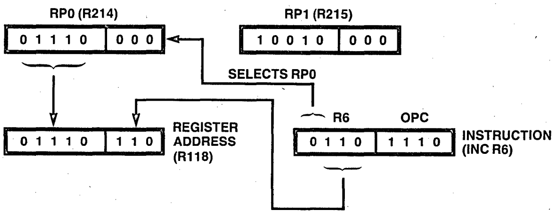


Figure 3-5. Working Register Addressing

Working registers are typically specified by short format instructions; when a working register destination is used in the instruction, only four bits of address are needed to specify the register; one bit selects the appropriate Register Pointer and three bits provide the least-significant bits of the register address. The five most-significant bits of the address come from the selected Register Pointer and together they form an 8-bit address. Applications using working registers require fewer bytes and have a reduced execution time.

The Register Pointer also speeds context switching when processing interrupts or changing tasks. A special Set Register Pointer (SRP) instruction is provided for setting the Register Pointer contents.

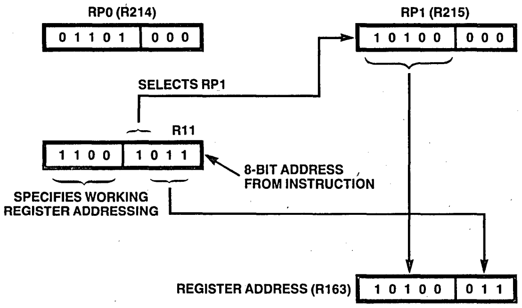


Figure 3-6. 8-Bit Working Register Addressing

Not all instructions have 4-bit addressing modes, but the active working registers can still be accessed using 8-bit addressing without having to know the contents of the Register Pointers. Figure 3-6 shows how this works. The upper four bits of the 8-bit address contain 1100 to specify working register addressing. Bit 3 selects Register Pointer 0 or 1, which supplies the upper five bits of the final address while the lower three bits come from bits 0-2 of the original 8-bit address.

Any address in the range $C0_H$-CF_H (R192-R207) will invoke working register addressing. Therefore the registers physically located at these addresses can only be accessed when selected by a Register Pointer (see Figure 3-2).

After Reset, the register pointers will be set to RP0 = $C0_H$ and RP1 = $C8_H$.

3.3 SYSTEM REGISTERS AND MODE AND CONTROL REGISTERS

The system registers govern the operation of the CPU and can be accessed using any of the instructions that reference the register file using Register addressing mode. These registers can be accessed as working registers. Table 3-2 shows the system registers.

The Super8 uses a 16-bit Program Counter (PC) to control the sequence of instructions in the currently executing program. The PC is not an addressable register.

Mode and control registers are used to transfer data, configure the mode of operation, and control the operation of the on-chip peripherals. These registers are accessed using Register addressing mode and are shown in Table 3-3. These registers can be accessed as working registers. The current "bank" is determined by bit D_0 in the Flag register (R213).

3.4 PROGRAM AND DATA MEMORY

Program memory is memory that can hold code or data. Instruction code can be fetched from program memory, data can be read from program memory and, if external program memory is implemented in RAM, data or code can be written to program memory. Memory addresses are 16 bits long, allowing a maximum of 64K bytes of program

Table 3-2. System Registers

Decimal Address	Hexadecimal Address	Register Name	Identifier
222	DE	System Mode	SYM
221	DD	Interrupt Mask Register	IMR
220	DC	Interrupt Request Register	IRQ
219	DB	Instruction Pointer (Bits 7-0)	IPL
218	DA	Instruction Pointer (Bits 15-8)	IPH
217	D9	Stack Pointer (Bits 7-0)	SPL
216	D8	Stack Pointer (Bits 15-8)	SPH
215	D7	Register Pointer 1	RP1
214	D6	Register Pointer 0	RP0
213	D5	Program Control Flags	FLAGS
212	D4	Port 4	P4
211	D3	Port 3	P3
210	D2	Port 2	P2
209	D1	Port 1	P1
208	D0	Port 0	P0

Table 3-3. Mode and Control Registers

Decimal Address	Hexadecimal Address	Register Name	Identifier
Bank 0 Registers			
255	FF	Interrupt Priority	IPR
254	FE	External Memory Timing	EMT
253	FD	Port 2/3B Interrupt Pending	P2BIP
252	FC	Port 2/3A Interrupt Pending	P2AIP
251	FB	Port 2/3D Mode	P2DM
250	FA	Port 2/3C Mode	P2CM
249	F9	Port 2/3B Mode	P2BM
248	F8	Port 2/3A Mode	P2AM
247	F7	Port 4 Open-Drain	P4OD
246	F6	Port 4 Direction	P4D
245	F5	Handshake 1 Control	H1C
244	F4	Handshake 0 Control	H0C
241	F1	Port Mode	PM
240	F0	Port 0 Mode	P0M
239	EF	UART Data	UIO
237	ED	UART Interrupt Enable	UIE
236	EC	UART Receive Control	URC
235	EB	UART Transmit Control	UTC
229	E5	Counter 1 Capture Low	C1CL
228	E4	Counter 1 Capture High	C1CH
227	E3	Counter 0 Capture Low	C0CL
226	E2	Counter 0 Capture High	C0CH
225	E1	Counter 1 Control	C1CT
224	E0	Counter 0 Control	C0CT
Bank 1 Registers			
255	FF	Wake-Up Mask	WUMSK
254	FE	Wake-Up Match	WUMCH
251	FB	UART Mode B	UMB
250	FA	UART Mode A	UMA
249	F9	UART Baud-Rate Generator Low	UBGL
248	F8	UART Baud-Rate Generator High	UBGH
241	F1	DMA Count Low	DCL
240	F0	DMA Count High	DCH
229	E5	Counter 1 Time Constant Low	C1TCL
228	E4	Counter 1 Time Constant High	C1TCH
227	E3	Counter 0 Time Constant Low	C0TCL
226	E2	Counter 0 Time Constant High	C0TCH
225	E1	Counter 1 Mode	C1M
224	E0	Counter 0 Mode	C0M

memory. The bottom of program memory is in the on-chip ROM; the remaining program memory can be implemented external to the Super8.

Data memory is memory that can hold only data to be read or written, not instruction code; instruction fetches never reference data memory. Data memory is always implemented external to the Super8.

External data memory can be incorporated with or separated from the external program memory address space. To implement separate program and data memory address spaces external to the Super8, a port output pin ($P3_5$) must be defined as the Data Memory select ($\overline{DM}$) output. This output remains high when fetching instructions or accessing data in the program memory address space and goes low when accessing data in the data memory address space. Thus, this signal can be used to segregate

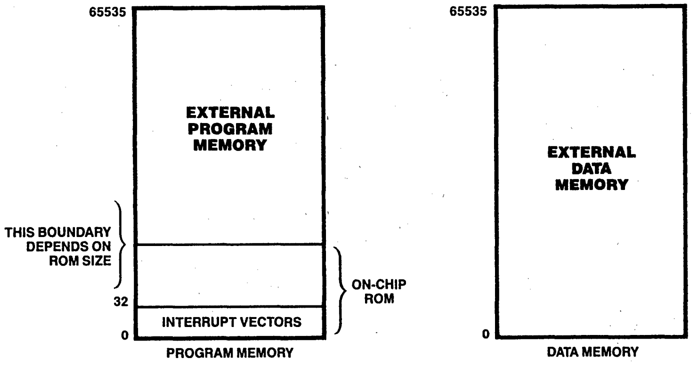


Figure 3-7. Program and Data Memory Address Spaces

the program and data spaces external to the Super8. Separate forms of Load instructions are used to access the two memory address spaces: the LDC instruction and its derivatives access program memory, and the LDE instruction and its derivatives access data memory.

Program and data memory maps are illustrated in Figure 3-7.

To access memory beyond the on-chip ROM, Ports 0 and 1 must be configured as a memory interface. Port 1 can be configured as a multiplexed address/data bus (AD_0-AD_7), thus providing address lines A_0-A_7 and data lines D_0-D_7. Port 0 can be configured on an individual bit basis for up to eight additional address lines (A_8-A_{15}). Both parts are supported by the control lines Address Strobe ($\overline{AS}$), Data Strobe ($\overline{DS}$), and Read/Write (R/$\overline{W}$).

In the ROMless version, Port 1 is automatically configured as a multiplexed address/data bus. Port 0 bits 0-4 will be configured as address bits A_8-A_{12} at Reset, but any Port 0 bit may be defined as either I/O or address as needed.

For more details on external memory interface, see section 12.3.

No matter which version of the Super8 is used, the first 32 bytes of program memory are reserved for the interrupt vectors. Thus the first address available for a user program is location 32. This address is automatically loaded into the Program Counter whenever a hardware Reset occurs.

3.5 CPU AND USER STACKS

The Super8 uses a stack for implementing subroutine calls and returns, interrupt processing, and general dynamic storage (via the Push and Pop instructions). The Super8 provides hardware support for stack operations from either the register file or data memory. Stack location selection is under software control via the External Memory Timing register (R254, Bank 0).

Register pair RR216 forms the 16-bit Stack Pointer, used for CPU stack operations. The address is stored with the most significant byte in R216 and least significant in R217 (Figure 3-8).

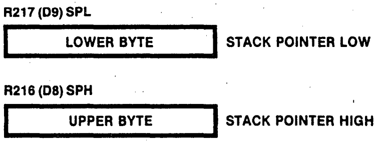


Figure 3-8. Stack Pointer

The Stack Pointer is decremented before a Push operation and incremented after a Pop operation. The stack address always points to the last data stored on the top-of-stack.

The stack is used to hold the return address for CALL instructions and interrupts, as well as data. The contents of the Program Counter are saved on the stack during a CALL instruction and restored during a RET instruction. During interrupts, the contents of the Program Counter and Flag register are saved on the stack. The IRET instruction restores them (Figure 3-9).

When the Super8 is configured to use an internal stack (the register file), register R217 serves as the Stack Pointer and register R216 is a general-purpose register. However, if an overflow or underflow condition occurs due to the incrementing

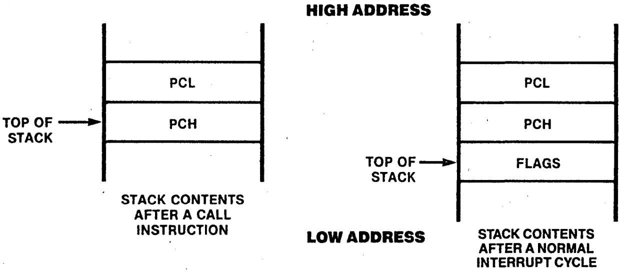

HIGH ADDRESS
PCL
PCH
TOP OF STACK
STACK CONTENTS AFTER A CALL INSTRUCTION
PCL
PCH
FLAGS
TOP OF STACK
LOW ADDRESS
STACK CONTENTS AFTER A NORMAL INTERRUPT CYCLE

Figure 3-9. Stack Operations

Table 3-4. User Stack Operations Summary

		--- Stack Location ---		
Stack Type*	**Operation**	**Register File**	**Program Memory**	**Data Memory**
Ascending	PUSH to stack	PUSHUI	LDCPI	LDEPI
	POP from stack	POPUD	LDCD	LDED
Descending	PUSH to stack	PUSHUD	LDCPD	LDEPD
	POP from stack	POPUI	LDCI	LDEI

* Ascending stack goes from low to high addresses within memory or register file. Descending stack goes from high to low addresses within memory or register file.

and decrementing of normal stack operations, the contents of register R216 are affected.

The Super8 also provides for user-defined stacks in both the register file and in program or data memory. These stacks can be made to increment or decrement on Push and Pop. Table 3-4 summarizes the kinds of stacks and the instructions used.

3.6 INSTRUCTION POINTER (IP)

The Super8 provides hardware support for implementation of threaded-code languages such as Forth. An important part of that support is in the form of a special register called the Instruction Pointer (IP) (Figure 3-10). The Instruction Pointer is made up of register pair RR218, with R218 holding the most significant byte of a memory address and R219 the least significant byte.

A threaded-code language may be considered to have created a higher level imaginary machine within the actual hardware machine. For comparison purposes, the IP is to the imaginary machine as the Program Counter is to the actual hardware machine.

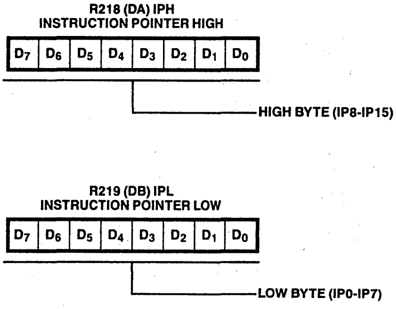


Figure 3-10. Instruction Pointer

The IP is used by three special instructions called NEXT, ENTER, and EXIT. The instruction NEXT passes control from the hardware machine to the imaginary machine, while ENTER and EXIT are the imaginary machine equivalents of subroutine CALLS and RETURNs in the hardware machine.

The IP can also be used in the fast interrupt processing mode for special interrupt handling (see section 6.2). It can be used either for interrupt processing or imaginary machine processing, but not for both at the same time.

Chapter 4
Addressing Modes

4.1 INTRODUCTION

Instructions are stored as lists of bytes in program memory that are fetched via instruction fetches using the Program Counter. Instructions will indicate both the action to be performed and the data to be operated on. The method used to determine the location of the data operand is called the addressing mode.

Operands specified in Super8 instructions are either condition codes, immediate data, or the designation of a register file, program memory, or data memory location.

For the Super8, there are seven explicit addressing modes (i.e., addressing modes designated by the programmer):

- Register (R)
- Indirect Register (IR)
- Indexed (X)
- Direct Address (DA)
- Indirect Address (IA)
- Relative Address (RA)
- Immediate (IM)

Not all modes are available with each instruction (refer to the individual instruction descriptions in section 5.5).

Accessing an individual register requires specifying an 8-bit address in the range 0-255 or a working register's 4-bit address. The most significant bit of the 4-bit working register address selects one of two Register Pointers: if this bit is 0, then R214 (RP0) is selected; if it is 1, then R215 (RP1) is selected. The address of the actual register being accessed is formed by the concatenation of the high order five bits of the value contained in the selected Register Pointer with the remaining three bit address supplied by the instruction.

A register pair can be used to specify a 16-bit value or memory address. The Load Constant instruction and its derivatives (LDC, LDCD, LDCI, LDCPD, LDCPI) load data from program memory; the Load External instruction and its derivatives (LDE, LDED, LDEI, LDEPD, LDEPI) load from program memory. See the instruction set in Chapter 5 for further details.

4.2 REGISTER ADDRESSING (R)

In the Register addressing mode, the operand value is the contents of the specified register or register pair (Figures 4-1 and 4-2).

Registers $C0_H$-FF_H (set one) can only be accessed with the Register addressing mode.

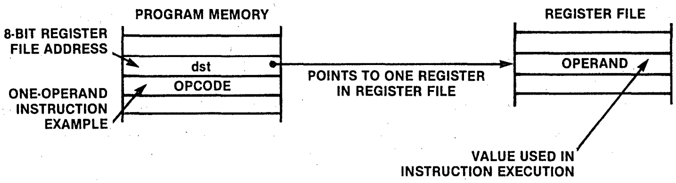

PROGRAM MEMORY
8-BIT REGISTER FILE ADDRESS
dst
OPCODE
ONE-OPERAND INSTRUCTION EXAMPLE
POINTS TO ONE REGISTER IN REGISTER FILE
REGISTER FILE
OPERAND
VALUE USED IN INSTRUCTION EXECUTION

Figure 4-1. Register Addressing

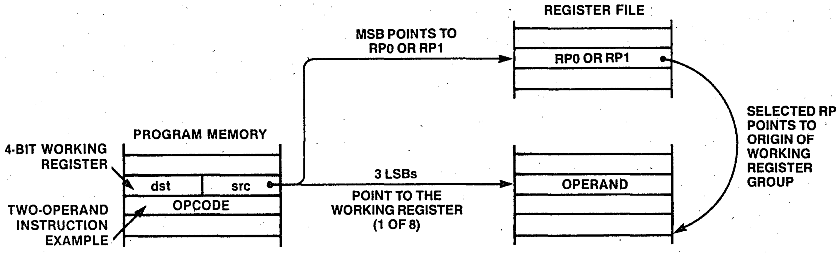

REGISTER FILE
MSB POINTS TO RP0 OR RP1
RP0 OR RP1
SELECTED RP POINTS TO ORIGIN OF WORKING REGISTER GROUP
PROGRAM MEMORY
4-BIT WORKING REGISTER
dst
src
OPCODE
TWO-OPERAND INSTRUCTION EXAMPLE
3 LSBs
POINT TO THE WORKING REGISTER (1 OF 8)
OPERAND

Figure 4-2. Working Register Addressing

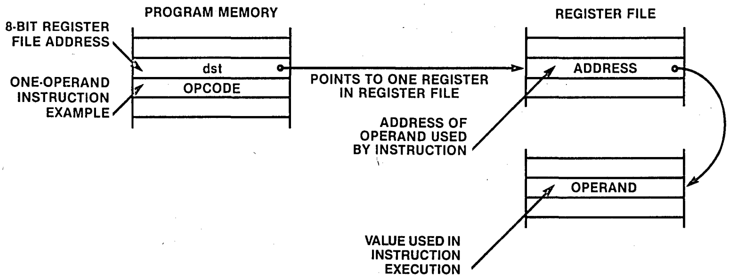

PROGRAM MEMORY
REGISTER FILE
8-BIT REGISTER FILE ADDRESS
dst
ONE-OPERAND INSTRUCTION EXAMPLE
OPCODE
POINTS TO ONE REGISTER IN REGISTER FILE
ADDRESS
ADDRESS OF OPERAND USED BY INSTRUCTION
OPERAND
VALUE USED IN INSTRUCTION EXECUTION

Figure 4-3. Indirect Register Addressing to Register File

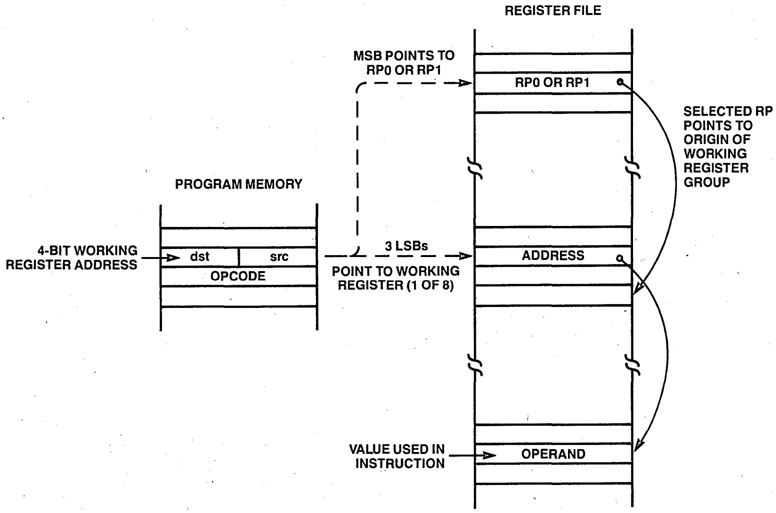

REGISTER FILE
MSB POINTS TO RP0 OR RP1
RP0 OR RP1
SELECTED RP POINTS TO ORIGIN OF WORKING REGISTER GROUP
PROGRAM MEMORY
4-BIT WORKING REGISTER ADDRESS
dst
src
OPCODE
3 LSBs
POINT TO WORKING REGISTER (1 OF 8)
ADDRESS
VALUE USED IN INSTRUCTION
OPERAND

Figure 4-4. Indirect Working Register Addressing to Register File

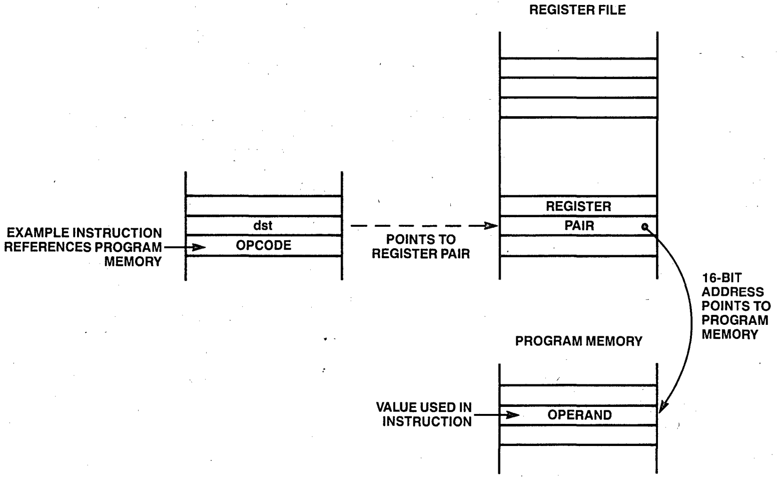

REGISTER FILE
EXAMPLE INSTRUCTION REFERENCES PROGRAM MEMORY
dst
OPCODE
POINTS TO REGISTER PAIR
REGISTER PAIR
16-BIT ADDRESS POINTS TO PROGRAM MEMORY
PROGRAM MEMORY
VALUE USED IN INSTRUCTION
OPERAND

Figure 4-5. Indirect Register Addressing to Program Memory

4.3 INDIRECT REGISTER ADDRESSING (IR)

In the Indirect Register addressing mode, the content of the specified register or register pair is the address of the operand (Figures 4-3, 4-4, 4-5, and 4-6). Depending on the instruction used, the actual address may point to a register, program memory, or data memory.

Any general-purpose byte register can be used to indirectly address another register; any general-purpose register pair can be used to indirectly address a memory location.

General-purpose registers $C0_H$-FF_H (set two) can be accessed only with the Indirect Register and Indexed addressing modes.

4.4 INDEXED ADDRESSING (X)

The Indexed addressing mode involves adding an offset to a base address during instruction execution to calculate the effective address of the operand. The Indexed addressing mode can be used to access registers or memory areas.

For register accesses, an 8-bit base address given in the instruction is added to an 8-bit offset given in a working register (Figure 4-7). General-purpose registers $C0_H$-FF_H (set two) can be accessed only with the Indirect Register and Indexed addressing modes. The LD instruction is the only instruction that allows Indexed addressing of the registers.

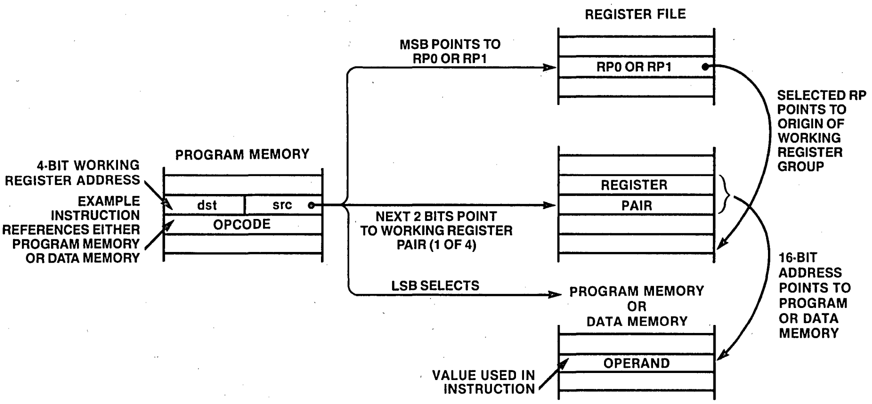

REGISTER FILE
MSB POINTS TO RP0 OR RP1
RP0 OR RP1
SELECTED RP POINTS TO ORIGIN OF WORKING REGISTER GROUP
PROGRAM MEMORY
4-BIT WORKING REGISTER ADDRESS
EXAMPLE INSTRUCTION REFERENCES EITHER PROGRAM MEMORY OR DATA MEMORY
dst
src
OPCODE
NEXT 2 BITS POINT TO WORKING REGISTER PAIR (1 OF 4)
REGISTER
PAIR
16-BIT ADDRESS POINTS TO PROGRAM OR DATA MEMORY
LSB SELECTS
PROGRAM MEMORY OR DATA MEMORY
OPERAND
VALUE USED IN INSTRUCTION

Figure 4-6. Indirect Working Register Addressing to Program or Data Memory

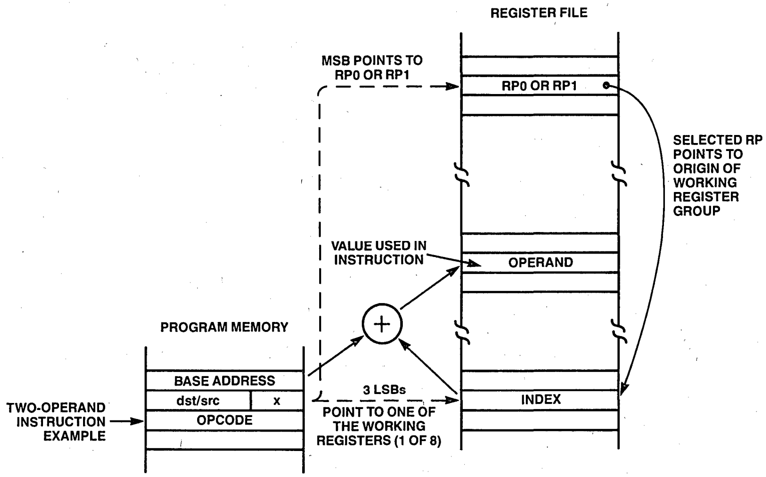

REGISTER FILE
MSB POINTS TO RP0 OR RP1
RP0 OR RP1
SELECTED RP POINTS TO ORIGIN OF WORKING REGISTER GROUP
VALUE USED IN INSTRUCTION
OPERAND
+
PROGRAM MEMORY
BASE ADDRESS
dst/src
x
OPCODE
TWO-OPERAND INSTRUCTION EXAMPLE
3 LSBs
POINT TO ONE OF THE WORKING REGISTERS (1 OF 8)
INDEX

Figure 4-7. Indexed Addressing to Register File

For memory accesses, the base address is held in the working register pair designated in the instruction and an 8-bit or 16-bit offset given in the instruction is added to that base address (Figures 4-8 and 4-9). In the short offset Indexed addressing mode, the 8-bit displacement is treated as a signed integer in the range -128 to +127. Only the LDC and LDE instructions allow Indexed addressing of memory.

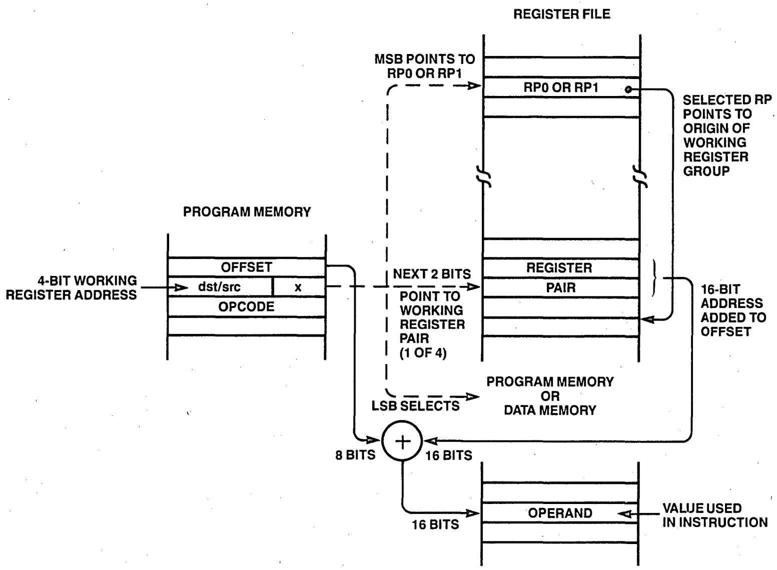

REGISTER FILE
MSB POINTS TO RP0 OR RP1
RP0 OR RP1
SELECTED RP POINTS TO ORIGIN OF WORKING REGISTER GROUP
PROGRAM MEMORY
OFFSET
4-BIT WORKING REGISTER ADDRESS
dst/src
x
OPCODE
NEXT 2 BITS POINT TO WORKING REGISTER PAIR (1 OF 4)
REGISTER PAIR
16-BIT ADDRESS ADDED TO OFFSET
PROGRAM MEMORY OR DATA MEMORY
LSB SELECTS
8 BITS
+
16 BITS
16 BITS
OPERAND
VALUE USED IN INSTRUCTION

Figure 4-8. Indexed Addressing to Program or Data Memory with Short Offset

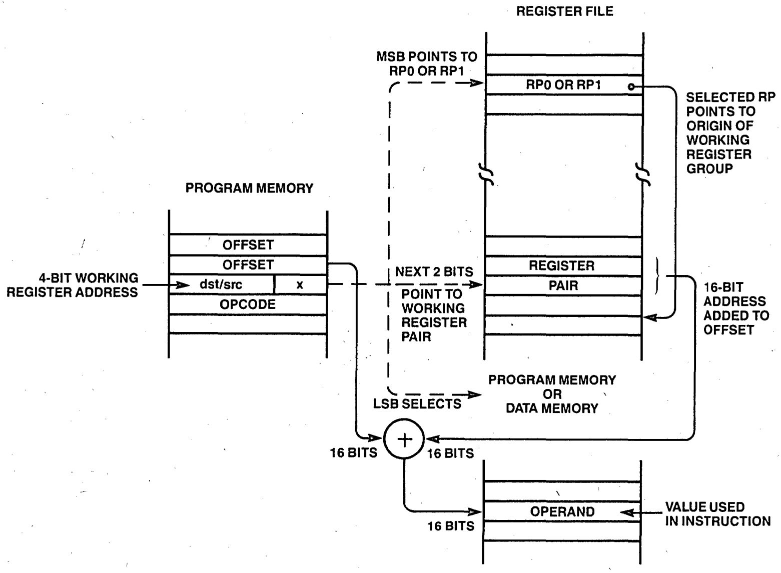

REGISTER FILE
MSB POINTS TO RP0 OR RP1
RP0 OR RP1
SELECTED RP POINTS TO ORIGIN OF WORKING REGISTER GROUP
PROGRAM MEMORY
OFFSET
OFFSET
4-BIT WORKING REGISTER ADDRESS
dst/src
x
OPCODE
NEXT 2 BITS POINT TO WORKING REGISTER PAIR
REGISTER PAIR
16-BIT ADDRESS ADDED TO OFFSET
PROGRAM MEMORY OR DATA MEMORY
LSB SELECTS
16 BITS
+
16 BITS
16 BITS
OPERAND
VALUE USED IN INSTRUCTION

Figure 4-9. Indexed Addressing to Program or Data Memory with Long Offset

4.5 DIRECT ADDRESSING (DA)

In Direct addressing mode, as seen in Figures 4-10 and 4-11, the 16-bit memory address of the operand is given in the instruction. This mode is used by the Jump and Call instructions to specify the 16-bit destination that is loaded into the Program Counter to implement the Jump or Call. This mode is also supported by the LDE and LDC instructions to specify the source or destination memory address for a load between a register and a memory location. Memory loads with LDC and LDE can use the Direct or Indirect Register addressing modes.

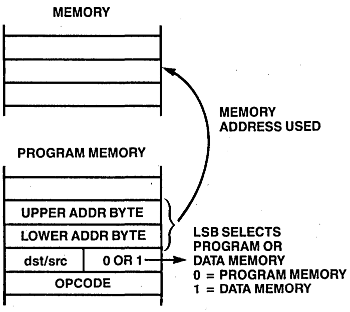


Figure 4-10. Direct Addressing for Load Instructions

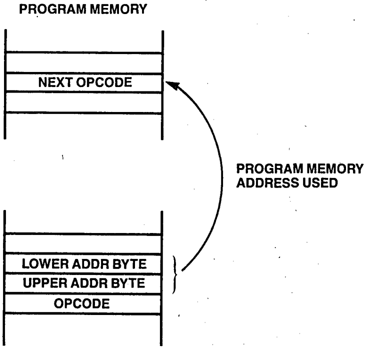


Figure 4-11. Direct Addressing for Call and Jump Instructions

4.6 INDIRECT ADDRESSING (IA)

In the Indirect addressing mode (Figure 4-12), the instruction specifies a pair of memory locations found in the lowest 256 bytes of program memory. The selected pair, in turn, contains the actual address of the next instruction to be executed.

Since the Indirect addressing mode assumes that the operand is located in the lowest 256 bytes of memory, only an 8-bit address is supplied in the instruction; the upper bytes of the destination address are assumed to be all 0s. Only the CALL instruction uses this addressing mode.

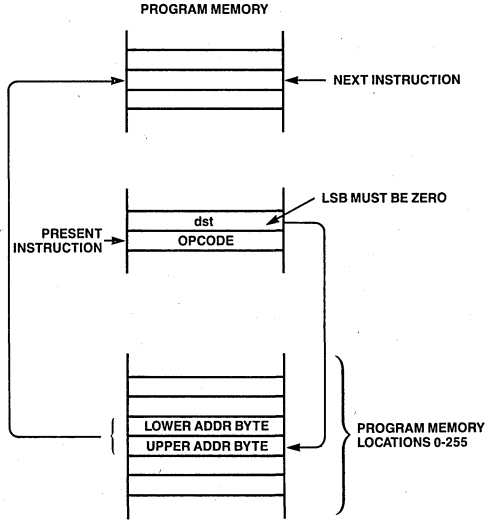


Figure 4-12. Indirect Addressing

4.7 RELATIVE ADDRESSING (RA)

In the Relative addressing mode (Figure 4-13), a twos-complement signed displacement in the range -128 to +127 is specified in the instruction and added to the value contained in the Program Counter. The result is the address of the next instruction to be executed. Prior to the add, the Program Counter contains the address of the instruction following the current instruction.

The Relative addressing mode is supported by several program control type instructions: BTJRF, BTJRT, DJNZ, CPIJE, CPIJNE, and JR.

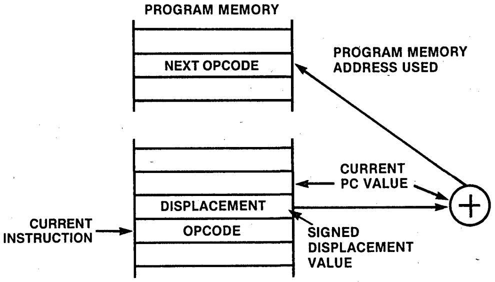


Figure 4-13. Relative Addressing

4.8 IMMEDIATE ADDRESSING (IM)

In the Immediate addressing mode (Figure 4-14), the operand value used in the instruction is the value supplied in the operand field itself. The operand may be a byte or word in length, depending on the instruction. The Immediate addressing mode is useful for loading constant values into registers.

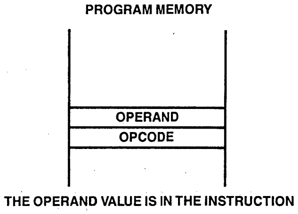


Figure 4-14. Immediate Addressing

Chapter 5
Instruction Set

5.1 FUNCTIONAL SUMMARY

Super8 instructions can be divided functionally into the following seven groups:

- Load
- Arithmetic
- Logical
- Program Control
- Bit Manipulation
- Rotate and Shift
- CPU Control

Table 5-1 shows the instructions belonging to each group and the number of operands required for each, where "src" is the source operand, "dst" is the destination operand, and "cc" is the condition code.

With few exceptions, all instructions that reference a register may do so to any of the 325 8-bit registers or 176 16-bit register pairs. Thus, the same instructions are used to operate on I/O ports, system registers, mode and control registers, and general-purpose registers.

The exceptions to the above are as follows:

- The Decrement and Jump on Non-Zero (DJNZ) instruction's register operand must be a general-purpose byte register.

- The following control registers are write-only registers: Port Mode, Port 2/3 A Mode, Port 2/3 B Mode, Port 2/3 C Mode, Port 2/3 D Mode, Handshake 0 Control, and Handshake 1 Control.

- The Flags register (R213) cannot be the destination for an instruction that alters the flags as part of its operation.

5.2 PROCESSOR FLAGS

Flag register R213 supplies the status of the Super8 CPU at any time. The flags and their bit positions are shown in Figure 5-1.

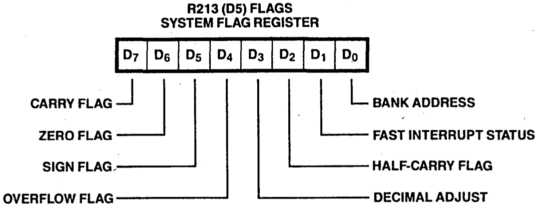


Figure 5-1. Flag Register

This register contains eight bits of status information that are set or cleared by CPU operations. Four of the bits (C, V, Z, and S) are testable for use with conditional Jump instructions. Two of the flags (H and D) are not testable and are used only for BCD arithmetic. All flags are restored to the pre-interrupt value by a return from interrupt.

Bank Address Flag (BA). This bit selects which of the two groups of mode and control registers is active.

Carry Flag (C). This flag is set to 1 whenever the result of an arithmetic operation generates a carry-out of or borrow into the high order bit 7. It is cleared to 0 whenever an operation does not generate a carry or borrow condition. This flag can be set, cleared, and complemented by the Set Carry Flag (SCF), Reset Carry Flag (RCF), and Complement Carry Flag (CCF) instructions.

Decimal-Adjust Flag (D). The Decimal-Adjust flag is used for BCD arithmetic. It is set to 1 following a subtraction operation and cleared to 0 following an addition operation. Since the algorithms for correcting BCD addition and subtraction are different, this flag is used to specify the type of instruction last executed so that the subsequent Decimal Adjust (DA) operation can function properly. It is not normally used as a test flag by the programmer.

Fast Interrupt Status Flag (FIS). This bit is set to 1 during a Fast Interrupt and cleared to 0 during the Interrupt Return (IRET).

Table 5-1. Instruction Group Summary

Mnemonic	Operands	Instruction
Load Instructions		
CLR	dst	Clear
LD	dst,src	Load
LDB	dst,src	Load Bit
LDE	dst,src	Load Data Memory
LDC	dst,src	Load Program memory
LDED	dst,src	Load Data Memory and Decrement
LDCD	dst,src	Load Program Memory and Decrement
LDEI	dst,src	Load Data Memory and Increment
LDCI	dst,src	Load Program Memory and Increment
LDEPD	dst,src	Load Data Memory with Pre-Decrement
LDCPD	dst,src	Load Program Memory with Pre-Decrement
LDEPI	dst,src	Load Data memory with Pre-Increment
LDCPI	dst,src	Load Program Memory with Pre-Increment
LDW	dst,src	Load Word
POP	dst	Pop
POPUD	dst,src	Pop User Stack (Decrementing)
POPUI	dst,src	Pop User Stack (Incrementing)
PUSH	src	Push
PUSHUD	dst,src	Push User Stack (Decrementing)
PUSHUI	dst,src	Push User Stack (Incrementing)
Arithmetic Instructions		
ADC	dst,src	Add with Carry
ADD	dst,src	Add
CP	dst,src	Compare
DA	dst	Decimal Adjust
DEC	dst	Decrement
DECW	dst	Decrement Word
DIV	dst,src	Divide
INC	dst	Increment
INCW	dst	Increment Word
MULT	dst,src	Multiply
SBC	dst,src	Subtract with Carry
SUB	dst,src	Subtract
Logical Instructions		
AND	dst,src	Logical AND
COM	dst	Complement
OR	dst,src	Logical OR
XOR	dst,src	Logical Exclusive OR
Program Control Instructions		
BTJRF	dst,src	Bit Test and Jump Relative on False
BTJRT	dst,src	Bit Test and Jump Relative on True
CALL	dst	Call Procedure
CPIJE	dst,src	Compare, Increment and Jump on Equal

Table 5-1. Instruction Group Summary (Continued)

Mnemonic	Operands	Instruction
Program Control Instructions (Continued)		
CPIJNE	dst,src	Compare, Increment and Jump on Non-Equal
DJNZ	r,dst	Decrement Register and Jump on Non-Zero
ENTER		Enter
EXIT		Exit
IRET		Interrupt Return
JP	cc,dst	Jump on Condition Code
JP	dst	Jump Unconditional
JR	cc,dst	Jump Relative on Condition Code
JR	dst	Jump Relative Unconditional
NEXT		Next
RET		Return
WFI		Wait for Interrupt
Bit Manipulation Instructions		
BAND	dst,src	Bit AND
BCP	dst,src	Bit Compare
BITC	dst	Bit Complement
BITR	dst	Bit Reset
BITS	dst	Bit Set
BOR	dst,src	Bit OR
BXOR	dst,src	Bit XOR
TCM	dst,src	Test Complement Under Mask
TM	dst,src	Test Under Mask
Rotate and Shift Instructions		
RL	dst	Rotate Left
RLC	dst	Rotate Left through Carry
RR	dst	Rotate Right
RRC	dst	Rotate Right through Carry
SRA	dst	Shift Right Arithmetic
SWAP	dst	Swap Nibbles
CPU Control Instructions		
CCF		Complement Carry Flag
DI		Disable Interrupts
EI		Enable Interrupts
NOP		No Operation
RCF		Reset Carry Flag
SB0		Set Bank 0
SB1		Set Bank 1
SCF		Set Carry Flag
SRP	src	Set Register Pointers
SRP0	src	Set Register Pointer 0
SRP1	src	Set Register Pointer 1

Half-Carry Flag (H). The Half-Carry flag is set to 1 whenever an addition generates a carry-out of bit 3 or subtraction generates a borrow into bit 3. The Half-Carry flag is used by the Decimal Adjust (DA) instruction to convert the binary result of a previous addition or subtraction into the correct decimal (BCD) result. It is not normally used as a test flag by the programmer.

Overflow Flag (V). This flag is set to 1 during arithmetic, rotate, or shift operations that result in a value greater than +127 or less than -128 (the maximum and minimum numbers that can be represented in twos-complement form); it is cleared to 0 whenever the result is a value within these ranges. This flag is also cleared to 0 following logical operations.

Sign Flag (S). When performing arithmetic operations on signed numbers, binary twos-complement notation is used to represent and process information. A positive number is identified by a 0 in the most significant bit position; when this occurs, the Sign flag is also cleared to 0. A negative number is identified by a 1 in the most significant bit position and therefore the Sign flag would be set to 1.

Zero Flag (Z). During arithmetic and logical operations, the Zero flag is set to 1 if the result is zero and cleared to 0 if the result is non-zero. When testing bits in a register or when shifting or rotating, the Zero flag is set to 1 if the result is zero; if the result is not zero, the flag is cleared to 0.

5.3 CONDITION CODES

Flags C, Z, S, and V control the operation of the "conditional" Jump instructions. Sixteen frequently used combinations of flag settings are encoded in a 4-bit field called the condition code (cc), which forms a part of the conditional instructions (bits 4-7).

The condition codes and the flag settings they represent are listed in Table 5-2.

5.4 NOTATION AND BINARY ENCODING

The following sections describe the symbols used for operands and status flags, and the flag settings and their meanings.

Table 5-2. Condition Codes

Binary	Mnemonic	Meaning	Flags Set
0000	F	Always False	-
1000		Always True	-
0111*	C	Carry	C = 1
1111*	NC	No Carry	C = 0
0110*	Z	Zero	Z = 1
1110*	NZ	Not Zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No Overflow	V = 0
0110*	EQ	Equal	Z = 1
1110*	NE	Not Equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	(Z OR (S XOR V)) = 0
0010	LE	Less than or equal	(Z OR (S XOR V)) = 1
1111*	UGE	Unsigned greater than or equal	C = 0
0111*	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1

*Indicates condition codes that relate to two different mnemonics but test the same flags. For example, Z and EQ are both True if the Zero flag is set, but after an ADD instruction, Z would probably be used, while after a CP instruction, EQ would probably be used.

Table 5-3. Notation and Binary Encoding

Notation	Meaning	Actual Operand/Range
cc	Condition code	See condition code list (Table 5-2)
r	Working register only	Rn: where n = 0-15
rb	Bit b of working register	Rn #b: where n = 0-15 and b = 0-7
r0	Bit 0 of working register	Rn: where n = 0-15
rr	Working register pair	RRp: where p = 0,2,4,...,14
R	Register or working register	Reg: where reg represents a number in the range 0-255 Rn: where n = 0-15
Rb	Bit b of register or working register	Reg #b: where reg represents a number in the range 0-255 and b = 0-7 Rn #b: where n = 0-15 and b = 0-7
RR	Register pair or working register pair	Reg: where reg reprsents an even number in the range 0-254 RRp: where p = 0,2,...,14
IA	Indirect addressing mode	# addrs: where addrs represents an even number in the range 0-254
Ir	Indirect working register only	@Rn: where n = 0-15
IR	Indirect register or working register	@reg: where reg represents a number in the range 0-255 @Rn: where n = 0-15
Irr	Indirect working register only	@RRp: where p = 0,2,...,14
IRR	Indirect register pair or working register pair	@reg: where reg represents an even number in the range 0-254 @RRp: where p = 0,2,...,14
X	Indexed addressing mode	reg (Rn): where reg represents a number in the range 0-255 and n = 0-15
XS	Indexed (Short Offset) addressing mode	addrs (RRp): where addrs represents a number in the range -128 to +127 and p = 0,2,...,14
XL	Indexed (Long Offset) addressing mode	addrs (RRp): where addrs represents a number in the range 0-65,535 and p = 0,2,...,14
DA	Direct addressing mode	addrs: where addrs represents a number in the range 0-65,535
RA	Relative addressing mode	addrs: where addrs represents a number in the range +127,-128 that is an offset relative to the address of the next instruction
IM	Immediate addressing mode	#data: where data is a number between 0 and 255
IML	Immediate (Long) addressing mode	#data: where data is a number between 0 and 65,535

5.4.1 Notational Shorthand

Operands and status flags are represented by a notational shorthand in the detailed instruction descriptions of section 5.5.2. The notation for operands (condition codes and addressing modes) and the actual operands they represent are shown in Table 5-3.

Additional Symbols Used:

Symbol	**Meaning**
dst	Destination operand
src	Source operand
@	Indirect Register address prefix
SP	Stack Pointer (R216 and R217)
PC	Program Counter
IP	Instruction Pointer (R218 and R219)
FLAGS	Flag register (R213)
RPO	Register Pointer 0 (R214)
RP1	Register Pointer 1 (R215)
IMR	Interrupt Mask register (R221)
#	Immediate operand or Register address prefix
%	Hexadecimal number prefix
OPC	Opcode

Assignment of a value is indicated by the symbol "<--"; for example,

```
dst <-- dst + src
```

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr (n)" is used to refer to bit "n" of a given location. For example,

```
dst (7)
```

refers to bit 7 of the destination operand.

5.4.2 Flag Settings

Notation for the flags is shown below.

Flag	**Meaning**
C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-Adjust flag
H	Half-Carry flag
0	Cleared to 0
1	Set to 1
*	Set or Cleared according to operation
-	Unaffected
X	Undefined

Figure 5-2 provides a quick reference guide to the commands.

SUPER8 OPCODE MAP

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6 **DEC** R_1	6 **DEC** IR_1	6 **ADD** r_1,r_2	6 **ADD** r_1,Ir_2	10 **ADD** R_2,R_1	10 **ADD** IR_2,R_1	10 **ADD** R_1,IM	10 **BOR*** r_0-R_b	6 **LD** r_1,R_2	6 **LD** r_2,R_1	12/10 **DJNZ** r_1,RA	12/10 **JR** cc,RA	6 **LD** r_1,IM	12/10 **JP** cc,DA	6 **INC** r1	14 **NEXT**
1	6 **RLC** R_1	6 **RLC** IR_1	6 **ADC** r_1,r_2	6 **ADC** r_1,Ir_2	10 **ADC** R_2,R_1	10 **ADC** IR_2,R_1	10 **ADC** R_1,IM	10 **BCP** r_1,b,R_2	↓	↓	↓	↓	↓	↓	↓	20 **ENTER**
2	6 **INC** R_1	6 **INC** IR_1	6 **SUB** r_1,r_2	6 **SUB** r_1,Ir_2	10 **SUB** R_2,R_1	10 **SUB** IR_2,R_1	10 **SUB** R_1,IM	10 **BXOR*** r_0-R_b	↓	↓	↓	↓	↓	↓	↓	22 **EXIT**
3	10 **JP** IRR_1	**NOTE C**	6 **SBC** r_1,r_2	6 **SBC** r_1,Ir_2	10 **SBC** R_2,R_1	10 **SBC** IR_2,R_1	10 **SBC** R_1,IM	**NOTE A**	↓	↓	↓	↓	↓	↓	↓	6 **WFI**
4	6 **DA** R_1	6 **DA** IR_1	6 **OR** r_1,r_2	6 **OR** r_1,Ir_2	10 **OR** R_2,R_1	10 **OR** IR_2,R_1	10 **OR** R_1,IM	10 **LDB*** r_0-R_b	↓	↓	↓	↓	↓	↓	↓	6 **SBO**
5	10 **POP** R_1	10 **POP** IR_1	6 **AND** r_1,r_2	6 **AND** r_1,Ir_2	10 **AND** R_2,R_1	10 **AND** IR_2,R_1	10 **AND** R_1,IM	8 **BITC** r_1,b	↓	↓	↓	↓	↓	↓	↓	6 **SBI**
6	6 **COM** R_1	6 **COM** IR_1	6 **TCM** r_1,r_2	6 **TCM** r_1,Ir_2	10 **TCM** R_2,R_1	10 **TCM** IR_2,R_1	10 **TCM** R_1,IM	10 **BAND*** r_0-R_b	↓	↓	↓	↓	↓	↓	↓	
7	10/12 **PUSH** R_2	12/14 **PUSH** IR_2	6 **TM** r_1,r_2	6 **TM** r_1,Ir_2	10 **TM** R_2,R_1	10 **TM** IR_2,R_1	10 **TM** R_1,IM	**NOTE B**	↓	↓	↓	↓	↓	↓	↓	
8	10 **DECW** RR_1	10 **DECW** IR_1	10 **PUSHUD** IR_1,R_2	10 **PUSHUI** IR_1,R_2	24 **MULT** R_2,RR_1	24 **MULT** IR_2,RR_1	24 **MULT** IM,RR_1	10 **LD** r_1,x,r_2	↓	↓	↓	↓	↓	↓	↓	6 **DI**
9	6 **RL** R_1	6 **RL** IR_1	10 **POPUD** IR_2,R_1	10 **POPUI** IR_2,R_1	28/12 **DIV** R_2,RR_1	28/12 **DIV** IR_2,RR_1	28/12 **DIV** IM,RR_1	10 **LD** r_2,x,r_1	↓	↓	↓	↓	↓	↓	↓	6 **EI**
A	10 **INCW** RR_1	10 **INCW** IR_1	6 **CP** r_1,r_2	6 **CP** r_1,Ir_2	10 **CP** R_2,R_1	10 **CP** IR_2,R_1	10 **CP** R_1,IM	**NOTE D**	↓	↓	↓	↓	↓	↓	↓	14 **RET**
B	6 **CLR** R_1	6 **CLR** IR_1	6 **XOR** r_1,r_2	6 **XOR** r_1,Ir_2	10 **XOR** R_2,R_1	10 **XOR** IR_2,R_1	10 **XOR** R_1,IM	**NOTE E**	↓	↓	↓	↓	↓	↓	↓	16/6 **IRET**
C	6 **RRC** R_1	6 **RRC** IR_1	16/18 **CPIJE** Ir,r_2,RA	12 **LDC*** r_1,Irr_2	10 **LDW** RR_2,RR_1	10 **LDW** IR_2,RR_1	12 **LDW** RR_1,IML	6 **LD** r_1,Ir_2	↓	↓	↓	↓	↓	↓	↓	6 **RCF**
D	6 **SRA** R_1	6 **SRA** IR_1	16/18 **CPIJNE** Ir_1,r_2,RA	12 **LDC*** r_2,Irr_1	20 **CALL** IA_1		10 **LD** IR_1,IM	6 **LD** Ir_1,r_2	↓	↓	↓	↓	↓	↓	↓	6 **SCF**
E	6 **RR** R_1	6 **RR** IR_1	16 **LDCD*** r_1,Irr_2	16 **LDCI*** r_1,Irr_2	10 **LD** R_2,R_1	10 **LD** IR_2,R_1	10 **LD** R_1,IM	18 **LDC*** r_1,Irr_2,xs	↓	↓	↓	↓	↓	↓	↓	6 **CCF**
F	8 **SWAP** R_1	8 **SWAP** IR_1	16 **LDCPD*** r_2,Irr_1	16 **LDCPI*** r_2,Irr_1	18 **CALL** IRR_1	10 **LD** R_2,IR_1	18 **CALL** DA_1	18 **LDC*** r_2,Irr_1,xs	↓	↓	↓	↓	↓	↓	↓	6 **NOP**

NOTE A

16/18 **BTJRF** r_2,b,RA	16/18 **BTJRT** r_2,b,RA

NOTE B

8 **BITR** r_1,b	8 **BITS** r_1,b

NOTE C

6 **SRP** IM	6 **SRP0** IM	6 **SRP1** IM

NOTE D

20 **LDC*** r_1,Irr_2,xL	20 **LDC*** r_1,DA_2

NOTE E

20 **LDC*** r_2,Irr_2,xL	20 **LDC*** r_2,DA_1

Legend:
r = 4-bit address
R = 8-bit address
b = bit number
R_1 or r_1 = dst address
R_2 or r_2 = src address

***Examples:**
BOR r_0-R_2
is BOR r_1,b,R_2
or BOR r_2,b,R_1
LDC r_1,Irr_2
is LDC r_1,Irr_2 = program
or LDE r_1,Irr_2 = data

Sequence:
Opcode, first, second, third operands

NOTE: The blank areas are not defined.

Figure 5-2. Super8 Opcode Map

ADC
Add With Carry

ADC dst,src

Operation: dst <-- dst + src + c

The source operand, along with the setting of the Carry flag, is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Twos-complement addition is performed. In multiple precision arithmetic, this instruction permits the carry from the addition of low-order operands to be carried into the addition of high-order operands.

Flags:

C: Set if there is a carry from the most significant bit of the result; cleared otherwise.
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
D: Always cleared
H: Set if there is a carry from the most significant bit of the low-order four bits of the result; cleared otherwise.

Instruction Format:

			Cycles	Opcode (Hex)	Addressing Mode dst	src
Opcode	dst | src		6	12 13	r r	r Ir
Opcode	src	dst	10	14 15*	R R	R IR
Opcode	dst	src	10	16	R	IM

*This format is used in the example.

Example:

If the register named SUM contains %16, the Carry flag is set to 1, working register 10 contains %20 (32 decimal), and register 32 contains %10, the statement

```
ADC SUM, @R10
```

leaves the value %27 in register SUM.

AND
Logical

AND dst,src

Operation: dst ◄-- dst AND src

The source operand is logically ANDed with the destination operand. The result is stored in the destination. The AND operation results in a 1 bit being stored whenever the corresponding bits in the two operands are both 1s; otherwise a 0 bit is stored. The contents of the source are unaffected.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Always cleared to 0.
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

				Opcode	Addressing Mode	
			Cycles	(Hex)	dst	src
Opcode	dst | src		6	52	r	r
				53	r	Ir
Opcode	src	dst	10	54	R	R
				55	R	IR
Opcode	dst	src	10	56*	R	IM

*This format is used in the example.

Example:

If the source operand is the immediate value %7B (01111011) and the register named TARGET contains %C3 (11000011), the statement

```
AND TARGET, #%7B
```

leaves the value %43 (01000011) in register TARGET.

BAND
Bit And

BAND dst,src,b
BAND dst,b,src

Operation:

dst(0) ← dst(0) AND src(b)
or
dst(b) ← dst(b AND src(0)

The specified bit of the source (or the destination) is logically ANDed with bit 0 of the destination (or source). The resultant bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Undefined
S: 0
H: Unaffected
D: Unaffected

Instruction Format:

					Cycles	Opcode (Hex)	Addressing Mode dst	src
Opcode	dst	b	0	src	10	67*	r_0	R_b
Opcode	src	b	1	dst	10	67	R_b	r_0

*This format is used in the example.

Example:

If the register named BYTE contains %73 (01110011) and working register 3 contains %01, the statement

```
BAND R3,BYTE,#7
```

leaves the value %00 in working register 3.

BCP
Bit Compare

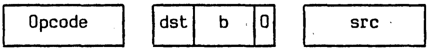
BCP dst,src,b

Operation: dst(0) - src(b)

The specified bit of the source is compared to (subtracted from) bit 0 of the destination. The Zero flag is set if the bits are the same; otherwise it is cleared. The contents of both operands are unaffected by the comparison.

Flags:
C: Unaffected
Z: Set if the two bits are the same; cleared otherwise.
V: Undefined
S: 0
H: Unaffected
D: Unaffected

Instruction Format:



Cycles	Opcode (Hex)	Addressing Mode dst	src
10	17	r_0	R_b

Example:

If working register 3 contains %01 and register 64 (%40) contains %FF, the statement

```
BCP R3,64,#0
```

sets the Zero flag bit in Flag register R213.

BITC
Bit Complement

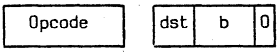
BITC dst,b

Operation: dst(b) ←-- NOT dst(b)

This instruction complements the specified bit within the destination without affecting any other bits in the destination.

Flags:
C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Undefined
S: 0
H: Unaffected
D: Unaffected

Instruction Format:



Cycles	Opcode (Hex)	Addressing Mode dst
8	57	r_b

Example:

If working register 3 contains %FF, the statement

```
BITC R3,#7
```

leaves the value %7F in that register.

BITR
Bit Reset

BITR dst,b

Operation: dst(b) ◄-- 0

This instruction clears the specified bit within the destination without affecting any other bits in the destination.

Flags: No flags affected

Instruction Format:

				Cycles	Opcode (Hex)	Addressing Mode dst
Opcode	dst	b	0	8	77	r_b

Example:

If working register 3 contains %80, the statement

    BITR R3,#7

leaves the value %00 in that register.

BITS
Bit Set

BITS dst,b

Operation: dst(b) ◄-- 1

This instruction sets the specified bit within the destination without affecting any other bits in the destination.

Flags: No flags affected

Instruction Format:

				Cycles	Opcode (Hex)	Addressing Mode dst
Opcode	dst	b	1	8	77	r_b

Example:

If working register 3 contains %00, the statement

    BITS R3,#7

leaves the value %80 in that register.

BOR
Bit OR

BOR dst,src,b
BOR dst,b,src

Operation: dst(0) ◄-- dst(0) OR src(b)
or
dst(b) ◄-- dst(b) OR src(0)

The specified bit of the source (or the destination) is logically ORed with bit 0 of the destination (or the source). The resultant bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags:
C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Undefined
S: 0
H: Unaffected
D: Unaffected

Instruction Format:

						Opcode	Addressing Mode	
					Cycles	(Hex)	dst	src
Opcode	dst	b	0	src	10	07	r_0	R_b
Opcode	src	b	1	dst	10	07*	R_b	r_0

*This format is used in the example.

Example:

If register 32 (%20) contains %0F and working register 3 contains %01, the statement

```
BOR 32,#7,R3
```

leaves the value %8F in register 32.

BTJRF
Bit Test and Jump Relative on False

BTJRF dst,src,b

Operation: If src(b) is a 0, PC ◄-- PC + dst

The specified bit within the source operand is tested. If it is a 0, the relative address is added to the Program Counter and control passes to the statement whose address is now in the PC; otherwise the instruction following the BTJRF instruction is executed.

Flags: No flags affected

Instruction Format:

Opcode	src	b	0	dst

Cycles	Opcode (Hex)	Addressing Mode dst	src
16/18*	37	RA	r_b

* 18 if jump taken, 16 if not

Example:

If working register 6 contains %7F,the statement

```
BTJRF SKIP,R6,#7
```

causes the Program Counter to jump to the memory location pointed to by SKIP. The memory location must be within the allowed range of +127,-128.

BTJRT
Bit Test and Jump Relative on True

BTJRT dst,src,b

Operation: If src(b) is a 1, PC ◄-- PC + dst

The specified bit within the source operand is tested. If it is a 1, the relative address is added to the Program Counter and control passes to the statement whose address is now in the PC; otherwise the instruction following the BTJRT instruction is executed.

Flags: No flags affected

Instruction Format:

Opcode	src	b	1	dst

Cycles	Opcode (Hex)	Addressing Mode dst	src
16/18*	37	RA	r_b

* 18 if jump taken, 16 if not

Example:

If working register 6 contains %80, the statement

```
BTJRT $+8,R6,#7
```

causes the next five bytes in memory to be skipped.

Note:

The $ refers to the address of the first byte of the instruction currently being executed.

BXOR
Bit XOR

BXOR dst,src,b
BXOR dst,b,src

Operation:

```
dst(0) ◄-- dst(0) XOR src(b)
  or
dst(b) ◄-- dst(b) XOR src(0)
```

The specified bit of the source (or the destination) is logically EXCLUSIVE ORed with bit 0 of the destination (or source). The resultant bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Undefined
S: 0
H: Unaffected
D: Unaffected

Instruction Format:

					Cycles	Opcode (Hex)	Addressing Mode	
							dst	src
Opcode	dst	b	0	src	10	27*	r_0	R_b
Opcode	src	b	1	dst	10	27	R_b	r_0

*This format is used in the example.

Example:

If working register 6 contains %FF and working register 7 contains %F0, the statement

```
BXOR R6,R7,#4
```

leaves the value %FE in working register 6.

CALL
Call Procedure

CALL dst

Operation:

SP ◄-- SP - 1
@SP ◄-- PCL
SP ◄-- SP - 1
@SP ◄-- PCH
PC ◄-- dst

The current contents of the Program Counter are pushed onto the top of the stack. The Program Counter value used is the address of the first instruction following the CALL instruction. The specified destination address is then loaded into the Program Counter and points to the first instruction of a procedure.

At the end of the procedure the Return (RET) instruction can be used to return to the original program flow. RET pops the top of the stack back into the Program Counter.

Flags: No flags affected

Instruction Format:

		Cycles	Opcode (Hex)	Addressing Mode dst
Opcode	dst	18	F6	DA
Opcode	dst	18	F4	IRR
Opcode	dst	20	D4	IA

Examples:

(1) If the contents of the Program Counter are %1A47 and the contents of the Stack Pointer (control registers 216-217) are %3002, the statement

CALL %3521

causes the Stack Pointer to be decremented to %3000, %1A4A (the address following the instruction) to be stored in external data memory locations %3000 and %3001 (%4A in %30001, %1A in %3000), and the Program Counter to be loaded with %3521. The Program Counter now points to the address of the first statement in the procedure to be executed.

(2) If the contents of the Program Counter and Stack Pointer are the same as in Example 1, working register 6 contains %35, and working register 7 contains %21, the statement

CALL @RR6

produces the same result as Example 1 except that %49 is stored in external data memory location %3000.

(3) If the contents of the Program Counter and Stack Pointer are the same as in Example 1, address %0040 contains %35, and address %0041 contains %21, the statement

CALL #%40

produces the same result as Example 2.

ADD
Add

ADD dst,src

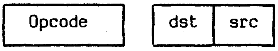
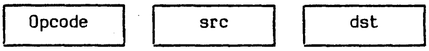
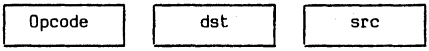
Operation: dst <-- dst + src

The source operand is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Twos-complement addition is performed.

Flags:

C: Set if there was a carry from the most significant bit of the result; cleared otherwise.
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if both operands were of the same sign and the result is of the opposite sign; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
H: Set if a carry from the low-order nibble occurred.
D: Always cleared to 0.

Instruction Format:

	Cycles	Opcode (Hex)	Addressing Mode dst	src
	6	02 03	r r	r Ir
	10	04* 05	R R	R IR
	10	06	R	IM

*This format is used in the example.

Example:

If the register named SUM contains %44 and the register named AUGEND contains %11, the statement

```
ADD SUM, AUGEND
```

leaves the value %55 in Register SUM.

CCF
Complement Carry Flag

CCF

Operation: C ◄-- NOT C

The Carry flag is complemented; if C = 1, it is changed to C = 0, and vice-versa.

Flags: **C:** Complemented

No other flags affected

Instruction Format:

	Cycles	Opcode (Hex)
Opcode	6	EF

Example:

If the Carry flag contains a 0, the statement

```
CCF
```

changes the 0 to 1.

CLR
Clear

CLR dst

Operation: dst ◄-- 0

The destination location is cleared to 0.

Flags: No flags affected

Instruction Format:

		Cycles	Opcode (Hex)	Addressing Mode dst
Opcode	dst	6	B0*	R
			B1	IR

*This format is used in the example.

Example:

If working register 6 contains %AF, the statement

```
CLR R6
```

leaves the value 0 in that register.

COM
Complement

COM dst

Operation: dst ◄-- NOT dst

The contents of the destination location are complemented (ones complement); all 1 bits are changed to 0, and vice-versa.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Always reset to 0
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

Opcode	dst

Cycles	Opcode (Hex)	Addressing Mode dst
6	60*	R
	61	IR

*This format is used in the example.

Example:

If working register 8 contains %24 (00100100), the statement

```
COM R8
```

leaves the value %DB (11011011) in that register.

CP
Compare

CP dst,src

Operation: dst - src

The source operand is compared to (subtracted from) the destination operand, and the appropriate flags are set accordingly. The contents of both operands are unaffected by the comparison.

Flags:
C: Set if a "borrow" occurred (src > dst); cleared otherwise.
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurred, cleared otherwise.
S: Set if the result is negative; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

				Opcode	**Addressing Mode**	
			Cycles	**(Hex)**	**dst**	**src**
Opcode	dst src		6	A2	r	r
				A3	r	Ir
Opcode	src	dst	10	A4	R	R
				A5*	R	IR
Opcode	dst	src	10	A6	R	IM

*This format is used in the example.

Example:

If the register named TEST contains %63, working register 0 contains %30 (48 decimal), and register 48 contains %63, the statement

```
CP TEST, @R0
```

sets (only) the Z flag. If this statement is followed by "JP EQ, true_routine," the jump will be taken.

DA
Decimal Adjust

DA dst

Operation: dst ◄-- DA dst

The destination operand is adjusted to form two 4-bit BCD digits following an addition or subtraction operation. For addition (ADD, ADC) or subtraction (SUB, SBC), the following table indicates the operation performed:

Instruction	Carry Before DA	Bits 4-7 Value (Hex)	H Flag Before DA	Bits 0-3 Value (Hex)	Number Added To Byte	Carry After DA
	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
ADD	0	A-F	0	0-9	60	1
ADC	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
	0	0-9	0	0-9	00 = -00	0
SUB	0	0-8	1	6-F	FA = -06	0
SBC	1	7-F	0	0-9	A0 = -60	1
	1	6-F	1	6-F	9A = -66	1

The operation is undefined if the destination operand was not the result of a valid addition or subtraction of BCD digits.

Flags:

C: Set if there was a carry from the most significant bit; cleared otherwise (see table above).
Z: Set if the result is 0; cleared otherwise.
V: Undefined
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

Opcode	dst

Cycles	Opcode (Hex)	Addressing Mode dst
6	40*	R
	41	IR

*This format is used in the example.

Example:

If working register R0 contains %15 and working register R1 contains %27, the statements

```
ADD R1, R0
DAB R1
```

leave %42 in working register R1.

If addition is performed using the BCD values 15 and 27, the result should be 42. The sum is incorrect, however, when the binary representations are added in the destination location using standard binary arithmetic.

```
   0001  0101
 + 0010  0111
 ------------
   0011  1100 = %3C
```

The DA statement adjusts this result so that the correct BCD representation is obtained.

```
   0011  1100
 + 0000  0110
 ------------
   0100  0010 = 42
```

CPIJE
Compare Increment and Jump on Equal

CPIJE dst,src,RA

Operation:

```
If dst - src = zero, PC <-- PC + RA
  Ir <-- Ir + 1
```

The source operand is compared to (subtracted from) the destination operand. If the result is 0, the relative address is added to the Program Counter and control passes to the statement whose address is now in the Program Counter; otherwise the instruction following the CPIJE instruction is executed. In either case the source pointer is incremented by one before the next instruction.

Flags: No flags affected

Instruction Format:

Opcode	src	dst	RA

	Opcode	Addressing Mode	
Cycles	(Hex)	dst	src
16/18*	C2	r	Ir

* 18 if jump taken, 16 if not

Example:

If working register 3 contains %AA, working register 5 contains %10, and register %10 contains %AA, the statement

```
CPIJE R3,@R5,$
```

puts the value %11 in working register 5 and then executes the same instruction again.

CPIJNE
Compare Increment and Jump on Non Equal

CPIJNE dst,src,RA

Operation:

```
If dst - src ≠ zero, PC <-- PC + RA
  Ir <-- Ir + 1
```

The source operand is compared to (subtracted from) the destination operand. If the result is not 0, the relative address is added to the Program Counter and control passes to the statement whose address is now in the Program Counter; otherwise the instruction following the CPIJNE instruction is executed. In either case, the source pointer is incremented by one before the next instruction.

Flags: No flags affected

Instruction Format:

Opcode	src	dst	RA

	Opcode	Addressing Mode	
Cycles	(Hex)	dst	src
16/18*	D2	r	Ir

* 18 if jump taken, 16 if not

Example:

If working register 3 contains %AA, working register 5 contains %10, and register %10 contains %AA, the statement

```
CPIJNE R3,@R5,$
```

puts the value %11 in working register 5 and then executes the next instruction following this instruction.

Note:

The $ refers to the address of the first byte of the instruction currently being executed.

DEC
Decrement

DEC dst

Operation: dst ◄-- dst - 1

The contents of the destination operand are decremented by one.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
S: Set if result is negative; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

Opcode	dst

Cycles	Opcode (Hex)	Addressing Mode dst
6	00*	R
	01	IR

*This format is used in the example.

Example:

If working register 10 contains %2A, the statement

```
DEC R10
```

leaves the value %29 in that register.

DECW
Decrement Word

DECW dst

Operation: dst ◄-- dst - 1

The contents of the destination location (which must be an even address) and the operand following that location are treated as a single 16-bit value which is decremented by one.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

Opcode	dst

Cycles	Opcode (Hex)	Addressing Mode dst
10	80	RR
	81*	IR

*This format is used in the example.

Example:

If working register 0 contains %30 (48 decimal) and registers 48-49 contain the value %FAF3, the statement

```
DECW @R0
```

leaves the value %FAF2 in registers 48 and 49.

DI
Disable Interrupts

DI

Operation: SMR (0) ◄-- 0

Bit 0 of control register 222 (the System Mode register) is cleared to 0. All interrupts are disabled; they can still set their respective interrupt status latches, but the CPU will not directly service them.

Flags: No flags affected

Instruction Format:

Opcode

Cycles	Opcode (Hex)
6	8F

Example:

If control register 222 contains %01, that is, interrupts are enabled, the statement

```
DI
```

sets control register 222 to %00, disabling all interrupts.

Divide (Unsigned)

DIV dst,src

Operation:

dst ÷ src
dst (UPPER) ←- REMAINDER
dst (LOWER) ←- QUOTIENT

The destination operand (16 bits) is divided by the source operand (8 bits). The quotient (8 bits) is stored in the lower half of the destination. The remainder (8 bits) is stored in the upper half of the destination. When the quotient is $\geq 2^8$, the numbers stored in the upper and lower halves of the destination for quotient and remainder are incorrect. Both operands are treated as unsigned integers.

Flags:

C: Set if V is set and quotient is between 2^8 and $2^9 - 1$; cleared otherwise.
Z: Set if divisor or quotient = 0; cleared otherwise.
V: Set if quotient is $\geq 2^8$ or divisor = 0; cleared otherwise.
S: Set if MSB of quotient = 1; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

Opcode	src	dst

	Opcode	**Addressing Mode**	
Cycles	**(Hex)**	**dst**	**src**
28/12*	94**	RR	R
28/12*	95	RR	IR
28/12*	96	RR	IM

* 12 if divide by zero is attempted
** This format is used in the example

Example:

If working register pair 6-7 (dividend) contains %10 in register 6 and %03 in register 7, and working register 4 (divisor) contains %40, the statement

```
DIV RR6,R4
```

leaves the value %40 in working register 7 (quotient) and the value %03 in working register 6 (remainder).

DJNZ
Decrement and Jump if Nonzero

DJNZ r,dst

Operation:

r ◄-- r -1
  If r ≠ 0, PC ◄-- PC + dst

The working register being used as a counter is decremented. If the contents of the register are not 0 after decrementing, the relative address is added to the Program Counter and control passes to the statement whose address is now in the Program Counter. The range of the relative address is +127 to -128, and the original value of the Program Counter is taken to be the address of the instruction byte following the DJNZ statement. When the working register counter reaches zero, control falls through to the statement following the DJNZ statement.

Flags: No flags affected

Instruction Format:

		Cycles	Opcode (Hex)	Addressing Mode dst
r | Opcode	dst	12 if jump taken 10 if jump not taken	rA r = 0 to F	RA

Example:

DJNZ is typically used to control a "loop" of instructions. In this example, 12 bytes are moved from one buffer area in the register file to another. The steps involved are:

o Load 12 into the counter (working register 6)
o Set up the loop to perform the moves
o End the loop with DJNZ

```
      LD R6,#12          !Load Counter!
LOOP: LD R9,OLDBUF (R6)  !Move one byte to!
      LD NEWBUF (R6),R9  !New location!
      DJNZ R6,LOOP       !Decrement and !
                         !Loop until counter = 0!
```

Note:

The working register being used as a counter must be one of the registers 00-CF. Using one of the I/O ports, control or peripheral registers will have undefined results.

EI
Enable Interrupts

EI

Operation:

```
SMR (0) ◄-- 1
```

Bit 0 of control register 220 (the System Mode register) is set to 1. This allows any interrupts to be serviced when they occur (assuming they have highest priority) or, if their respective interrupt status latch was previously enabled by its interrupt, then its interrupt can also be serviced.

Flags: No flags affected

Instruction Format:

	Cycles	Opcode (Hex)
Opcode	6	9F

Example:

If control register 222 contains %00, (i.e., interrupts are disabled), the statement

```
EI
```

sets control register 222 to %01, enabling all interrupts.

ENTER
Enter

ENTER

Operation:

```
SP  ◄-- SP - 2
@SP ◄-- IP
IP  ◄-- PC
PC  ◄-- @IP
IP  ◄-- IP + 2
```

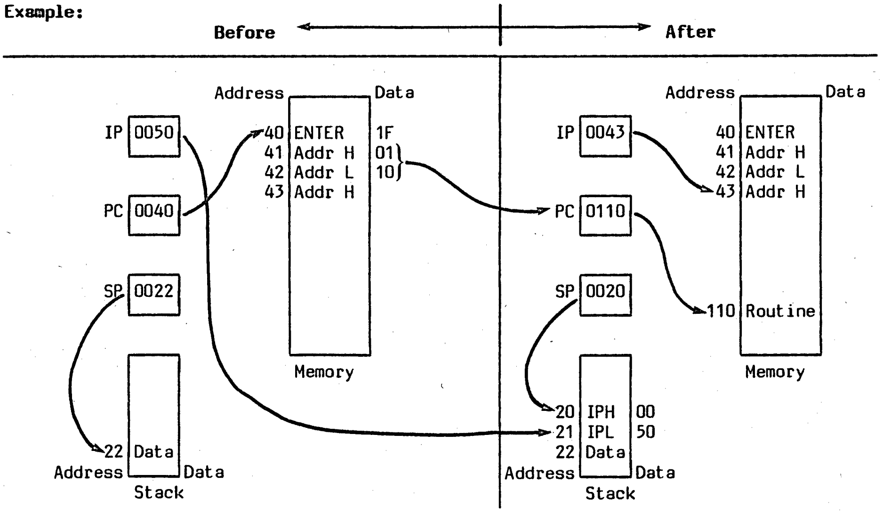
This instruction is useful for the implementation of threaded-code languages. The contents of the Instruction Pointer are pushed onto the stack. The value in the Program Counter is then transferred to the Instruction Pointer. The program memory word pointed to by the Instruction Pointer is loaded into the Program Counter. The Instruction Pointer is then incremented by two.

Flags: No flags affected

Instruction Format:

	Cycles	Opcode (Hex)
Opcode	20	1F

Example:



EXIT
Exit

EXIT

Operation:

```
IP ◄-- @SP
SP ◄-- SP + 2
PC ◄-- @IP
IP ◄-- IP + 2
```

This instruction is useful for the implementation of threaded-code languages. The stack is POPed and the Instruction Pointer is loaded. The program memory word pointed to by the Instruction Pointer is loaded into the Program Counter. The Instruction Pointer is then incremented by two.

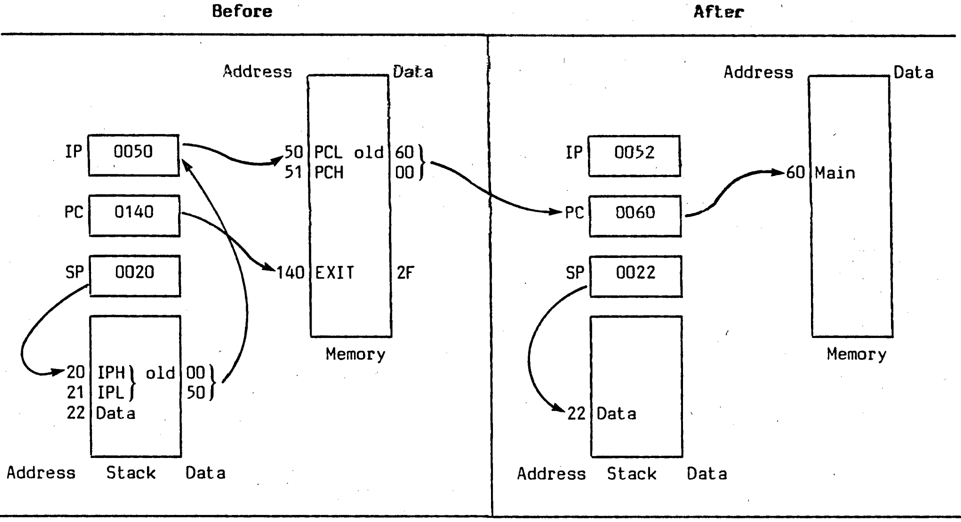
Flags: No flags affected

Instruction Format:

	Cycles	Opcode (Hex)
Opcode	22	2F

EXIT
Exit (Continued)

Example:



Note:

The examples for ENTER, EXIT, and NEXT illustrate how these instructions could actually be used together in a program.

INC
Increment

INC dst

Operation: dst ◄-- dst + 1

The contents of the destination operand are incremented by one.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

Format	Cycles	Opcode (Hex)	Addressing Mode dst
dst | Opcode	6	rE* r = 0 to F	r
Opcode | dst	6	20 21	R IR

*This format is used in the example.

Example:

If working register 10 contains %2A, the statement

INC R10

leaves the value %2B in that register.

INCW
Increment Word

INCW dst

Operation: dst ◄-- dst + 1

The contents of the destination (which must be an even address) and the byte following that location are treated as a single 16-bit value which is incremented by one.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

		Cycles	Opcode (Hex)	Addressing Mode dst
Opcode	dst	10	A0*	RR
			A1	IR

*This format is used in the example.

Example:

If working register pair 0-1 contains the value %FAF3, the statement

```
INCW RR0
```

leaves the value %FAF4 in working register pair 0-1.

IRET
Interrupt Return

	IRET (Normal)	IRET (Fast)

Operation:	Flags ← @SP
	SP ← SP + 1
	PC ← @SP
	SP ← SP + 2
	SYM(0) ← 1
		PC ↔ IP
		Flag ← Flag'
		FIS ← 0

This instruction is issued at the end of an interrupt service routine. It restores the Flag register and the Program Counter. It also reenables global interrupts.

Normal IRET is executed only if the Fast Interrupt Status bit (FIS, bit 1 of the Flags register R213) is cleared. Fast IRET is executed if FIS is set, indicating that a fast interrupt is being serviced.

Flags: All flags are restored to original settings (before interrupt occurred).

Instruction Format:

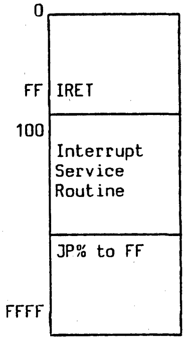
IRET (Normal)	Cycles	Opcode (Hex)
Opcode	16	BF

IRET (Fast)	Cycles	Opcode (Hex)
Opcode	6*	BF

*This format is used in the example.

Example:

In the figure below, the Instruction Pointer is initially loaded with %100 in the main program before interrupts are enabled. When an interrupt occurs, the Program Counter and Instruction Pointer are swapped. This causes the Program Counter to jump to address %100 and the Instruction Pointer to keep the return address. The last instruction in the service routine normally is a Jump to IRET at address %FF. This causes the Instruction Pointer to be loaded with %100 "again" and the Program Counter to jump back to the main program. Now the next interrupt can occur and the Instruction Pointer is still correct at %100.



Note:

For the Fast Interrupt example above, if the last instruction is not a Jump to IRET, then care must be taken with the order of the last two instructions. The instruction IRET cannot be immediately preceded by a clear of interrupt status (such as a reset of the Interrupt Pending register).

JP
Jump

JP cc,dst
JP dst

Operation: If cc is true, PC ◄-- dst

The conditional Jump transfers program control to the destination address if the condition specified by "cc" is true; otherwise, the instruction following the JP instruction is executed. See section 5.3 for a list of condition codes.

The unconditional Jump simply replaces the contents of the Program Counter with the contents of the specified register pair. Control then passes to the statement addressed by the Program Counter.

Flags: No flags affected

Instruction Format:

			Cycles	Opcode (Hex)	Addressing Mode dst
Conditional	cc | Opcode	dst	10/12*	ccD** cc = 0 to F	DA
Unconditional	Opcode	dst	10	30	IRR

*12 if jump taken, 10 if not
**This format is used in the example.

Example:

If the Carry flag is set to 1, the statement

```
JP C,%1520
```

replaces the contents of the Program Counter with %1520 and transfers control to that location. Had the Carry flag not been set, control would have fallen through to the statement following the JP.

JR
Jump Relative

JR cc,dst

Operation: If cc is true, PC ◄-- PC + dst

If the condition specified by "cc" is true, the relative address is added to the Program Counter and control passes to the statement whose address is now in the Program Counter; otherwise, the instruction following the JR instruction is executed. (See section 5.3 for a list of condition codes.) The range of the relative address is +127, -128, and the original value of the Program Counter is taken to be the address of the first instruction byte following the JR statement.

Flags: No flags affected

Instruction Format:

	Cycles	Opcode (Hex)	Addressing Mode dst
cc Opcode dst	10/12*	ccB cc = 0 to F	RA

* 12 if jump taken, 10 if not

Example:

If the result of the last arithmetic operation executed is negative, then the four following statements (which occupy a total of seven bytes) are skipped with the statement

```
JR MI,$+9
```

If the result is not negative, execution continues with the statement following the JR. A short form of a jump to label LO is

```
JR LO
```

where LO must be within the allowed range. The condition code is "blank" in this case, and JR has the effect of an unconditional JP instruction.

Note:

The $ refers to the address of the first byte of the instruction currently being executed.

LD
Load

LD dst,src

Operation: dst ◄-- src

The contents of the source are loaded into the destination. The contents of the source are unaffected.

Flags: No flags affected

Instruction Format:

				Opcode	Addressing Mode	
			Cycles	(Hex)	dst	src
dst Opcode	src		6	rC	r	IM
			6	r8	r	R
src Opcode	dst		6	r9 r=0 to F	R	r
Opcode	dst src		6	C7	r	Ir
			6	D7	Ir	r
Opcode	src	dst	10	E4	R	R
			10	E5	R	IR
Opcode	dst	src	10	E6	R	IM
			10	D6	IR	IM
Opcode	src	dst	10	F5	IR	R
Opcode	dst src	x	10	87	r	x(r)
Opcode	src dst	x	10	97*	x(r)	r

*This format is used in the example.

Example:

If working register 0 contains %08 (11 decimal) and working register 10 contains %83, the statement

LD 240(R0),R10

loads the value %83 into register 251 (240 +11). The contents of working register 10 are unaffected by the load.

LDB
Load Bit

LDB dst,src,b
LDB dst,b,src

Operation:

dst(0) ← src(b)
or
dst(b) ← src(0)

The specified bit of the source is loaded into bit 0 of the destination, or bit 0 of the source is loaded into the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags: No flags affected

Instruction Format:

				Opcode	Addressing Mode	
			Cycles	(Hex)	dst	src
Opcode	dst | b | 0	src	10	47	r_0	R_b
Opcode	src | b | 1	dst	10	47	R_b	r_0

Example:

If working register 3 contains %00 and working register 5 contains %FF, the statement

```
LDB R3,R5,#7
```

leaves the value %01 in working register 3.

LDE/LDC
Load Memory

LDE/LDC dst,src

Operation: dst ◄-- src

This instruction is used to load a byte from program or data memory into a working register or vice-versa. The contents of the source are unaffected.

Flags: No flags affected

Instruction Format:

					Cycles	Opcode (Hex)	Addressing Mode dst	Addressing Mode src	
Opcode	dst	src			12	C3	r	Irr	
Opcode	src	dst			12	D3**	Irr	r	
Opcode	dst	src	xs		18	E7	r	xs(rr)	
Opcode	src	dst	xs		18	F7	xs(rr)	r	
Opcode	dst	src*	$x1_L$	$x1_H$	20	A7	r	xl(rr)	
Opcode	src	dst*	$x1_L$	$x1_H$	20	B7	x1(rr)	r	
Opcode	dst	0000	DA_L	DA_H	20	A7	r	DA	Program Memory
Opcode	src	0000	DA_L	DA_H	20	B7	DA	r	
Opcode	dst	0001	DA_L	DA_H	20	A7	r	DA	Data Memory
Opcode	src	0001	DA_L	DA_H	20	B7	DA	r	

*The src or (rr) cannot use register pair 0-1.
**This format is used in the example.

Example:

If the working register pair 6-7 contains %404A and working register 2 contains %22, the statement

```
LDE @RR6,R2
```

will load the value %22 into data memory location %404A.

Note:

LDE refers to data memory.
LDC refers to program memory.

The assembler makes Irr or rr even for program memory and odd for data memory. In the example above, the assembler produces this code: D3 27.

LDED/LDCD
Load Memory and Decrement

LDED/LDCD dst,src

Operation:

dst ◄-- src
rr ◄-- rr -1

This instruction is used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then decremented. The contents of the source are unaffected.

Flags: No flags affected

Instruction Format:

				Opcode	Addressing Mode	
			Cycles	(Hex)	dst	src
Opcode	dst	src	16	E2	r	Irr

Example:

If working register pair 6-7 contains %30A3 and data memory locations %30A2 and %30A3 contain %22BC, the statement

```
LDED R2, @RR6
```

loads the value %BC into working register 2 and the value %30A2 into working register pair 6-7. A second statement

```
LDED R2, @RR6
```

loads the value %22 into working register 2 and the value %30A1 into working register pair 6-7.

Note:

LDED refers to data memory.
LDCD refers to program memory.

The assembler makes Irr even for program memory and odd for data memory. In the example above, the assembler produces this code: E2 27.

This instruction is the equivalent of a POPUD with the stack in memory rather than in the register file.

LDEI/LDCI
Load Memory and Increment

LDEI/LDCI dst,src

Operation:

```
dst ◄-- src
rr  ◄-- rr + 1
```

This instruction is used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then incremented automatically. The contents of the source are unaffected.

Flags: No flags affected

Instruction Format:

			Addressing Mode	
	Cycles	Opcode (Hex)	dst	src
Opcode dst src	16	E3	r	Irr

Example:

If working register pair 6-7 contains %30A2 and program memory locations %30A2 and %30A3 contain %22BC, the statement

```
LDCI R2,@RR6
```

loads the value %22 into working register 2, and working register pair 6-7 is incremented to %30A3. A second

```
LDCI R2,@RR6
```

loads the value %BC into register 2, and working register pair 6-7 is incremented to %30A4.

Note:

LDEI refers to data memory.
LDCI refers to program memory.

The assembler makes Irr even for program memory and odd for data memory. In the example above, the assembler produces this code: E3 26.

This instruction is the equivalent of a POPUI with the stack in memory rather than the register file.

LDEPD/LDCPD
Load Memory with Pre-Decrement

LDEPD/LDCPD dst,src

Operation:

```
rr  <-- rr - 1
dst <-- src
```

This instruction is used for block transfers of data to program or data memory from the register file. The address of the memory location is specified by a working register pair and is first decremented. The contents of the source location are loaded into the destination location. The contents of the source are unaffected.

Flags: No flags affected

Instruction Format:

				Opcode	Addressing Mode	
			Cycles	(Hex)	dst	src
Opcode	src	dst	16	F2	Irr	r

Example:

If working register pair 6-7 contains %404B and working register 2 contains %22 (34 decimal), the statement

```
LDEPD @RR6,R2
```

loads the value %22 into data memory location %404A and the value %404A into working register pair 6-7.

Note:

LDEPD refers to data memory.
LDCPD refers to program memory.

The assembler makes Irr even for program memory and odd for data memory.

This instruction is the equivalent of a PUSHUD with the stack in memory rather than the register file.

LDEPI/LDCPI
Load Memory with Pre-Increment

LDEPI/LDCPI dst,src

Operation:

```
rr  <-- rr + 1
dst <-- src
```

This instruction is used for block transfers of data to program or data memory from the register file. The address of the memory location is specified by a working register pair and is first incremented. The contents of the source location are loaded into the destination location. The contents of the source are unaffected.

Flags: No flags affected

Instruction Format:

Opcode		src	dst

Cycles	Opcode (Hex)	Addressing Mode dst	Addressing Mode src
16	F3	Irr	r

Example:

If working register pair 6-7 contains %404A and working register 2 contains %22 (34 decimal), the statement

```
LDEPI @RR6,R2
```

loads the value %22 into external data memory location %404B and the value %404B into working register pair 6-7.

Note:

LDEPI refers to data memory.
LDCPI refers to program memory.

The assembler makes Irr even for program memory and odd for data memory.

This instruction is the equivalent of a PUSHUI with the stack in memory rather than the register file.

LDW
Load Word

LDW dst,src

Operation: dst ◄-- src

The contents of the source (a word) are loaded into the destination. The contents of the source are unaffected.

Flags: No flags affected

Instruction Format:

			Cycles	Opcode (Hex)	Addressing Mode dst	src
Opcode	src	dst	10 10	C4 C5	RR RR	RR IR
Opcode	dst	src	12	C6*	RR	IML

*This format is used in the example.

Example:

If the source operand is the immediate value %5AA5, the statement

```
LDW RR6,#%5AA5
```

leaves the value %5A in working register 6 and the value %A5 in working register 7.

MULT
Multiply (Unsigned)

MULT dst,src

Operation: dst ◄-- dst x src

The 8-bit destination operand (even register of the register pair) is multiplied by the source operand (8 bits) and the product (16 bits) is stored in the register pair specified by the destination address. Both operands are treated as unsigned integers.

Flags:

C: Set if result is > 255; cleared otherwise.
Z: Set if the result is 0; cleared otherwise.
V: Cleared
S: Set if MSB of the result is a 1; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

			Cycles	Opcode (Hex)	Addressing Mode dst	src
Opcode	src	dst	24 24 24	84* 85 86	RR RR RR	R IR IM

*This format is used in the example.

Example:

If working register 6 contains %40 (64 decimal) and working register 4 contains %42 (66 decimal), the statement

```
MULT RR6, R4
```

leaves the value %10 in working register 6 and %80 in working register 7 (%1080 is 4224 decimal).

NEXT

Operation:

PC ◄-- @IP
IP ◄-- IP + 2

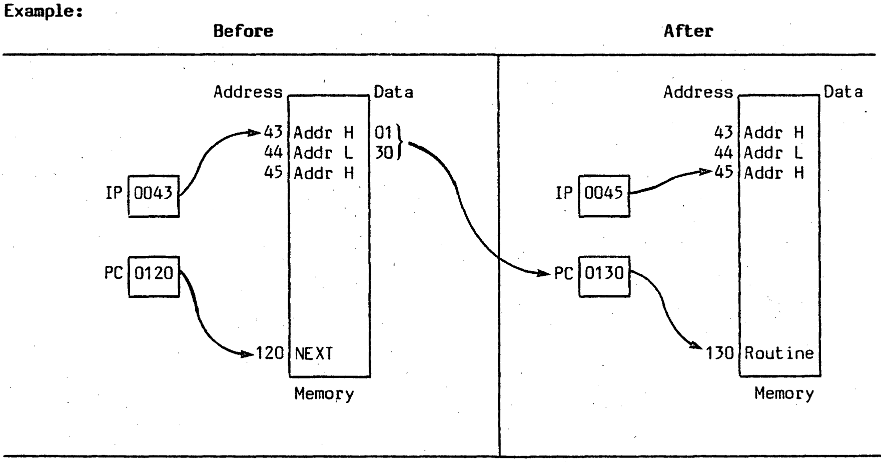
This instruction is useful for the implementation of threaded-code languages. The program memory word pointed to by the Instruction Pointer is loaded into the Program Counter. The Instruction Pointer is then incremented by two.

Flags: No flags affected

Instruction Format:

Opcode	Cycles	Opcode (Hex)
	14	0F

Example:



Note:

The examples for ENTER, EXIT, and NEXT illustrate how they could actually be used together in a program.

NOP
No Operation

NOP

Operation:

No action is performed by this instruction. It is typically used for timing delays.

Flags: No flags affected

Instruction Format:

	Cycles	Opcode (Hex)
Opcode	6	FF

OR
Logical OR

OR dst,src

Operation: dst ◄-- dst OR src

The source operand is logically ORed with the destination operand and the result is stored in the destination. The contents of the source are unaffected. The OR operation results in a 1 bit being stored whenever either of the corresponding bits in the two operands is 1; otherwise a 0 bit is stored.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Always cleared to 0
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

				Opcode	Addressing Mode	
			Cycles	(Hex)	dst	src
Opcode	dst src		6	42	r	r
			6	43	r	Ir
Opcode	src	dst	10	44	R	R
			10	45	R	IR
Opcode	dst	src	10	46*	R	IM

*This format is used in the example.

Example:

If the source operand is the immediate value %7B (01111011) and the register named TARGET contains %C3 (11000011), the statement

```
OR TARGET, #%7B
```

leaves the value %FB (11111011) in register TARGET.

POP
Pop

POP dst

Operation:

```
dst <-- @SP
SP  <-- SP + 1
```

The contents of the location addressed by the Stack Pointer are loaded into the destination. The Stack Pointer is then incremented by one.

Flags: No flags affected

Instruction Format:

Opcode	dst

Cycles	Opcode (Hex)	Addressing Mode dst
10	50	R
10	51*	IR

*This format is used in the example.

Example:

If the Stack Pointer (control registers 216-217) contains %1000, external data memory location %1000 contains %55, and working register 6 contains %22 (34 decimal), the statement

```
POP @R6
```

loads the value %55 into register 34. After the POP operation, the Stack Pointer contains %1001.

POPUD
Pop User Stack (Decrementing)

POPUD dst,src

Operation:

```
dst <-- src
IR  <-- IR - 1
```

This instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user Stack Pointer are loaded into the destination. The user Stack Pointer is then decremented.

Flags: No flags affected

Instruction Format:

Opcode	src	dst

Cycles	Opcode (Hex)	Addressing Mode dst	Addressing Mode src
10	92	R	IR

Example:

If the user Stack Pointer (register %42, for example) contains %80 and register %80 contains 5A, the statement

```
POPUD R2,@%42
```

loads the value %5A into working register 2. After the POP operation, the user Stack Pointer contains %7F.

POPUI
Pop User Stack (Incrementing)

POPUI dst,src

Operation:

```
dst ←- src
IR  ←- IR + 1
```

This instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user Stack Pointer are loaded into the destination. The user Stack Pointer is then incremented.

Flags: No flags affected

Instruction Format:

			Cycles	Opcode (Hex)	Addressing Mode dst	Addressing Mode src
Opcode	src	dst	10	93	R	IR

Example:

If the user Stack Pointer (register %42, for example) contains %80 and register %80 contains %5A, the statement

```
POPUI R2,@%42
```

loads the value %5A into working register 2. After the POP operation, the user Stack Pointer contains %81.

PUSH
Push

PUSH src

Operation:

```
SP  ←- SP - 1
@SP ←- src
```

The contents of the Stack Pointer are decremented, then the contents of the source are loaded into the location addressed by the decremented Stack Pointer, thus adding a new element to the top of the stack.

Flags: No flags affected

Instruction Format:

		Cycles	Opcode (Hex)	Addressing Mode src
Opcode	src	10 Internal stack 12 External stack	70*	R
		12 Internal stack 14 External stack	71	IR

*This format is used in the example.

Example:

If the Stack Pointer contains %1001, the statement

```
PUSH FLAGS
```

stores the contents of the register named FLAGS in location %1000. After the PUSH operation, the Stack Pointer contains %1000.

PUSHUD
Push User Stack (Decrementing)

PUSHUD dst,src

Operation:

```
IR  <-- IR - 1
dst <-- src
```

This instruction is used for user-defined stacks in the register file. The user Stack Pointer is decremented, then the contents of the source are loaded into the register file location addressed by the decremented user Stack Pointer.

Flags: No flags affected

Instruction Format:

				Opcode	Addressing Mode	
			Cycles	(Hex)	dst	src
Opcode	dst	src	10	82	IR	R

Example:

If the user Stack Pointer (%42, for example) contains %81, the statement

```
PUSHUD @%42,R2
```

stores the contents of working register 2 in location %80. After the PUSH operation, the user Stack Pointer contains %80.

PUSHUI
Push User Stack (Incrementing)

Push User Stack (Incrementing)

PUSHUI dst,src

Operation:

```
IR  <-- IR + 1
dst <-- src
```

This instruction is used for user-defined stacks in the register file. The user Stack Pointer is incremented, then the contents of the source are loaded into the register file location addressed by the incremented user Stack Pointer.

Flags: No flags affected

Instruction Format:

				Opcode	Addressing Mode	
			Cycles	(Hex)	dst	src
Opcode	dst	src	10	83	IR	R

Example:

If the user Stack Pointer (%42, for example) contains %81, the statement

```
PUSHUI @%42,R2
```

stores the contents of working register 2 in location %82. After the PUSH operation, the user Stack Pointer contains %82.

RCF
Reset Carry Flag

RCF

Operation: C ◄-- 0

The Carry flag is cleared to 0, regardless of its previous value.

Flags: **C:** Cleared to 0

No other flags affected

Instruction Format:

	Cycles	Opcode (Hex)
Opcode	6	CF

RET
Return

RET

Operation:

PC ◄-- @SP
SP ◄-- SP + 2

This instruction is normally used to return to the previously executing procedure at the end of a procedure entered by a CALL instruction. The contents of the location addressed by the Stack Pointer are popped into the Program Counter. The next statement executed is that addressed by the new contents of the Program Counter.

Flags: No flags affected

Instruction Format:

	Cycles	Opcode (Hex)
Opcode	14	AF

Example:

If the Program Counter contains %35B4, the Stack Pointer contains %2000, external data memory location %2000 contains %18, and location %2001 contains %B5, then the statement

```
RET
```

leaves the value %2002 in the Stack Pointer and %18B5, the address of the next instruction, in the Program Counter.

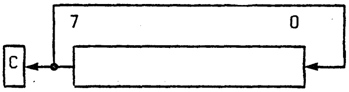
RL
Rotate Left

RL dst

Operation:

C ◄-- dst (7)
dst (0) ◄-- dst (7)
dst (n + 1) ◄-- dst (n) n = 0 - 6

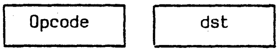
The contents of the destination operand are rotated left one bit position. The initial value of bit 7 is moved to the bit 0 position and also replaces the Carry flag.



Flags:

C: Set if the bit rotated from the most significant bit position was 1, i.e., bit 7 was 1.
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:



Cycles	Opcode (Hex)	Addressing Mode dst
6	90*	R
6	91	IR

*This format is used in the example.

Example:

If the contents of the register named SHIFTER are %88 (10001000), the statement

```
RL SHIFTER
```

leaves the value %11 (00010001) in that register and the Carry and Overflow flags are set to 1.

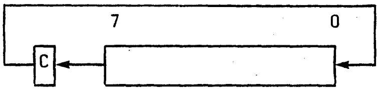
RLC
Rotate Left Through Carry

RLC dst

Operation:

dst (0) ←-- C
C ←-- dst (7)
dst (n + 1) ←-- dst (n) n = 0 - 6

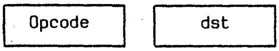
The contents of the destination operand with the Carry flag are rotated left one bit position. The initial value of bit 7 replaces the Carry flag; the initial value of the Carry flag replaces bit 0.



Flags:

C: Set if the bit rotated from the most significant bit position was 1, i.e., bit 7 was 1.
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:



Cycles	Opcode (Hex)	Addressing Mode dst
6	10*	R
6	11	IR

*This format is used in the example.

Example:

If the Carry flag is cleared to 0 and the register named SHIFTER contains %8F (10001111), the statement

```
RLC SHIFTER
```

sets the Carry and Overflow flags to 1 and leaves the value %1E (00011110) in SHIFTER.

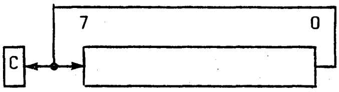
RR
Rotate Right

RR dst

Operation:

C ◄-- dst (0)
dst (7) ◄-- dst (0)
dst (n) ◄-- dst (n + 1) n = 0 - 6

The contents of the destination operand are rotated right one bit position. The initial value of bit 0 is moved to bit 7 and also replaces the Carry flag.



Flags:

C: Set if the bit rotated from the least significant bit position was 1, i.e., bit 0 was 1.
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

Opcode	dst

Cycles	Opcode (Hex)	Addressing Mode dst
6	E0*	R
6	E1	IR

*This format is used in the example.

Example:

If the contents of register 6 are %31 (00110001), the statement

RR R6

sets the Carry flag to 1 and leave the value %98 (10011000) in working register 6. Since bit 7 now equals 1, the Sign and Overflow flags are also set to 1.

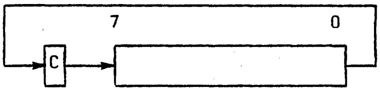
RRC
Rotate Right Through Carry

RRC dst

Operation:

```
dst (7) ◄-- C
C ◄-- dst (0)
dst (n) ◄-- dst (n + 1) n = 0 - 6
```

The contents of the destination operand and the Carry flag are rotated right one bit position. The initial value of bit 0 replaces the Carry flag; the initial value of the Carry flag replaces bit 7.



Flags:

C: Set if the bit rotated from the least significant bit position was 1, i.e., bit 0 was 1.
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination changed during rotation; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

Opcode	dst

Cycles	Opcode (Hex)	Addressing Mode dst
6	C0*	R
6	C1	IR

*This format is used in the example.

Example:

If the contents of the register named SHIFTER are %DD (11011101), and the Carry flag is cleared to 0, the statement

```
RRC SHIFTER
```

sets the Carry and Overflow flags to 1 and leaves the value %6E (01101110) in the register.

SB0
Set Bank 0

SB0

Operation: BANK ◄-- 0

This instruction causes the Bank Address flag (bit 0) of Flag register 213 to be cleared to 0.

Flags: No flags affected

Instruction Format:

Format	Cycles	Opcode (Hex)
Opcode	6	4F

SB1
Set Bank 1

SB1

Operation: BANK ◄-- 1

This instruction causes the Bank Address flag (bit 0) of Flag register 213 to be set to 1.

Flags: No flags affected

Instruction Format:

Format	Cycles	Opcode (Hex)
Opcode	6	5F

SBC
Subtract With Carry

SBC dst,src

Operation: dst <-- dst - src - C

The source operand, along with the setting of the Carry flag, is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the twos complement of the source operand to the destination operand. In multiple precision arithmetic, this instruction permits the carry ("borrow") from the subtraction of low-order operands to be subtracted from the subtraction of high-order operands.

Flags:

C: Set if a borrow occurred (src > dst); cleared otherwise.
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occured, that is, if the operands were of opposite sign and the sign of the result is the same as the sign of the source; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise, indicating a "borrow."
D: Always set to 1.

Instruction Format:

				Opcode	**Addressing Mode**	
			Cycles	**(Hex)**	**dst**	**src**
Opcode	dst src		6	32	r	r
			6	33*	r	Ir
Opcode	src	dst	10	34	R	R
			10	35	R	IR
Opcode	dst	src	10	36	R	IM

*This format is used in the example.

Example:

If the register named MINUEND contains %16, the Carry flag is set to 1, working register 10 contains %20 (32 decimal), and register 32 contains %05, the statement

```
SBC MINUEND, @R10
```

leaves the value %10 in register MINUEND.

SCF
Set Carry Flag

SCF

Operation: C <-- 1

The Carry flag is set to 1, regardless of its previous value.

Flags: **C:** Set to 1

No other flags affected

Instruction Format:

	Cycles	Opcode (Hex)
Opcode	6	DF

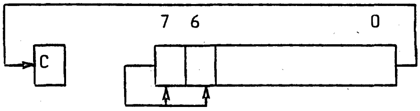
SRA
Shift Right Arithmetic

SRA dst

Operation:

dst (7) <-- dst (7)
C <-- dst (0)
dst (n) <-- dst (n + 1) n = 0 - 6

An arithmetic shift right one bit position is performed on the destination operand. Bit 0 replaces the Carry flag. Bit 7 (the sign bit) is unchanged, and its value is also shifted into bit position 6.



Flags:

C: Set if the bit shifted from the least significant bit position was 1, i.e., bit 0 was 1.
Z: Set if the result is 0; cleared otherwise.
V: Always cleared to 0
S: Set if the result is negative; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

		Cycles	Opcode (Hex)	Addressing Mode dst
Opcode	dst	6	D0*	R
		6	D1	IR

*This format is used in the example.

Example:

If the register named SHIFTER contains %B8 (10111000), the statement

```
SRA SHIFTER
```

clears the Carry flag to 0 and leaves the value %DC (11011100) in the register SHIFTER. The Sign flag is set to 1.

SRP/SRP0/SRP1
Set Register Pointer

SRP/SRP0/SRP1 src

Operation:

```
If src (1) = 1 and src (0) = 0 then:  RP0 (3-7) ◄-- src (3-7)

If src (1) = 0 and src (0) = 1 then:  RP1 (3-7) ◄-- src (3-7)

If src (1) = 0 and src (0) = 0 then:  RP0 (4-7) ◄-- src (4-7),
                                      RP0 (3) ◄-- 0
                                      RP1 (4-7) ◄-- src (4-7),
                                      RP1 (3) ◄-- 1
```

The source data bits 1 and 0 determine if one or both of the Register Pointers is to be written. Bits 3-7 of the selected Register Pointer are written unless both Register Pointers are selected. Then bit 3 of RP0 is forced to a 0 and bit 3 of RP1 is forced to a 1.

Flags: No flags affected

Instruction Format:

		Cycles	Opcode (Hex)	Addressing Mode src
Opcode	src	6	31	IM

Examples:

(1) The statement

SRP0 #%50

sets Register Pointer 0 (control register 214) to %50.
The assembler produces this code: 31 52.

(2) The statement

SRP1 #%68

sets Register Pointer 1 (control register 215) to %68.
The assembler produces this code: 31 69.

(3) The statement

SRP #%40

sets Register Pointer 0 to %40 and Register Pointer 1 to %48.
The assembler produces this code: 31 40.

SUB
Subtract

SUB dst,src

Operation: dst ◄-- dst - src

The source operand is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the twos complement of the source operand to the destination operand.

Flags:

C: Set if a "borrow" occurred; cleared otherwise.
Z: Set if the result is 0; cleared otherwise.
V: Set if arithmetic overflow occured, that is, if the operands were of opposite signs and the sign of the result is the same as the sign of the source operand; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a "borrow."
D: Always set to 1.

Instruction Format:

				Opcode	Addressing Mode	
			Cycles	(Hex)	dst	src
Opcode	dst | src		6	22	r	r
			6	23	r	Ir
Opcode	src	dst	10	24	R	R
			10	25	R	IR
Opcode	dst	src	10	26*	R	IM

*This format is used in the example.

Example:

If the register named MINUEND contains %29, the statement

SUB MINUEND, #%11

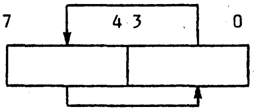
leaves the value %18 in the register.

SWAP
Swap Nibbles

SWAP dst

Operation: dst (0 - 3) <-> dst (4 - 7)

The contents of the lower four bits and upper four bits of the destination operand are swapped.



Flags:

C: Undefined
Z: Set if the result is 0; cleared otherwise.
V: Undefined
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

Opcode	dst

Cycles	Opcode (Hex)	Addressing Mode dst
8	F0*	R
8	F1	IR

*This format is used in the example.

Example:

If the register named BCD_Operands contains %B3 (10110011), then the statement

```
SWAP BDC_Operands
```

leaves the value %3B (00111011) in the register.

TCM
Test Complement Under Mask

TCM dst,src

Operation: (NOT dst) AND src

This instruction tests selected bits in the destination operand for a logical "1" value. The bits to be tested are specified by setting a 1 bit in the corresponding position of the source operand (mask). The TCM statement complements the destination operand, which is then ANDed with the source mask. The Zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Always cleared to 0.
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

				Opcode	**Addressing Mode**	
			Cycles	**(Hex)**	**dst**	**src**
Opcode	dst | src		6	62*	r	r
			6	63	r	Ir
Opcode	src	dst	10	64	R	R
			10	65	R	IR
Opcode	dst	src	10	66	R	IM

*This format is used in the example.

Example:

If the register named TESTER contains %F6 (11110110) and the register named MASK contains %06 (00000110), that is, bits 1 and 2 are being tested for a 1 value, then the statement

```
TCM TESTER, MASK
```

complements TESTER (to 00001001) and then does a logical AND with register MASK, resulting in %00. A subsequent test of the Z flag

```
JP Z, label
```

causes a transfer of program control. At the end of this sequence, TESTER still contains %F6.

TM
Test Under Mask

TM dst,src

Operation: dst AND src

This instruction tests selected bits in the destination operand for a logical "0" value. The bits to be tested are specified by setting a 1 bit in the corresponding position of the source operand (mask), which is ANDed with the destination operand. The Zero (Z) flag can then be checked to determine the result. The destination and source operands are unaffected.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Always reset to 0.
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:

		Opcode	Addressing Mode	
	Cycles	(Hex)	dst	src
Opcode · dst · src	6	72*	r	r
	6	73	r	Ir
Opcode · src · dst	10	74	R	R
	10	75	R	IR
Opcode · dst · src	10	76	R	IM

*This format is used in the example.

Example:

If the register named TESTER contains %F6 (11110110) and the register named MASK contains %06 (00000110), that is, bits 1 and 2 are being tested for a 0 value, then the statement

```
TM TESTER, MASK
```

results in the value %06 (00000110). A subsequent test for nonzero

```
JP NZ, label
```

causes a transfer of program control. At the end of this sequence, TESTER still contains %F6.

WFI

Operation:

The CPU is effectively halted until an interrupt occurs, except that DMA transfers still take place in the halt state. Either a fast interrupt or normal interrupt can take the CPU out of the halt state.

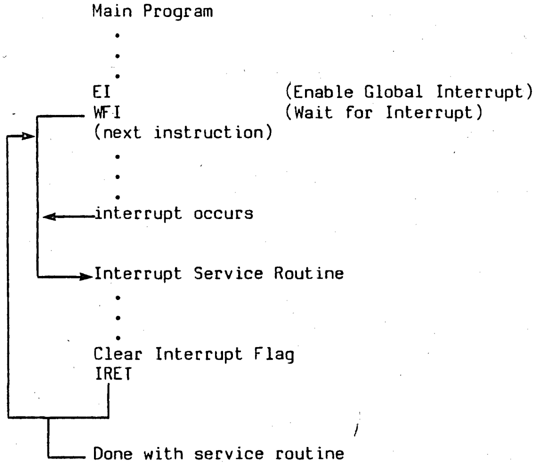
Flags: No flags affected

Instruction Format:

Format	Cycles	Opcode (Hex)
Opcode	6n	3F

n = 1,2,3,...

Example:



```
Main Program
  .
  .
  .
EI                    (Enable Global Interrupt)
WFI                   (Wait for Interrupt)
(next instruction)
  .
  .
  .
interrupt occurs

Interrupt Service Routine
  .
  .
  .
Clear Interrupt Flag
IRET

Done with service routine
```

XOR
Logical Exclusive OR

XOR dst,src

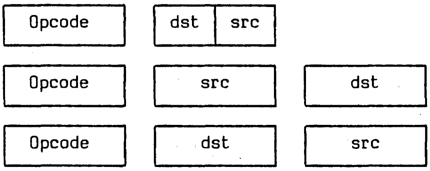
Operation: dst ◄-- dst XOR src

The source operand is logically EXCLUSIVE ORed with the destination operand and the result is stored in the destination. The EXCLUSIVE OR operation results in a 1 bit being stored whenever the corresponding bits in the operands are different; otherwise, a 0 bit is stored.

Flags:

C: Unaffected
Z: Set if the result is 0; cleared otherwise.
V: Always reset to 0.
S: Set if the result bit 7 is set; cleared otherwise.
H: Unaffected
D: Unaffected

Instruction Format:



Cycles	Opcode (Hex)	Addressing Mode dst	Addressing Mode src
6	B2	r	r
6	B3	r	Ir
10	B4	R	R
10	B5	R	IR
10	B6*	R	IM

*This format is used in the example.

Example:

If the source is the immediate value %7B (01111011) and the register named TARGET contains %C3 (11000011), the statement

```
XOR TARGET, #%7B
```

leaves the value %B8 (10111000) in the register.

Chapter 6
Interrupts

6.1 INTRODUCTION

The interrupt structure of the Super8 consists of 27 different interrupt sources, 16 vectors, and 8 levels (Figure 6-1). Two of the vectors are reserved for future members of the Super8 family.

Interrupt priority is assigned by level, which is controlled by the Interrupt Priority register (IPR). Each level is masked (or enabled) according to the bits in the Interrupt Mask register (IMR), and the entire interrupt structure can be disabled by clearing bit 0 in the System Mode register (R222). The three major components of the interrupt structure are sources, vectors, and levels.

A source is anything that generates an interrupt. This can be internal or external to the Super8. Internal sources are hardwired to a particular vector and level, while external sources can be assigned to various external events. External interrupts are falling edge triggered.

6.1.2 Vectors

The vector number is used to generate the address of a particular interrupt servicing routine; therefore all interrupts using the same vector must use the same interrupt handling routine.

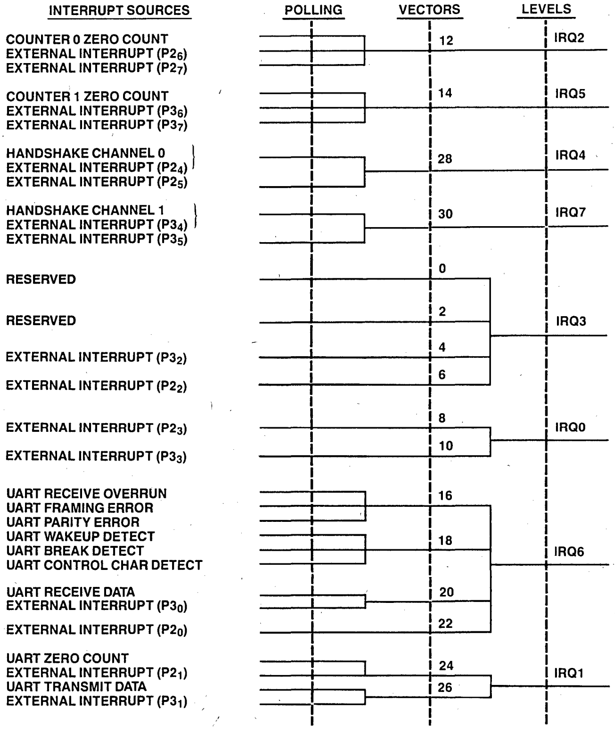

INTERRUPT SOURCES
POLLING
VECTORS
LEVELS
COUNTER 0 ZERO COUNT
EXTERNAL INTERRUPT ($P2_6$)
EXTERNAL INTERRUPT ($P2_7$)
COUNTER 1 ZERO COUNT
EXTERNAL INTERRUPT ($P3_6$)
EXTERNAL INTERRUPT ($P3_7$)
HANDSHAKE CHANNEL 0
EXTERNAL INTERRUPT ($P2_4$)
EXTERNAL INTERRUPT ($P2_5$)
HANDSHAKE CHANNEL 1
EXTERNAL INTERRUPT ($P3_4$)
EXTERNAL INTERRUPT ($P3_5$)
RESERVED
RESERVED
EXTERNAL INTERRUPT ($P3_2$)
EXTERNAL INTERRUPT ($P2_2$)
EXTERNAL INTERRUPT ($P2_3$)
EXTERNAL INTERRUPT ($P3_3$)
UART RECEIVE OVERRUN
UART FRAMING ERROR
UART PARITY ERROR
UART WAKEUP DETECT
UART BREAK DETECT
UART CONTROL CHAR DETECT
UART RECEIVE DATA
EXTERNAL INTERRUPT ($P3_0$)
EXTERNAL INTERRUPT ($P2_0$)
UART ZERO COUNT
EXTERNAL INTERRUPT ($P2_1$)
UART TRANSMIT DATA
EXTERNAL INTERRUPT ($P3_1$)
12
14
28
30
0
2
4
6
8
10
16
18
20
22
24
26
IRQ2
IRQ5
IRQ4
IRQ7
IRQ3
IRQ0
IRQ6
IRQ1

Figure 6-1. Interrupt Structure

When more than one vector shares an interrupt level, the priorities of the vectors on that level are fixed. Figure 6-1 lists the vectors within a level in the order of decreasing priority (i.e., the top vector in each level has the highest priority). For example, for IRQ6, vector 16 always has priority over vectors 18, 20, and 22.

6.1.3 Levels

While the sources and vectors are hardwired within each level, the priorities of the levels can be changed by using the Interrupt Priority register (R255, Bank 0) (Figure 6-2).

Although it does not cover all possible combinations, the Interrupt Priority register does provide the capability of assigning 192 different combinations of priority among the interrupt levels. For example, an IPR with the contents 01101011 would have the following priority order (Figure 6-3):

If more than one interrupt source is active, the source from the highest priority level is serviced first. If both sources are from the same level, the source with the lowest vector number has priority. For example, if the UART Receive Data bit and UART Parity Error bit are both active, the UART Parity Error is serviced first because it is vector 16 and the UART Receive Data bit is vector 20.

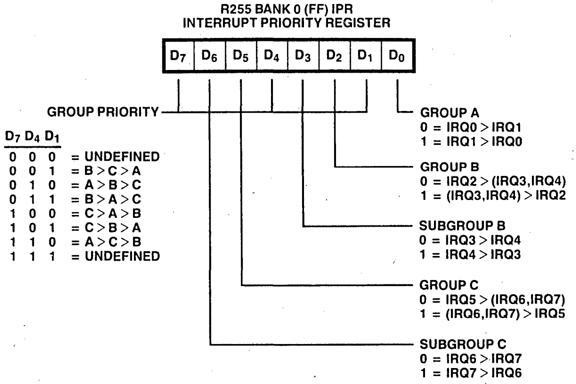


Figure 6-2. Interrupt Priority Register

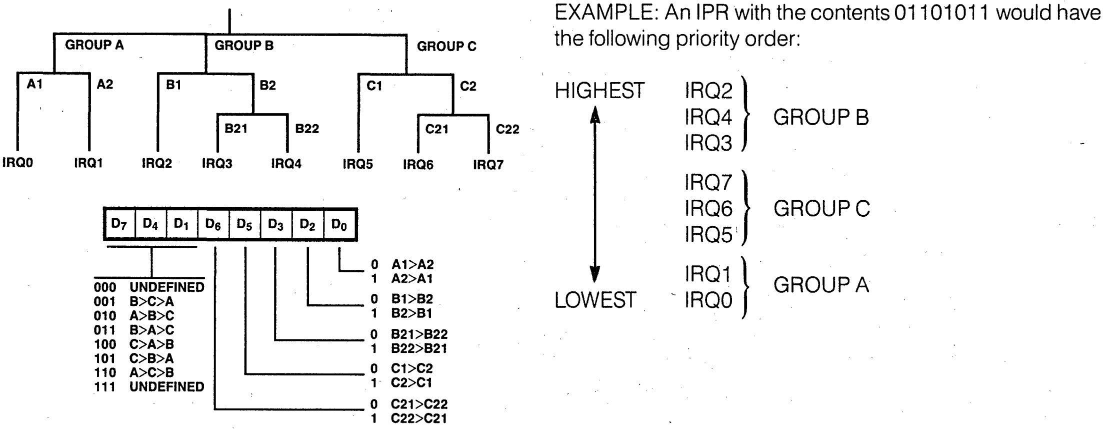


Figure 6-3. Interrupt Priority Tree

When an interrupt occurs, the software is automatically vectored to one of 16 possible service routines. If more than one active source shares that vector, the software must poll the individual sources connected with that vector to find the interrupting source or sources. Each interrupt source has its own Interrupt Enable bit located in the mode and control registers of the I/O section associated with the source. The software has complete control over which sources are allowed to cause interrupts. If only one source associated with a particular vector is enabled, then when an interrupt occurs that uses that vector, no polling is required and the software is automatically vectored to the appropriate service routine.

Table 6-1. Super8 Vector Address Table

Vectors (Decimal Memory Address)	Levels	Interrupt Sources
30,31	IRQ7	$P3_4$ External Interrupt or HS1 / $P3_5$ External Interrupt
28,29	IRQ4	$P2_4$ External Interrupt or HS0 / $P2_5$ External Interrupt
26,27	IRQ1	UART Transmit Data / $P3_1$ External Interrupt
24,25	IRQ1	UART Zero Count / $P2_1$ External Interrupt
22,23	IRQ6	$P2_0$ External Interrupt
20,21	IRQ6	UART Receive Data / $P3_0$ External Interrupt
18,19	IRQ6	UART Break / Control Character / Wake-Up
16,17	IRQ6	UART Overrun / Framing / Parity
14,15	IRQ5	Counter 1 Zero Count / $P3_6$ External Interrupt / $P3_7$ External Interrupt
12,13	IRQ2	Counter 0 Zero Count / $P2_6$ External Interrupt / $P2_7$ External Interrupt
10,11	IRQ0	$P3_3$ External Interrupt
8,9	IRQ0	$P2_3$ External Interrupt
6,7	IRQ3	$P2_2$ External Interrupt
4,5	IRQ3	$P3_2$ External Interrupt
2,3	IRQ3	Reserved
0,1	IRQ3	Reserved

6.1.4 Enables

Interrupts can be enabled or disabled as follows:

- **Interrupt enable/disable.** The entire interrupt structure can be enabled or disabled by setting bit 0 in the System Mode register (R222).

- **Level enable.** Each level can be enabled or disabled by setting the appropriate bit in the Interrupt Mask register (R221).

- **Level priority.** The priority of each level can be controlled by the values in the Interrupt Priority register (R255, Bank 0).

- **Source enable/disable.** Each interrupt source can be enabled or disabled in the source's Mode and Control register.

6.1.5 The Interrupt Routine

Interrupts are sampled at the end of each instruction. Before an interrupt request can be granted a) interrupts must be enabled, b) the level must be enabled and must be the highest priority interrupting level, and c) the interrupt request must be enabled at the interrupting source and must have the highest priority within the level.

If all this occurs, an interrupt request is granted.

The Super8 then enters an interrupt machine cycle that completes the following sequence:

- Resets the Interrupt Enable bit to disable all subsequent interrupts

- Saves the Program Counter and status flags on the stack

- Branches to the address contained within the vector location for the interrupt

- Passes control to the interrupt servicing routine

Interrupts can be re-enabled by the interrupt handling routine (EI instruction), which allows interrupt nesting. First, however, the contents of the Interrupt Mask register should be saved and a new mask loaded which disables the present level being serviced and all lower levels.

When the interrupt handling routine is finished, it should issue an Interrupt Return (IRET) instruction. This instruction restores the Program Counter and status flags from the stack and sets the Global Interrupt Enable bit. If nesting was used, the interrupt handling routine should first execute a Disable Interrupt (DI) instruction and restore the saved mask before executing the IRET instruction. Figure 6-4 illustrates the interrupt cycle process that occurs when an interrupt request occurs.

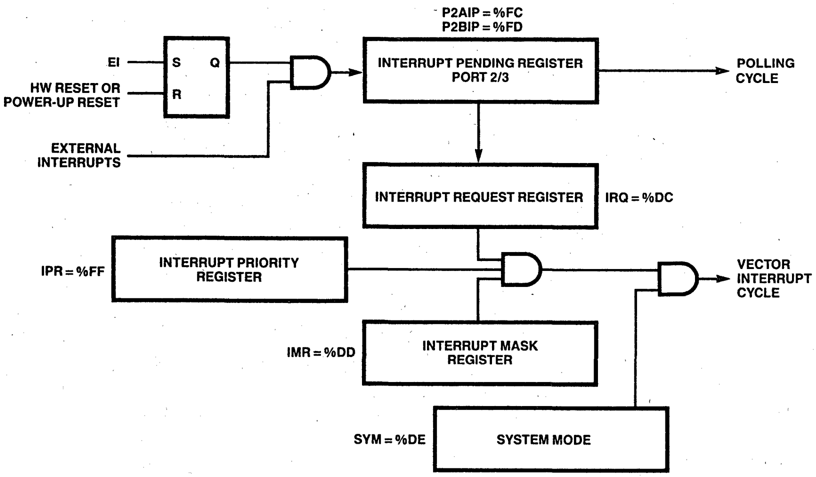


Figure 6-4. Interrupt Cycle Process

6.2 FAST INTERRUPT PROCESSING

The Super8 provides a feature called fast interrupt processing, which completes the interrupt servicing in 6 clock periods instead of the usual 22.

Any one of the eight interrupt levels can be programmed to use this feature by loading the fast interrupt select field of the System Mode register (R222) with the level number and setting the Fast Interrupt Enable bit.

Two hardware registers support fast interrupts. The Instruction Pointer (IP) holds the starting address of the service routine and saves the Program Counter (PC) value when a fast interrupt occurs. A dedicated register, Flag', saves the contents of the Flag register when a fast interrupt occurs.

To use this feature, software must first set the Instruction Pointer to the starting location of the interrupt service routine during initialization and before interrupts are enabled for the first time. Then the level number is loaded into the Fast Interrupt Select field and the Fast Interrupt Enable bit in the System Mode register is turned on.

When an interrupt occurs in the level selected for fast interrupt processing, the following occurs:

- The contents of the Instruction Pointer and the Program Counter are swapped.
- The contents of the Flag register are copied into Flag'.
- The Fast Interrupt Status bit in the Flag register is set.
- The interrupt is serviced.
- When IRET is issued after the interrupt service routine is completed, the Instruction Pointer and the Program Counter are swapped again.
- The contents of Flag' are copied back into the Flag register.
- The Fast Interrupt Status bit in the Flag register is cleared.

After the Interrupt Return (IRET) of a fast interrupt, the Instruction Pointer (IP) will point to the next byte following the IRET. Before using the fast interrupt again, the IP should be re-initialized to point to the beginning of the interrupt routine. While fast interrupt processing is enabled, normal interrupt processing still functions for the unselected levels.

The Super8 supports both polled and interrupt-driven systems or a combination of both. To accommodate a polled structure or a partially polled structure, any or all of the interrupt levels can be masked and the individual bits of the IRQ register polled.

6.3 CLEARING THE INTERRUPT SOURCE

Internally, the interrupt requests are represented as levels. This level-activated system requires that the software that services an interrupt must perform some action that removes the interrupting source before re-enabling that interrupt.

For external interrupt inputs on the Port 2 and 3 pins, edge-triggered "interrupt pending" flip-flops are used to convert an edge-triggered input to a level-activated interrupt. Thus, the service routine must reset the interrupt pending flip-flop to clear the interrupt request by writing to the Port 2/3 Interrupt Pending register.

For receive character available interrupts from the UART receiver, emptying the Receive Data register (UIOR) will automatically clear the interrupt source. For receiver interrupts due to a receive error, detection of a control character, or detection of the wake-up condition, resetting the appropriate status bit in the Receive Control register (URC) will clear the interrupt source. For interrupts from the UART transmitter, filling the Transmit Data register (UIOT) will automatically clear the interrupt source.

For end-of-count interrupts from the counter/timers, resetting the Reset/End of Count Status bit (D_1) in the Counter Control register will clear the interrupt source.

For interrupts from the on-chip DMA channel, loading a non-zero value into the DMA Count register will clear the interrupt source.

6.4 INTERRUPT CONTROL REGISTERS

The interrupt hardware is controlled by fields in the System Mode register (R222), the Interrupt Request register IRQ (R220), the Interrupt Mask register IMR (R221), the Interrupt Priority register IPR (R255, Bank 0), and the Fast Interrupt Status bit (FIS) of the Flags register (R213).

6.4.1 System Mode Register

The System Mode register (R222) controls the mode of operation of the interrupt hardware. The format of the System Mode register is shown in Figure 6-5.

The fields in this register pertaining to the interrupt hardware are:

Global Interrupt Enable (D_0). When this bit is set to 1, interrupts are enabled. When this bit is cleared to 0, all interrupts are disabled regardless of the state of individual interrupt enable or mask bits. This bit is automatically cleared during an interrupt machine cycle and can also be cleared by the DI instruction. It can be set by using an EI or IRET instruction. A hardware reset clears this bit.

Fast Interrupt Enable (D_1). When this bit is a 1, the fast interrupt processing feature is enabled for the selected interrupt level. When this bit is a 0, fast interrupt processing is disabled. When fast interrupt processing is used, the Interrupt Mask Register bit for the selected level must also be set.

Fast Interrupt Select (D_2-D_4). The value of this 3-bit field selects the interrupt level for fast interrupt processing. All other levels still operate in the normal interrupt mode.

(Bit 7 relates to external memory and not to interrupts. For more details on bit 7, see section 12.3.)

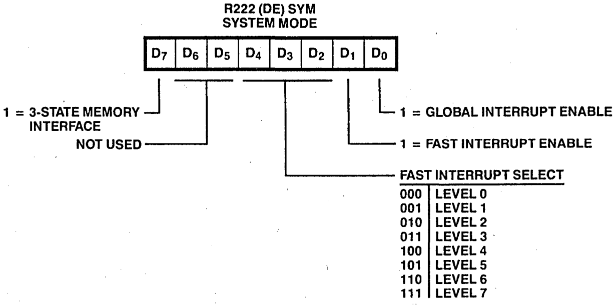


Figure 6-5. System Mode Register

6.4.2 Interrupt Request Register

The Interrupt Request (IRQ) register (R220) indicates which interrupt levels have pending interrupts. It takes a snapshot once for each instruction near the end of execution. Each bit in the register corresponds to one interrupt level. Software can use the IRQ for polling those levels that are not using hardware interrupts and have been masked off by the IMR. Even when polling, the software is responsible for removing the interrupting source when servicing that source. Writing to the IRQ has no effect. The interrupt request must be renewed at the source, such as the UART or a port.

External interrupts are disabled by a reset and must be enabled via execution of an EI instruction before bits in the Port 2/3 Interrupt Pending registers can be set and external hardware interrupts can occur.

The format of the Interrupt Request register is shown in Figure 6-6.

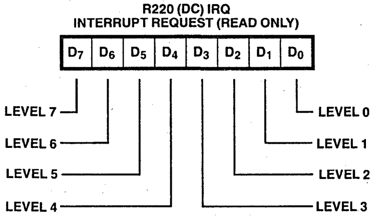


Figure 6-6. Interrupt Request Register

6.4.3 Interrupt Mask Register

The Interrupt Mask (IMR) register (R221) is used to mask individual interrupt levels, thus preventing interrupts at that level. A 1 enables interrupts at that level, a 0 disables them. Interrupts should be globally disabled before writing to this register.

The format of the Interrupt Mask register is shown in Figure 6-7.

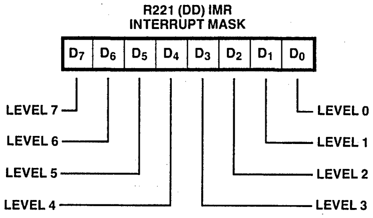


Figure 6-7. Interrupt Mask Register

6.4.4 Interrupt Priority Register

The Interrupt Priority (IPR) register (R255, Bank 0) defines the priority order of the interrupt levels. The coding of this register is defined in Figure 6-2. Interrupts should be globally disabled before writing to this register.

6.4.5 Fast Interrupt Status Bit (FIS of Flags Register)

This is a status bit; when it is set to 1, it indicates that a fast interrupt has occurred. This bit determines what type of action is taken during an IRET. If it is a 1, then an IRET causes a swap between the Program Counter and the Instruction Pointer, and the Flags' register to be written into the Flag register. If it is a 0, then IRET causes a normal interrupt return. A hardware reset clears this bit to 0.

The format of the Flags register is shown in Figure 5-1, Chapter 5.

6.5 INTERRUPTS AND THE DMA CHANNEL

When the DMA channel is enabled to work with a handshake-driven I/O port or the UART, the interrupt request from the specific device is replaced by an interrupt request from the DMA channel when the specified number of transfers has been completed (see Figure 6-8).

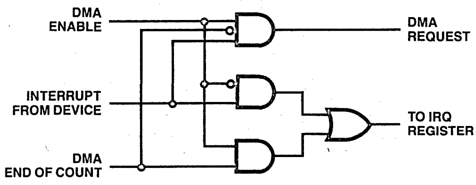


Figure 6-8. Interrupts and the DMA

Chapter 7
Reset and Clock

7.1 RESET

A system reset, activated by a low level on the $\overline{\text{RESET}}$ input, overrides all other operating conditions and puts the Super8 into a known state. The $\overline{\text{RESET}}$ input is internally synchronized with the internal clock of the Super8 to form the internal reset line. For a power-up reset operation when using the on-chip oscillator, the $\overline{\text{RESET}}$ input must be held low for at least 50 milliseconds after the power supply is within tolerance to allow the on-chip clock oscillator to stabilize. If an external clock oscillator is used or power has been applied long enough for the on-chip oscillator to stabilize, then the $\overline{\text{RESET}}$ input must be held low for at least 18 clock periods to cause a system reset.

While $\overline{\text{RESET}}$ is active low, the $\overline{\text{DS}}$ output is forced low while $\overline{\text{AS}}$ pulses low once every four clock cycles and R/$\overline{\text{W}}$ remains high. Z-BUS-compatible peripherals use the $\overline{\text{AS}}$ and $\overline{\text{DS}}$ coincident low state as a peripheral reset function.

Resets also result in the following:

- Interrupts are disabled (the Global Interrupt Enable bit is cleared and the Interrupt Request register is disabled)

- Ports 2, 3, and 4 are placed in input mode

- In parts with on-chip ROM, Ports 0 and 1 are placed in input mode; in ROMless parts, Port 1 is configured as an address/data bus to external memory while Port 0 bits 0-4 are configured as address bits 8-12 and bits 5-7 are in input mode

- The on-chip peripherals are all disabled

- The Program Counter is loaded with 0020_H

Table 7-1 shows the reset values of the control and peripheral registers. Specific reset values are shown by 1s or 0s, while an x indicates bits whose states are not defined and † indicates not used.

Table 7-1. Control and Peripheral Register Reset Values

Register Name **Mnemonic, Decimal, Hex**	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Comments**
General Registers									
Program Control Flags FLAGS, R213, D5	x	x	x	x	x	x	0	0	Bank 0, no fast interrupts
Register Pointer 0 RP0, R214, D6	1	1	0	0	0	0	0	0	Working register C0
Register Pointer 1 RP1, R215, D7	1	1	0	0	1	0	0	0	Working register C8
Stack Pointer SP, R216-7, D8-D9	x	x	x	x	x	x	x	x	
Instruction Pointer IP, R218-9, DA,DB	x	x	x	x	x	x	x	x	
Interrupt Request IRQ, R220, DC	0	0	0	0	0	0	0	0	Interrupts disabled
Interrupt Mask IMR, R221, DD	x	x	x	x	x	x	x	x	
System Mode SYM, R222, DE	0	†	†	x	x	x	0	0	Disable interrupts disable 3-state
External Memory Timing EMT, R254, FE (Bank 0)	0	1	1	1	1	1	0	0	3 wait states for Program and Data, Slow memory
Interrupt Priority IPR, R255, FF (Bank 0)	x	x	x	x	x	x	x	x	
Port Registers									
Port 0 P0, R208, D0	x	x	x	x	x	x	x	x	
Port 1 P1, R209, D1	x	x	x	x	x	x	x	x	

Key		
	1 = Reset value of 1	x = bits whose states are not defined
	0 = Reset value of 0	† = not used

Table 7-1. Control and Peripheral Register Reset Values (Continued)

Register	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Comments
Port Registers (Continued)									
Port 2 P2, R210, D2	1	1	1	1	1	1	1	1	Output register = 1 Value will not be observable until ports are configured as output
Port 3 P3, R211, D3	1	1	1	1	1	1	1	1	Output register = 1 Value will not be observable until ports are configured as output
Port 4 P4, R212, D4	x	x	x	x	x	x	x	x	
Handshake 0 Control H0C, R244, F4	x	x	x	x	x	0	x	0	Disable handshake Ports 1 and 4, disable DMA, (write only)
Handshake 1 Control H1C, R245, F5	x	x	x	x	x	x	x	0	Disable handshake Port 0 (write only)
Port 4 Direction P4D, R246, F6	1	1	1	1	1	1	1	1	Inputs
Port 4 Open-Drain P4OD, R247, F7	0	0	0	0	0	0	0	0	Push-pull
Port 2/3 Mode P2AM, R248-251, F8,F9,FA,FB (Bank 0)	0	0	0	0	0	0	0	0	Inputs (write only) (P2AM, P2BM, P2CM, P2DM)
Port 2/3 Interrupt Pending P2AIP, R252-3, FC,FD	0	0	0	0	0	0	0	0	(Write only) software reset (P2AIP, P2BIP)
Port 0 Mode P0M, R240, F0 (Bank 0)	0 0	0 0	0 0	0 1	0 1	0 1	0 1	0 1	With ROM: input/output ROMless: 1 = Address
Port Mode PM, R241, F1 (Bank 0)	† †	† †	0 1	1 0	0 0	0 0	0 0	1 1	With ROM: Port 0/1 inputs (write only) ROMless: Port 0/1 outputs

Key: 1 = Reset value of 1 x = bits whose states are not defined
0 = Reset value of 0 † = not used

Table 7-1. Control and Peripheral Register Reset Values (Continued)

Register	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Comments
UART and DMA Registers									
UART Transmit Control UTC, R235, EB	0	0	0	0	0	0	1	0	Disable transmitter, transmit buffer empty
UART Receive Control URC, R236, EC	0	0	0	0	0	0	0	0	Disable receiver No character received
UART Interrupt Enable UIE, R237, ED	0	0	0	0	0	0	0	0	Disable interrupts
UART Data UIO, R239, EF	x	x	x	x	x	x	x	x	
UART Baud-Rate Generator UBG, R248-9, F8,F9 (Bank 1)	x	x	x	x	x	x	x	x	
UART Mode A UMA, R250, FA (Bank 1)	x	x	x	x	x	x	x	x	
UART Mode B UMB, R251, FB (Bank 1)	0	0	0	0	0	0	0	0	Disable baud-rate generator
Wake-Up Match WUMCH, R254, FE (Bank 1)	x	x	x	x	x	x	x	x	
Wake-Up Mask WUMSK, R255, FF (Bank 1)	x	x	x	x	x	x	x	x	
DMA Count DC, R240-1, F0,F1 (Bank 1)	x	x	x	x	x	x	x	x	
Counter Registers									
Counter 0 Control C0CT, R224, E0 (Bank 0)	x	x	0	0	0	0	0	0	Disable counter 0, interrupts, software capture

Key: 1 = Reset value of 1 x = bits whose states are not defined
0 = Reset value of 0 † = not used

Table 7-1. Control and Peripheral Register Reset Values (Continued)

Register	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	**Comments**
Counter Registers (Continued)									
Counter 1 Control C1CT, R225, E1 (Bank 0)	x	x	0	0	0	0	0	0	Disable counter 1, interrupts, software capture
Counter 0 Capture C0C, R226-7, E2,E3 (Bank 0)	x	x	x	x	x	x	x	x	
Counter 1 Capture C1C, R228-9, E4,E5 (Bank 0)	x	x	x	x	x	x	x	x	
Counter 0 Mode C0M, R224, E0 (Bank 1)	0	0	0	0	x	x	x	x	Port 2 I/O
Counter 1 Mode C1M, R225, E1 (Bank 1)	0	0	0	0	x	x	x	x	Port 3 I/O
Counter 0 Time Constant C0TC, R226-7, E2,E3 (Bank 1)	x	x	x	x	x	x	x	x	
Counter 1 Time Constant C1TC, R228-9, E4,E5 (Bank 1)	x	x	x	x	x	x	x	x	

Key: 1 = Reset value of 1 x = bits whose states are not defined
0 = Reset value of 0 † = not used

Eight clock cycles after RESET has returned high, the Super8 starts program execution. The initial instruction fetch is from location 0020_H. The first program segment executed is typically a routine to initialize the control registers to the required system configuration. Figures 7-1 and 7-2 show the reset timing.

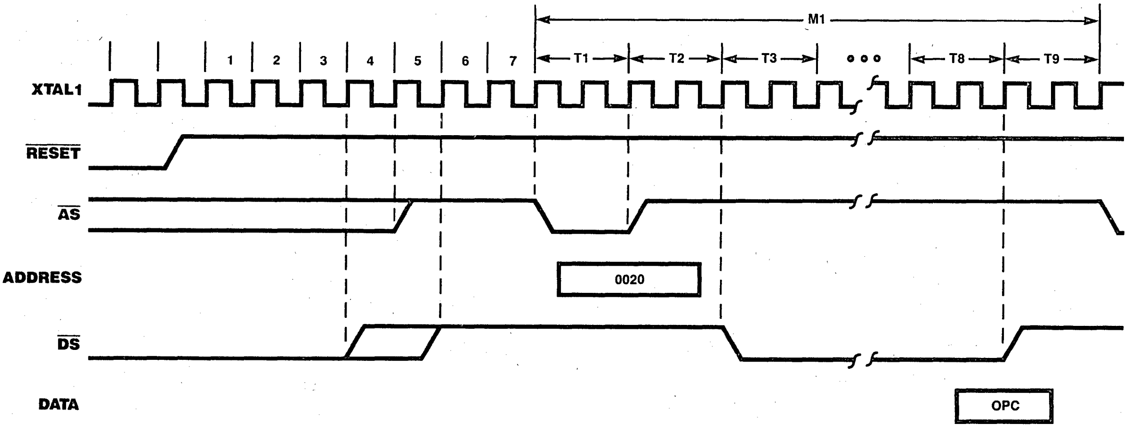


Figure 7-1. Reset Timing for ROMless Devices

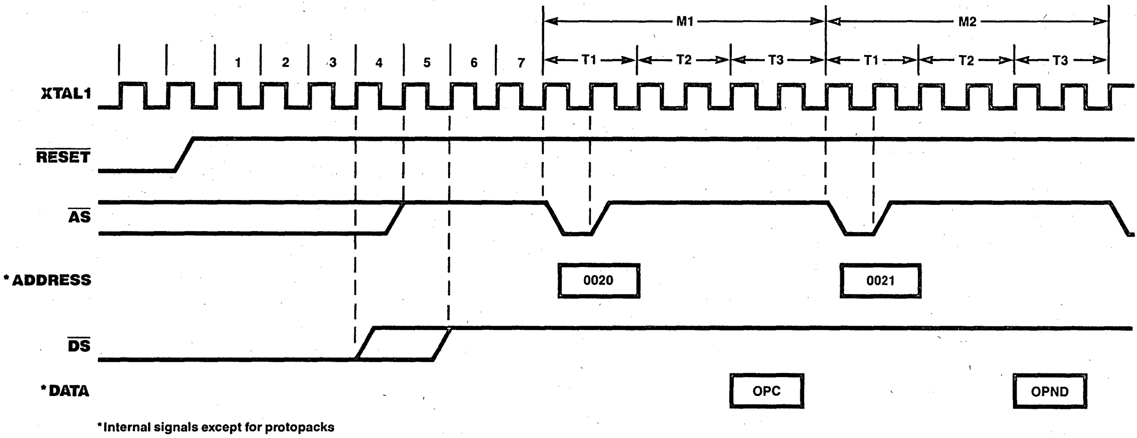


Figure 7-2. Reset Timing for ROM and Protopack Devices

7.2 CLOCK

The Super8 derives its timing from on-board clock circuitry connected to pins XTAL1 and XTAL2. The clock circuitry consists of an oscillator, a divide-by-two shaping circuit, and a clock buffer. Figure 7-3 illustrates the clock circuitry.

The oscillator's inputs are XTAL1 and XTAL2, which can be driven by a crystal, a ceramic resonator, or an external clock source. The divide-by-two circuit can also be driven directly from a TTL level on the XTAL1 pin.

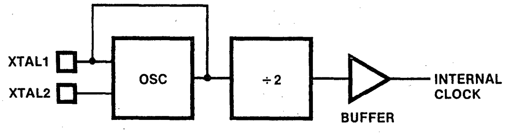


Figure 7-3. Super8 Clock Circuit

Crystals and ceramic resonators would be connected across XTAL1 and XTAL2 and should have the following characteristics to ensure proper oscillator operation:

Cut:	AT (crystal only)
Mode:	Parallel, fundamental
Output Frequency:	1 MHz-12 MHz
Resistance:	100 ohms maximum
Capacitance:	30 pf maximum

When an external frequency source is used, only the XTAL1 input needs to be driven. Any TTL-compatible driver can be used for this function. The XTAL2 input can be left floating.

7.3 TEST MODE

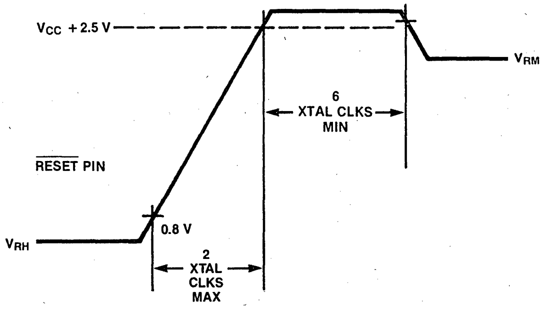
Test mode is a special mode of operation designed to facilitate testing of Super8 devices that contain on-board ROM. Test mode consists of a special 128-byte "shadow" ROM that is mapped into the first 128 locations of program memory and accessible only when test mode is invoked.

Test mode is entered by driving the $\overline{\text{RESET}}$ input to a voltage level of V_{CC} + 2.5V upon terminating a normal reset cycle. The voltage waveform needed to enter test mode is shown in Figure 7-4 and must be adhered to for proper operation.

After entering test mode, instructions are fetched from the internal test ROM and are used to configure Ports 0 and 1 as an external memory interface and then jump to external memory location 4030_H. Once in external memory, diagnostic routines used to verify the functionality of the Super8 are invoked by the test system via the address/data bus. During this process, Port 1 is used only in its address/data mode; therefore, additional routines are provided in the test ROM which the test system uses to verify the I/O and handshake modes of Port 1.

To support testing the interrupt structure, the first 32 locations of test ROM contain interrupt vectors. Interrupt vectors point to locations 4000_H for IRQ0, 4003_H for IRQ1, 4006_H for IRQ2, and so on in external memory. This allows the external program to have a 2- or 3-byte jump instruction for each interrupt service routine.

The Super8 stays in test mode until a normal reset occurs.



Note the maximum ramp for application of +7.5V dc to $\overline{\text{RESET}}$ pin. After a minimum of 6 XTAL CLK cycles, the $\overline{\text{RESET}}$ voltage can be relaxed to V_{RM}.

Figure 7-4. Voltage Waveform for Test Mode

Chapter 8
I/O Ports

8.1 INTRODUCTION

The Super8 has 40 lines dedicated to input and output. These are grouped into five ports of eight lines each. All the lines can be configured as inputs or outputs; some can be configured as address/data lines. All ports have TTL-compatible input and output characteristics and can drive two standard TTL loads.

8.2 GENERAL STRUCTURE

In general, each bit of the five ports has an associated input register, output register, and buffer and control logic. When the CPU writes to a port, it causes data to be stored in the output register. Those bits of that port configured as outputs enable the output buffer, and the output register contents are present on the external pin. If those bits configured as outputs are read by the CPU, the data present on the external pin is returned. Under normal output loading, this is the equivalent of reading the output register. However, if a bit of the port is configured as an open-drain output, the data returned may not be the value contained in the output register; rather it is the value forced on the input pins by the external system.

When a bit of any port is defined as an input, reading that bit causes data present on the external pin to be returned. Ports that are under handshake control are an exception. Reading a handshake-driven input bit returns the data last latched into the input register by the input strobe.

Bits configured as inputs can be written to by the CPU, but in this case, the data is stored in the output register and cannot be read back because the output buffer is disabled. However, if the input bits are reconfigured as output bits, the data stored in the output register is then reflected on the output pins. This mechanism allows the user to initialize outputs prior to driving their loads.

8.3 PORT 0

Port 0 (R208) can be configured as I/O or as an address output port for addressing external memory on a bit basis. Those bits selected as I/O can be configured as all inputs or all outputs. When configured as outputs, the option exists to select open-drain outputs. The open-drain option does not apply to those bits configured as address lines.

Accesses to Port 0 are made by reading and writing to register R208 ($D0_H$ in set one). When a Port 0 bit is configured as an address output, it cannot be accessed as a register (writes have no effect, reads return the state of the external pin). When used as an I/O port, Port 0 may be placed under handshake control by using the facilities of Handshake Channel 1 (see section 8.8).

The following control registers are associated with configuring Port 0:

- **Port Mode register (R241, Bank 0).** Controls direction of I/O lines and selection of open-drain or push-pull outputs.
- **Port 0 Mode register (R240, Bank 0).** Configures each bit as I/O or address bit.
- **Handshake 1 Control register (R245, Bank 0).** Controls enabling and configuration of handshake signals.

8.4 PORT 1

Port 1 (R209) can be configured as an address/data port for interfacing external memory or as a byte I/O port. The configuration is set using the Port Mode register (R241, Bank 0). (For a description of Port 1 as part of the external memory interface, see section 12.3.) When configured as a byte output port, there is an option to select open-drain outputs on the entire port. In the ROMless parts, Port 1 is always an address/data bus and cannot be programmably configured.

When configured as an input or output port, accesses are made to Port 1 via reads or writes to register R209 ($D1_H$ in set one). When Port 1 is configured as a multiplexed address/data port, it cannot be accessed as a register; writes have no effect and reads return an FF_H. When used as an I/O port, Port 1 can be placed under handshake control by using the facilities of Handshake Channel 0 (see section 8.8).

The following control registers are associated with configuring Port 1:

- **Port Mode register (R241, Bank 0).** Controls Port 1 configuration (input port, output port, or address/data bus) and selection of open-drain or push-pull outputs.

- **Handshake 0 Control register (R244, Bank 0).** Controls the enabling and configuration of the handshake signals.

8.5 PORTS 2 AND 3

Ports 2 and 3 (R210 and R211) are used to provide the external control inputs and outputs for the UART, the handshake channels, and the counter/timers. The relationship between port pins and their control function is shown in Table 8-1. When Port 2 and 3 bits are not used for control inputs and outputs, they are available for use as general-purpose I/O lines and/or external interrupt inputs. Each bit is individually configured as to its function.

When Ports 2 and 3 are used as general-purpose I/O lines, the direction of each bit can be configured individually. Each bit selected as an output can also be configured individually as an open-drain or push-pull output. All inputs of Ports 2 and 3 are Schmidt-triggered.

The following control registers are associated with configuring Ports 2 and 3:

- **Port 2/3 A Mode register (R248, Bank 0).** Controls the configuration of bits 0 and 1 (input, input with interrupt enabled, push-pull input, open-drain output).

- **Port 2/3 B Mode register (R249, Bank 0).** Controls configuration of bits 2 and 3.

- **Port 2/3 C Mode register (R250, Bank 0).** Controls configuration of bits 4 and 5.

- **Port 2/3 D Mode register (R251, Bank 0).** Controls configuration of bits 6 and 7.

The various control functions are enabled in the control register for the associated device (Handshake Control register, Counter Mode register, etc.). When using Port 2 and 3 pins as control signals, the Port 2/3 Mode registers must still be programmed to specify which bits are inputs and which bits are outputs.

Each bit of Ports 2 and 3 can be used as an external interrupt input. Each bit used as an external interrupt input must be configured as an input, but may still be used as an external control input or as a general-purpose input line. Each external interrupt bit has an edge-triggered "interrupt-pending" flip-flop that captures the external interrupt requests. Software can read and reset the edge-triggered flip-flops without affecting the normal I/O operation of the bit. Each external interrupt has its own interrupt enable control that determines if that bit is allowed to cause an interrupt. The edge-triggered flip-flops still capture edges when the interrupt enable control is disabled. Port 2 is accessed as general register R210, Port 3 as general register R211.

Table 8-1. Ports 2 and 3 Control Functions

-- Port 2 --		-- Port 3 --	
Bit	**Function**	**Bit**	**Function**
0	UART Receive Clock	0	UART Receive Data
1	UART Transmit Clock	1	UART Transmit Data
2	Reserved	2	Reserved
3	Reserved	3	Reserved
4	Handshake 0 Input	4	Handshake 1 Input/$\overline{\text{WAIT}}$
5	Handshake 0 Output	5	Handshake 1 Output/$\overline{\text{DM}}$
6	Counter 0 Input	6	Counter 1 Input
7	Counter 0 I/O	7	Counter 1 I/O

Two registers are directly associated with the interrupt flip-flops:

- **Port 2/3 A Interrupt Pending register (R252, Bank 0).** Controls interrupt flip-flops for bits 0, 1, 2 and 3 of Ports 2 and 3.

- **Port 2/3 B Interrupt Pending register (R253, Bank 0).** Controls interrupt flip-flops for bits 4, 5, 6, and 7 of Ports 2 and 3.

These registers can be used to poll the external interrupts and to reset the interrupt pending bits (the flip-flops). Reading these registers returns the state of the interrupt pending flip-flop. When writing to these registers, writing a 1 to a bit position clears that flip-flop and writing a 0 to a bit position has no effect.

The Interrupt Mask register (R221) and Port 2/3 Mode registers determine which interrupts are enabled.

8.6 PORT 4

Port 4 (R212) is always an I/O port whose direction can be configured on a bit-by-bit basis. Each bit configured as an output can be configured individually as an open-drain or push-pull output.

Port 4 I/O lines are accessed via reads and writes to register R212 ($D4_H$ in set one).

Port 4 can be placed under handshake control by using the facilities of Handshake Channel 0 (see section 8.8).

The following control registers are associated with configuring Port 4:

- **Port 4 Direction register (R246, Bank 0).** Controls direction of each bit of Port 4.

- **Port 4 Open-Drain register (R247, Bank 0).** Selects open-drain or push-pull for each Port 4 output.

- **Handshake 0 Control register (R244, Bank 0).** Controls the enabling and configuration of the handshake signals.

8.7 PORT MODE AND CONTROL REGISTERS

The ports are configured and controlled by the following set of registers:

- Port Mode
- Port 0 Mode
- Port 2/3 A Mode
- Port 2/3 B Mode
- Port 2/3 C Mode
- Port 2/3 D Mode
- Port 2/3 A Interrupt Pending
- Port 2/3 B Interrupt Pending
- Port 4 Direction
- Port 4 Open-Drain

8.7.1 Port Mode Register

The Port Mode register provides some additional mode control for Ports 0 and 1. The fields in this register are (Figure 8-1):

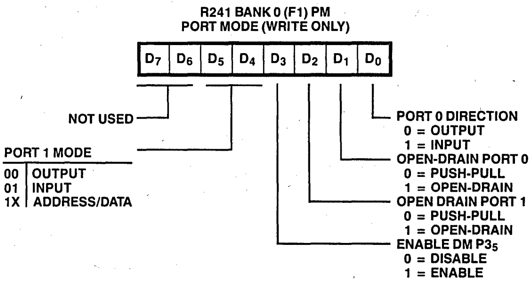


Figure 8-1. Port Mode Register

Port 0 Direction (D_0). If this bit is a 1, all bits of Port 0 configured as I/O will be inputs. If this bit is a 0, then the I/O lines will be outputs. A hardware reset forces this bit to a 1.

Open-Drain Port 0 (D_1). If this bit is a 1, all bits of Port 0 configured as outputs will be open-drain outputs; if 0, they will be push-pull outputs. This bit has no effect on those bits not configured as outputs. A hardware reset forces this bit to a 0.

Open-Drain Port 1 (D_2). If Port 1 is configured as an output port and this bit is a 1, then all of the port will be open-drain outputs. If this bit is a 0, they will be push-pull outputs. This bit has no effect if Port 1 is not configured as an output port or A/D_{0-7}. A hardware reset forces this bit to a 0.

Enable $\overline{\text{DM}}$ (D_3). If this bit is a 1, Port 3_5 is configured as Data Memory output line ($\overline{\text{DM}}$). A hardware reset forces this bit to a 0.

Port 1 Mode (D_4-D_5). This field selects the configuration of Port 1 as an output port, input port, or address/data port as part of the external memory interface. The coding for this field is as follows:

Field	Function
00	Output port
01	Input port
1X	Address/data

A hardware reset forces this field to the 01 (input port) state. The ROMless part has this field forced to 1X.

8.7.2 Port 0 Mode Register

The Port 0 Mode reqister programs each bit of Port 0 as an address output (part of an external memory interface) or as an I/O bit (Figure 8-2). When a bit of this reqister is a 1, the correspondinq bit of Port 0 is defined as an address output. When a 0, the corresponding bit of Port 0 is defined as an I/O bit. For ROMless parts, a hardware reset forces this reqister to all 1s for pins $P0_0$-$P0_4$ and 0s for pins $P0_5$-$P0_7$; for parts with on-chip ROM, a hardware reset forces all pins to 0.

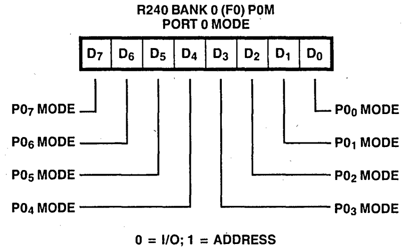


Figure 8-2. Port 0 Mode Register

8.7.3 Port 2/3 Mode Registers

The Port 2/3 A Mode, Port 2/3 B Mode, Port 2/3 C Mode, and Port 2/3 D Mode registers control the modes of Ports 2 and 3 (Figures 8-3, 8-4, 8-5, and 8-6). A separate 2-bit field for each of the bits of Ports 2 and 3 configures the bit as input or output. The field also controls whether the bit is enabled as an external interrupt source and selects the output as open-drain or push-pull. The field is coded as follows:

Field	Function
00	Input
01	Input and interrupt enabled
10	Output, push-pull drivers
11	Output, open-drain

A hardware reset forces all bits of the four registers to the 0 state.

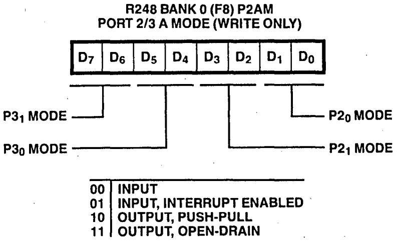


Figure 8-3. Port 2/3 A Mode Register

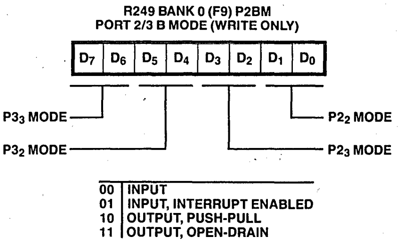


Figure 8-4. Port 2/3 B Mode Register

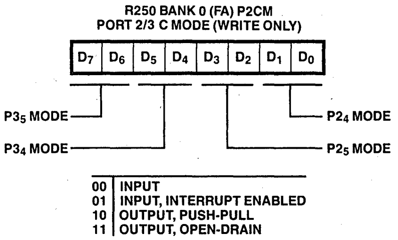


Figure 8-5. Port 2/3 C Mode Register

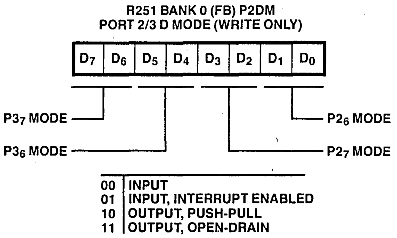


Figure 8-6. Port 2/3 D Mode Register

8.7.4 Port 2/3 Interrupt Pending Registers

The Port 2/3 A Interrupt Pending and Port 2/3 B Interrupt Pending registers represent the software interface to the edge-triggered flip-flops associated with external interrupt inputs. Each bit of these registers corresponds to an interrupt generated by an external source. When one of these registers is read, the value of each bit represents the state of the corresponding interrupt. When one of these registers is written to, a 1 in a bit position causes the corresponding edge-triggered flip-flop to be reset to 0; a 0 causes no action.

The software interfaces with these registers to poll the interrupts and also to reset pending interrupts as they are processed. The relationship between these registers and the corresponding externally generated interrupts is shown in Figures 8-7 and 8-8. A hardware reset forces all interrupt edge-triggered flip-flops to the 0 state.

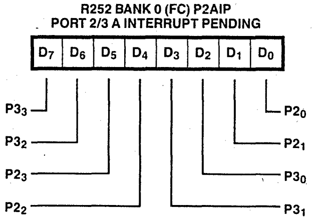


Figure 8-7. Port 2/3 A Interrupt Pending Register

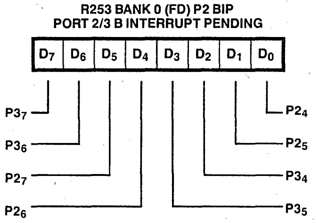


Figure 8-8. Port 2/3 B Interrupt Pending Register

8.7.5 Port 4 Direction Register

The Port 4 Direction register defines the I/O direction of Port 4 on a bit basis (Figure 8-9). If a bit in this register is a 1, the corresponding bit of Port 4 is configured as an input line. If the bit is a 0, the corresponding bit of Port 4 is configured as an output line. A hardware reset forces this register to the all 1s state.

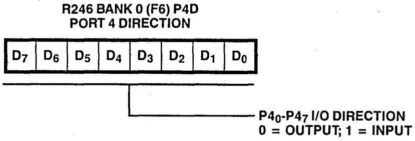


Figure 8-9. Port 4 Direction Register

8.7.6 Port 4 Open-Drain Register

The Port 4 Open-Drain register defines the output driver type for Port 4 (Figure 8-10). If a bit of Port 4 has been configured as an output and the corresponding bit in the Port 4 Open-Drain register is a 1, then the Port 4 bit will have an open-drain output driver; if it is a 0, then the Port 4 bit will have a push-pull output driver. If the bit of Port 4 has been configured as an input, then the corresponding bit in the Port 4 Open-Drain register has no effect. A hardware reset forces this register to the all 0s state.

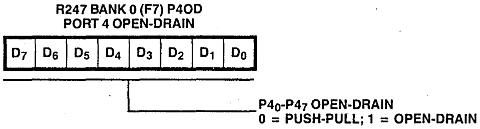
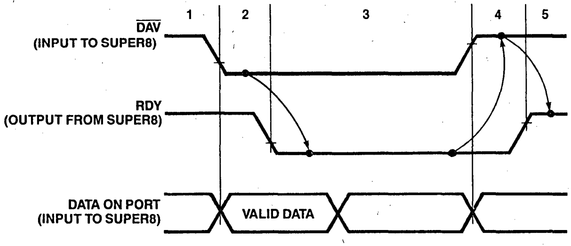


Figure 8-10. Port 4 Open-Drain Register

8.8 HANDSHAKING CHANNELS

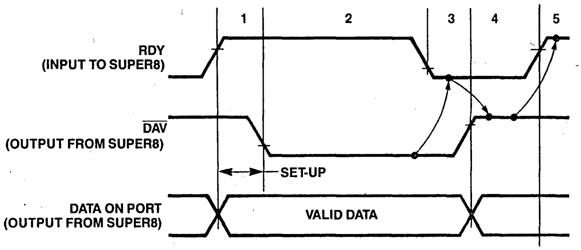
The Super8 has two handshaking channels. Channel "0" is associated with Ports 1 or 4; Channel "1" is associated with Port 0. They are identical in function except Channel 0 also has DMA capability.

There are two basic modes of operation. The first is the "fully interlocked" or two-wire mode. In this mode, there is an incoming control wire and an outgoing control wire. Each transition on a control wire must be answered by a transition on the other control wire before the first can make another transition. Thus both the sender and receiver control the data transmission rate. Figures 8-11 and 8-12 illustrate the operation of the "fully interlocked handshake."



State 1. Ready output is high indicating that the Super8 is ready to accept data.
State 2. The I/O device puts data on the port and then activates the $\overline{DAV}$ input. This causes the data to be latched into the port input register and generates an interrupt or DMA request.
State 3. The Super8 forces the Ready (RDY) output low, signaling to the I/O device that the data has been latched.
State 4. The I/O device returns the $\overline{DAV}$ line high in response to RDY going low.
State 5. The Super8 DMA or interrupt software must respond to the service request and read the contents of the port in order for the handshake sequence to be completed. The RDY line goes high if, and only if, the port has been read and $\overline{DAV}$ is high. This returns the interface to its initial state.

Figure 8-11. Super8 Input Handshake—Fully Interlocked Mode



State 1. RDY input is high indicating that the I/O device is ready to accept data.
State 2. The Super8 writes to the port register to initiate a data transfer. Writing the port outputs new data and forces $\overline{DAV}$ low if, and only if, RDY is high and set-up time is done.
State 3. The I/O device forces RDY low after latching the data. RDY low causes an interrupt or DMA request to be generated. The Super8 can write new data in response to RDY going low.
State 4. The $\overline{DAV}$ output from the Super8 is driven high in response to RDY going low.
State 5. After $\overline{DAV}$ goes high, the I/O device is free to raise RDY high thus returning the interface to its initial state.

Figure 8-12. Super8 Output Handshake—Fully Interlocked Mode

The second mode is the "strobed" or single-wire mode. In this mode there is a single control wire and it is generated by the sender. Figures 8-13 and 8-14 illustrate the operation of "strobed" handshaking.

Each channel has a 4-bit counter, called the Deskew Counter, that is used to count processor clocks. In the "strobed" mode, this counter is used to generate the set-up time and strobe width for the output handshake. In the "fully inter-

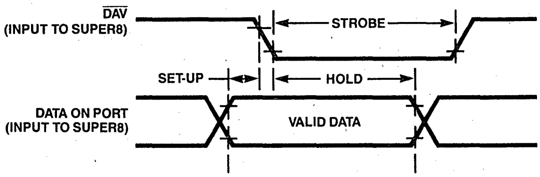
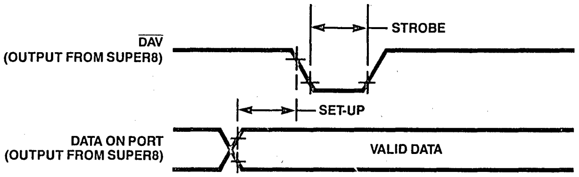


Figure 8-13. Super8 Input Handshake—Strobed Mode



THE SET-UP AND STROBE MINIMUM TIMES ARE DETERMINED BY THE VALUE IN THE DESKEW COUNTER.

Figure 8-14. Super8 Output Handshake—Strobed Mode

locked" mode, the counter generates the set-up time. This set-up time is the delay between outputting valid data at the port and activating the Data Available handshake signal. The Deskew Counter can be loaded with a value from 1 to 16 that represents the minimum number of CPU clock cycles in the data set-up and strobe times.

The direction of data transfer during handshake is determined by the selected direction of bit 0 of the parallel port associated with the handshake channel. This also controls the DMA direction when used.

8.8.1 Pin Descriptions

The handshake channels each use two pins of Ports 2 and 3 (bits 4 and 5) for interfacing with the external world:

Handshake Channel 0 Input	$P2_4$
Handshake Channel 0 Output	$P2_5$
Handshake Channel 1 Input	$P3_4$
Handshake Channel 1 Output	$P3_5$

The individual Port 2 and 3 pins should be configured for the appropriate I/O direction as needed by the handshake function. Note that the open-drain options of Ports 2 and 3 can be applied to the handshake outputs. Note also that Port 2 and 3 pins used by the handshake channels as inputs can still be used as external interrupt pins to drive the handshake service routines.

Handshake Input. This input provides the $\overline{DAV}$ signal for input handshaking or the RDY signal for output handshaking.

Handshake Output. This output provides the RDY signal for input handshaking or the $\overline{DAV}$ signal for output handshaking.

8.8.2 Handshake Control Registers

Each handshake channel is controlled by an 8-bit control register (Figures 8-15 and 8-16). Handshake 0 Control register (R244) and Handshake 1 Control register (R245) include the controls for enabling handshakes, selecting the associated port (Channel 0 only), selecting the handshake type, enabling DMA capability (Channel 0 only), and initializing the Deskew Counter. The fields in these registers are:

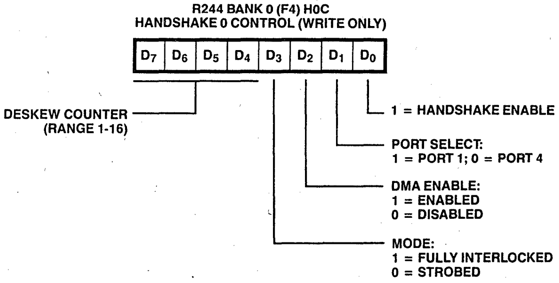


Figure 8-15. Handshake 0 Control Register

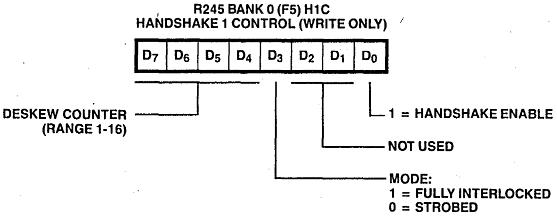


Figure 8-16. Handshake 1 Control Register

Handshake Enable (D_0). When this bit is set to 1, the handshake function is enabled.

Port Select (Channel 0 only)(D_1). This bit selects which port is controlled by Handshake Channel 0. When it is set to 1, Port 1 is selected and when it is cleared to 0, Port 4 is selected.

DMA Enable (Channel 0 only)(D_2). When this bit is set to 1, the DMA function is enabled for Handshake Channel 0. When it is cleared to 0, the DMA function is not used by the handshake channel and may be used by the UART.

Mode (D_3). When this bit is set to 1, the "fully interlocked" mode is enabled. When it is cleared to 0, the "strobed" mode is enabled.

Deskew Counter (D_4-D_7). This 4-bit field is used to select a count value from 1 to 16 (0000-1111). This value is the number of processor clocks used to generate the set-up and strobe when using the "strobed" mode, or the set-up when using the "fully-interlocked" mode.

Chapter 9
Counter/Timers

9.1 INTRODUCTION

The Super8 has two identical 16-bit counter/timers that can be programmed independently. They can be cascaded to produce a counter 32 bits in length and can operate from internal inputs (as timers) or external inputs (counters). When used as timers, the internal input is the internal CPU clock divided by two, which is the XTAL divided by four. Figure 9-1 shows the counter/timer block diagram.

The counter/timers can count up or down. The direction can be controlled on the fly by either software or an external event.

The counter/timers have the option of single cycle or continuous counting capability. In the single cycle mode, the counters count to zero (up or down) from the preset time-constant value and then stop. In the continuous mode, counting is continuous and each time the counter reaches zero, it is reloaded with the preset time-constant value from the Time Constant register (or the Capture register in bi-value mode).

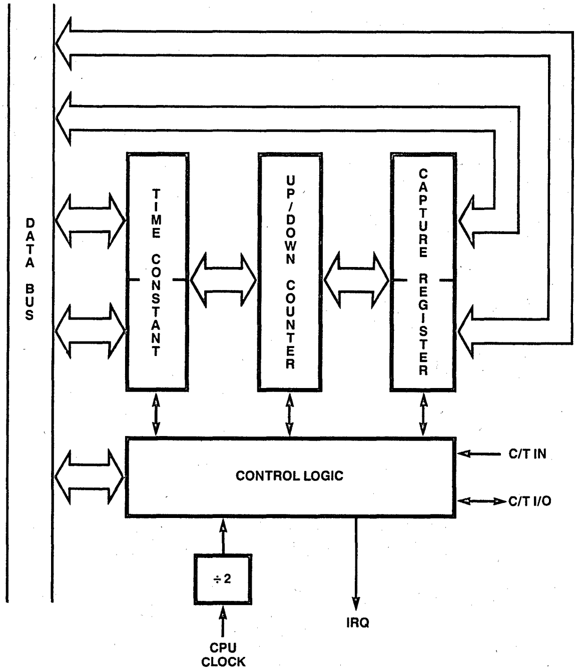


Figure 9-1. Counter/Timer Block Diagram

9.1.1 Bi-Value Mode

Another option allows either a single or dual (bi-value) preset time constant value. In bi-value mode, both the Time Constant register and Capture register are used to supply load values to the counter/ timer. The two registers alternate in loading the counter/timer each time the counter/timer makes a transition between a count of 0 and a count of FFFF$_H$ when counting down, or between a count of FFFF$_H$ and 0 when counting up (assuming continuous mode operation), or when a trigger causes the counter/timer to be reloaded. This can be used to produce an output pulse train with a variable duty cycle. The bi-value feature is not available when the capture feature is enabled and vice versa. Upon enabling a counter/timer in bi-value mode from a previously disabled condition, the initial load of the counter/timer is from the Time Constant register.

9.1.2 Capture

Another feature, called "capture on external event," takes a snapshot of the counter when a specific event occurs. The external event can be simulated by software. When "captured," the current value in the counter is loaded into a special register that can subsequently be read via software. The capture feature is needed to look at counters on the fly, especially cascaded counters.

The external event can be either the rising edge of the counter/timer I/O line (P2$_7$ for C/T0, P3$_7$ for C/T1) or both edges. On the rising edge, the current count value is loaded into the Capture register. If capture on both edges is enabled, the current count value is loaded into the Time Constant register on the falling edge, overwriting the initial load value for that counter.

The capture feature is not available when the bi-value counting feature is being used and vice versa.

If interrupts are enabled, the interrupt request is generated on the transition from a count of 0 to a count of FFFF$_H$ or from a count of FFFF$_H$ to a count of 0, and/or on an external event. If configured for an external output, the output pin toggles at this same count change.

9.1.3 External Gate and Trigger

The counter/timers have an external gate capability. When this feature is selected, an external input line (GATE) is monitored. The counting or timing operation is performed only when this line is low. The gate facility is illustrated in Figure 9-2.

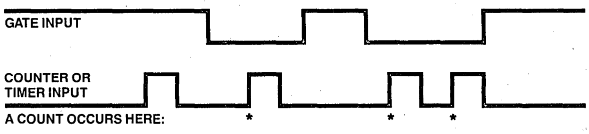


Figure 9-2. Gate Facility

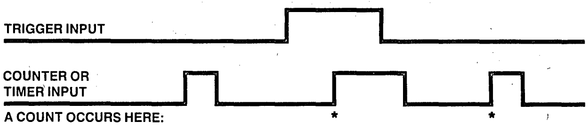


Figure 9-3. Trigger Operation

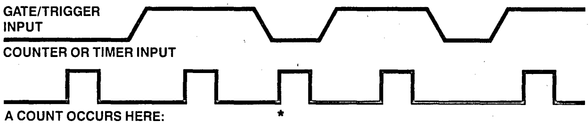


Figure 9-4. Gate/Trigger Function

An external input can be used as a trigger input to a counter/timer. When this feature is selected, an external line is monitored. A software trigger is also present in a control register. The trigger input to the Counter/Timer is an OR of the software and hardware triggers. Prior to a low-to-high transition on the trigger, the Counter is disabled. After the low-to-high transition on the trigger, counting is enabled. Retriggerable or non-retriggerable mode can be selected.

Clearing the Counter Enable bit in the Control register also resets the triggered condition; a new trigger must be received after the Counter Enable bit is set again before counting will resume. The trigger operation is illustrated in Figure 9-3.

One input line (GATE/TRIGGER) can be used for both the gating and the triggering functions. An initial low-to-high transition on this line acts as a trigger and subsequent low signals on this line function as gate signals (Figure 9-4).

9.2 COUNTER/TIMER CONTROL AND MODE REGISTERS

Each counter/timer has an 8-bit Mode register, an 8-bit Control register, a 16-bit Time Constant register, and a 16-bit Capture register.

The Mode and Control registers determine the counter/timer operations. The Mode register selects the configuration of the counter/timers and is generally loaded only at initialization time, while the Control register handles those features that are likely to be dynamically changed.

The Time Constant register contains the initialization value for the counter/timer and also holds the counter value saved on the falling edge of $P2_7/P3_7$ when capture on both edges is enabled.

The Capture register holds the counter value saved when using the "capture on external event" function. When capture on both edges is enabled, it holds the value saved on the rising edge of $P2_7/P3_7$. It also holds a second initialization value when using the bi-value counting feature.

9.2.1 Counter/Timer Control Registers

The fields in these registers, as shown in Figures 9-5 and 9-6, are:

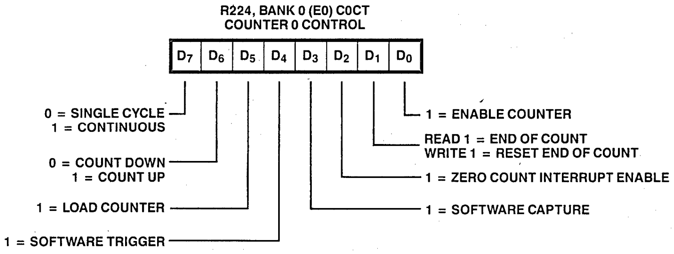


Figure 9-5. Counter 0 Control Register

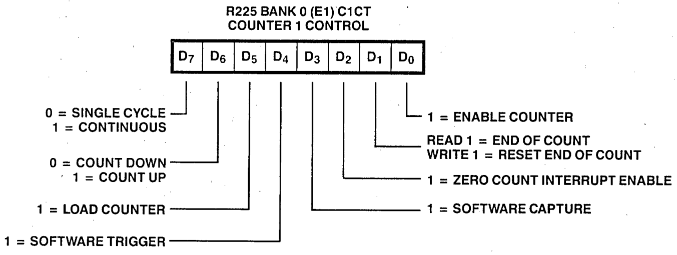


Figure 9-6. Counter 1 Control Register

Enable Counter (D_0). When this bit is set to 1, the counter/timer is enabled; operation begins on the rising edge of the first processor clock period following the setting of this bit from a previously cleared value. Writing a 1 in this field when the previous value was 1 has no effect on the operation of the counter/timer. When this bit is cleared to 0, the counter/timer performs no operation during the next (and subsequent) processor clock periods. A hardware reset forces this bit to 0.

Reset/End of Count Status (D_1). This bit is set to 1 each time the counter reaches 0. Writing a 1 to this bit resets it, while writing a 0 has no effect.

Zero Count Interrupt Enable (D_2). When this bit is set to 1, the counter/timer generates an interrupt request when it counts to 0. A hardware reset forces this bit to 0.

Software Capture (D_3). When this bit is set to 1, the current counter value is loaded into the capture register. This bit is automatically cleared following the capture.

Software Trigger (D_4). This bit is effectively "ORed" with the external rising-edge trigger input and can be used by the software to force a trigger signal. This bit produces a trigger signal regardless of the setting of the Input Pin Assignment field of the Mode register. This bit is automatically cleared following the trigger.

Load Counter (D_5). The contents of the Time Constant register are transferred to the Counter prescaler one clock period after this bit is set. This operation alone does not start the Counter. This bit is automatically cleared following the load.

Count Up/Down (D_6). This bit determines the count direction if internal up/down control is specified in the Mode register. A 1 indicates up, a 0 down.

Continuous/Single Cycle (D_7). When this bit is set to 1 and the count reaches 0, the countdown sequence is automatically restarted by loading the time-constant value into the counter. When this bit is cleared to 0, no reloading occurs.

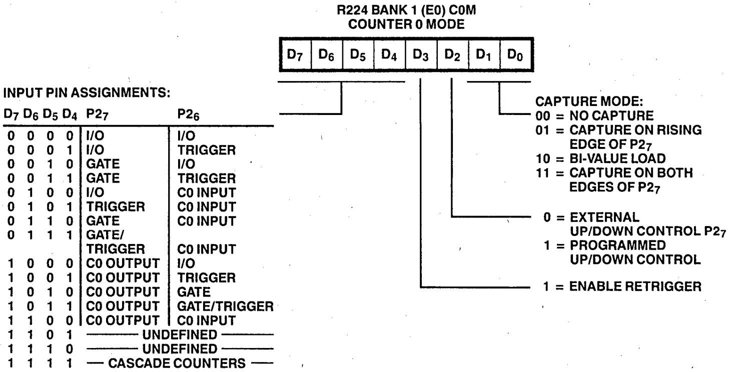
9.2.2 Counter/Timer Mode Registers

The fields in these registers, as shown in Figure 9-7 and 9-8, are:

Capture Mode (D_1, D_0). This 2-bit field selects the capture or bi-value count mode. A value of 01 enables capture on the rising edge of the I/O pin, a value of 11 enables capture on both edges of the I/O pin, a value of 10 enables the bi-value count mode and disables capture, and a value of 00 disables both capture and bi-value load.

Programmed/External Up/Down Control (D_2). A 1 enables programmed up/down control and a 0 enables external up/down control. If external up/down is enabled, a 0 on $P2_7/P3_7$ indicates down and a 1 indicates up.

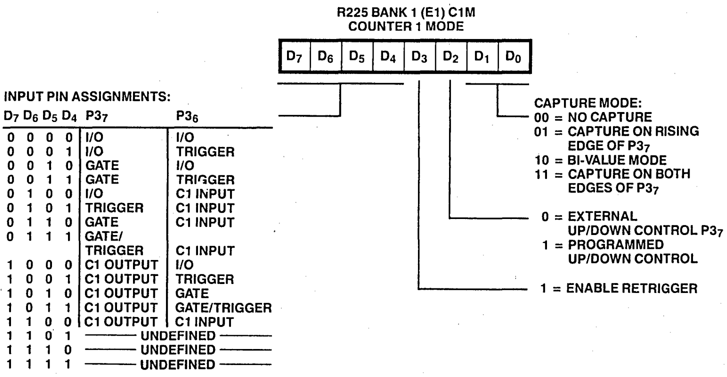
Enable Retrigger (D_3). When this bit is set to 1, the time-constant value is automatically loaded into the Counter/Timer register when a trigger



INPUT PIN ASSIGNMENTS:

D_7	D_6	D_5	D_4	$P2_7$	$P2_6$
0	0	0	0	I/O	I/O
0	0	0	1	I/O	TRIGGER
0	0	1	0	GATE	I/O
0	0	1	1	GATE	TRIGGER
0	1	0	0	I/O	C0 INPUT
0	1	0	1	TRIGGER	C0 INPUT
0	1	1	0	GATE	C0 INPUT
0	1	1	1	GATE/ TRIGGER	C0 INPUT
1	0	0	0	C0 OUTPUT	I/O
1	0	0	1	C0 OUTPUT	TRIGGER
1	0	1	0	C0 OUTPUT	GATE
1	0	1	1	C0 OUTPUT	GATE/TRIGGER
1	1	0	0	C0 OUTPUT	C0 INPUT
1	1	0	1	UNDEFINED	
1	1	1	0	UNDEFINED	
1	1	1	1	CASCADE COUNTERS	

Figure 9-7. Counter 0 Mode Register


R225 BANK 1 (E1) C1M
COUNTER 1 MODE
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
CAPTURE MODE:
00 = NO CAPTURE
01 = CAPTURE ON RISING EDGE OF $P3_7$
10 = BI-VALUE MODE
11 = CAPTURE ON BOTH EDGES OF $P3_7$
0 = EXTERNAL UP/DOWN CONTROL $P3_7$
1 = PROGRAMMED UP/DOWN CONTROL
1 = ENABLE RETRIGGER

INPUT PIN ASSIGNMENTS:

D_7 D_6 D_5 D_4	$P3_7$	$P3_6$
0 0 0 0	I/O	I/O
0 0 0 1	I/O	TRIGGER
0 0 1 0	GATE	I/O
0 0 1 1	GATE	TRIGGER
0 1 0 0	I/O	C1 INPUT
0 1 0 1	TRIGGER	C1 INPUT
0 1 1 0	GATE	C1 INPUT
0 1 1 1	GATE/ TRIGGER	C1 INPUT
1 0 0 0	C1 OUTPUT	I/O
1 0 0 1	C1 OUTPUT	TRIGGER
1 0 1 0	C1 OUTPUT	GATE
1 0 1 1	C1 OUTPUT	GATE/TRIGGER
1 1 0 0	C1 OUTPUT	C1 INPUT
1 1 0 1	UNDEFINED	
1 1 1 0	UNDEFINED	
1 1 1 1	UNDEFINED	

Figure 9-8. Counter 1 Mode Register

input is received while the counter/timer is counting (Counter/Timer not equal to 0). When this bit is cleared to 0, no reloading occurs.

Input Pin Assignments (D_4-D_7). This 4-bit field specifies the functionality of the port lines associated with the counter/timer. It also determines whether the counter/timer will monitor an external input (counting operation) or use the scaled internal processor clock (timing operation). The four bits in the field select the following options: enable output (EO), external signal or internal clock (C/T), enable gate facility (G), and enable triggering facility (T). The selected options determine the functions associated with each external line of the counter/timer as illustrated in Table 9-1. A hardware reset forces these four pins to 0.

If 1111 is coded in this field in the Counter 0 Mode register, then the two counter/timers are linked together as a 32-bit counter with Counter 0 as the low-order 16 bits and Counter 1 as the high-order 16 bits. Counter 1 selects the mode and control options for the 32-bit counter and external accesses are made through the lines associated with Counter 1 ($P3_6$ and $P3_7$).

Table 9-1. IPA Field Encoding in Counter Mode Registers

IPA Field				— Pin Functionality —		
EO D_7	C/T D_6	G D_5	T D_4	Counter/Timer I/O ($P2_7$ or $P3_7$)*	Counter/Timer Input ($P2_6$ or $P3_6$)*	Notes
0	0	0	0	I/O	I/O	Timer
0	0	0	1	I/O	Trigger	Timer
0	0	1	0	Gate	I/O	Timer
0	0	1	1	Gate	Trigger	Timer
0	1	0	0	I/O	Input	Counter
0	1	0	1	Trigger	Input	Counter
0	1	1	0	Gate	Input	Counter
0	1	1	1	Gate/trigger	Input	Counter
1	0	0	0	Output	I/O	Timer
1	0	0	1	Output	Trigger	Timer
1	0	1	0	Output	Gate	Timer
1	0	1	1	Output	Gate/trigger	Timer
1	1	0	0	Output	Input	Counter
1	1	0	1	Undefined	Undefined	Reserved
1	1	1	0	Undefined	Undefined	Reserved
1	1	1	1	Undefined	Undefined	Reserved for Counter 1, Cascade for Counter 0

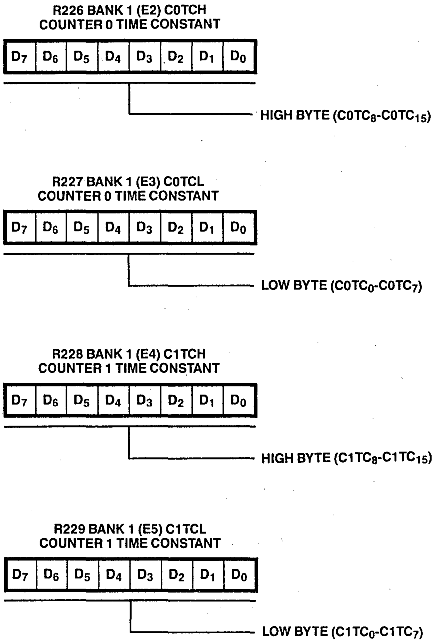
* Counter/timer 0 - $P2_7$ and $P2_6$
Counter/timer 1 - $P3_7$ and $P3_6$

The counter/timer I/O line ($P2_7$ for C/T0, $P3_7$ for C/T1) is also used as the external capture input if the capture feature is enabled, and the up/down control input (0=down, 1=up) if external up/down control is enabled.

9.2.3 Time Constant Register

This 16-bit register pair holds the value that is automatically loaded into the counter/timer 1) when the counter/timer is enabled, 2) in continuous mode, when the count reaches zero, or 3) in re-trigger mode, when the trigger is asserted. If capture on both edges is enabled, then this register captures the contents of the counter on the falling edge of the I/O pin.

The format of the Time Constant register is illustrated in Figure 9-9.



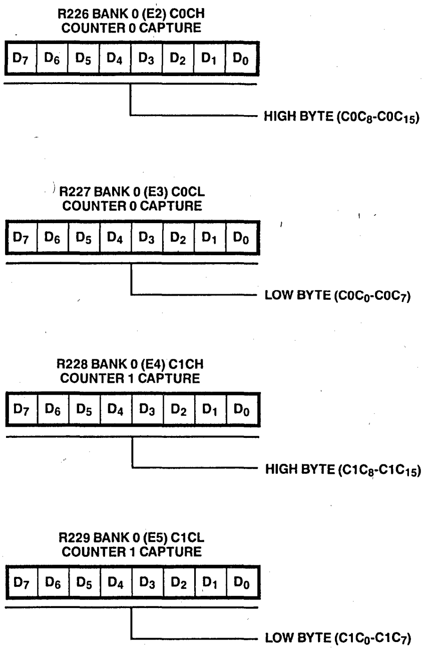
R226 BANK 1 (E2) C0TCH
COUNTER 0 TIME CONSTANT
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
HIGH BYTE ($C0TC_8$-$C0TC_{15}$)
R227 BANK 1 (E3) C0TCL
COUNTER 0 TIME CONSTANT
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
LOW BYTE ($C0TC_0$-$C0TC_7$)
R228 BANK 1 (E4) C1TCH
COUNTER 1 TIME CONSTANT
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
HIGH BYTE ($C1TC_8$-$C1TC_{15}$)
R229 BANK 1 (E5) C1TCL
COUNTER 1 TIME CONSTANT
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
LOW BYTE ($C1TC_0$-$C1TC_7$)

Figure 9-9. Time Constant Register Format

9.2.4 Capture Register

This 16-bit register pair is used to hold the counter value saved when using the "capture on external event" function. This register will capture at the rising edge of the I/O pin or when software capture is asserted. When the bi-value mode of operation is enabled, this register is used as a second Time Constant register and the counter is alternately loaded from each.

The format of the Capture Register is shown in Figure 9-10.



R226 BANK 0 (E2) C0CH
COUNTER 0 CAPTURE
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
HIGH BYTE ($C0C_8$-$C0C_{15}$)
R227 BANK 0 (E3) C0CL
COUNTER 0 CAPTURE
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
LOW BYTE ($C0C_0$-$C0C_7$)
R228 BANK 0 (E4) C1CH
COUNTER 1 CAPTURE
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
HIGH BYTE ($C1C_8$-$C1C_{15}$)
R229 BANK 0 (E5) C1CL
COUNTER 1 CAPTURE
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
LOW BYTE ($C1C_0$-$C1C_7$)

Figure 9-10. Capture Register Format

Chapter 10
UART

10.1 INTRODUCTION

The universal asynchronous receiver/transmitter (UART) is a full-duplex asynchronous channel. Transmission and reception can be accomplished independently with 5 to 8 data bits per character, plus optional even or odd parity, and an optional wake-up bit.

Data can be read into or out of the UART via R239. This single address is able to serve a full-duplex channel because it contains two complete 8-bit registers--one for the transmitter and the other for the receiver.

10.2 TRANSMITTER

When the UART's register address is specified as the destination (dst) of an operation, the data is output on the UART. The UART automatically adds the start bit, the programmed parity bit (odd, even, or no parity), and the programmed number of stop bits to the data character to be transmitted. The transmitter can also add a Wake-Up bit (optional) between the parity bit (or the last bit in the character if parity is disabled) and the first stop bit, as shown in Figure 10-1. When the character is five, six, or seven bits long, the unused bits in the Transmit Data register (UIO) are automatically ignored by the UART.

Serial data is shifted from the transmitter at a rate equal to 1, 1/16th, 1/32nd, or 1/64th of the clock rate supplied to the transmitter clock input (as determined by the clock-rate field in the UMA register). Serial data is shifted out on the falling edge of the transmitter clock.

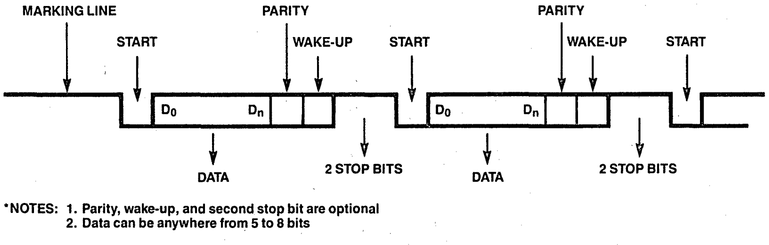
The Transmit Data output ($P3_1$) line is held marking (high) when the transmitter has no data to send. If the Send Break (SENBRK) bit of the UART Transmit Control (UTC) register is set to 1, the Data Output line will be held spacing (low) until it is cleared.

10.3 RECEIVER

An asynchronous receive operation begins when the Receive Enable bit (RENB) in the UART Receive Control register (URC) is set. A low (spacing) condition on the Receive Data line ($P3_0$) indicates a start bit. If this low persists for at least one-half of a bit time, the start bit is assumed to be valid and the data input is then sampled at the middle of each bit time until the entire character is assembled and placed in the Receive Data (UIOR) register. This method of detecting a start bit improves error rejection when noise spikes exist on an otherwise marking line.

If X1 clock mode is selected, bit synchronization must be accomplished externally, and the received data is sampled on the rising edge of the clock input.

A received character can be read from the 8-bit Receive Data register (UIOR). The receiver inserts 1s into the unused bits when a character length of other than eight bits is used. If parity is enabled, the parity bit is not stripped from the assembled character for character lengths less than eight bits; i.e., for lengths less than eight bits, the receiver assembles a character for the required number of data bits, plus a parity bit, wake-up bit, and 1s for any unused bits, and places it in the UART Data register (UIO).



*NOTES: 1. Parity, wake-up, and second stop bit are optional
2. Data can be anywhere from 5 to 8 bits

Figure 10-1. Asynchronous Transmission Data Format

Since the receiver is buffered by one 8-bit register in addition to the Receive Data register, the CPU has enough time to service an interrupt and to accept the data character assembled by the UART. The receiver also has a buffer that stores error flags for each data character in the receive buffer. These error flags are loaded at the same time as the data character.

After a character is received, it is checked for the following conditions:

- If the received character is an ASCII control character, it sets the Control Character Detect (CCD) bit in the UART Receive Control (URC) register. (An ASCII control character is any character that has bits 5 and 6 cleared to 0.) It can also cause an interrupt if the Control Character Interrupt Enable (CCIE) bit in the UART Interrupt Enable (UIE) register is set to 1. Once this bit is set, it remains set until cleared by software.

- The wake-up settings are checked and any indicated action is completed. In wake-up mode, the CPU can be selectively interrupted on a match condition that includes all of the eight bits in the received character and a Wake-Up bit. The Wake-Up bit match and character match can be enabled simultaneously or individually. Each bit in this character match can also be masked individually. (For more discussion of this feature, see section 10.4.) Once this bit is set, it remains set until cleared by software.

- If parity is enabled, the Parity Error bit (PERR) in the UART Receive Control (URC) register is set to 1 whenever the parity bit of the character does not match the programmed parity. Once this bit is set, it remains set until cleared by software.

- The Framing Error bit (FERR) in the URC register is set to 1 if the character is assembled without any stop bits (i.e., a low level is detected for a stop bit) and it is set with the character on which it occurs. It stays latched until cleared by software.

- If the CPU fails to read a data character when more than one character has been received, the Receive Overrun Error bit (OVERR) in the URC is set to 1. When this occurs, the new character assembled replaces the previous character in the Receive Data register. With this arrangement, only the overwriting character is flagged with the Receive Overrun Error. Like the Parity Error bit, this bit can be cleared only by software command from the CPU.

10.4 WAKE-UP FEATURE

The Super8 offers a powerful scheme to configure the UART receiver to interrupt only on certain special match conditions. Figure 10-2 shows the logic diagram for the scheme.

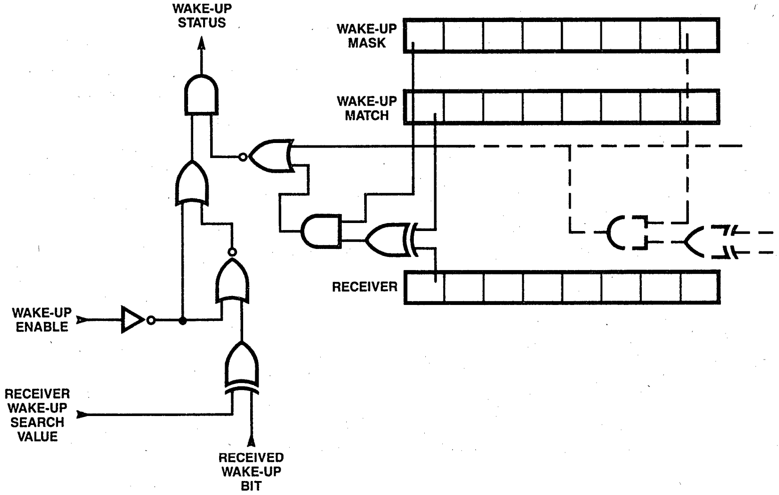


Figure 10-2. Logic Diagram for Wake-Up Feature

The pattern match logic can be used with or without the Wake-Up bit. The Wake-Up Match register and Wake-Up Mask register determine the character or characters that will generate a pattern match when detected at the receiver. If the Wake-Up bit is enabled, the pattern match occurs if the Wake-Up bit in the received character matches a pre-determined value, and the received character matches the value(s) specified in the Wake-Up Match and Wake-Up Mask registers. If the Wake-Up bit is disabled, the pattern match depends only on the character's value.

The Receive Data (UIOR) register is the receive buffer that is loaded if a new character is received and the previous character has been read by the CPU. The Wake-Up Match (WUMCH) register contains the match value. The Wake-Up Mask (WUMSK) register is used to mask out any selected bit positions in the WUMCH register. The Wake-Up Enable (WUENB) bit in the UART Transmit Control (UTC) register is enabled only if a match for the Wake-Up bit is also desired. If this is disabled, the scheme can still be used to look for a character match. The Receive Wake-Up Value (RWUVAL) bit in UART Mode A (UMA) register is the expected value of the Wake-Up bit; the Received Wake-Up bit (RWUIN) is the Wake-Up bit value received by the receiver.

The following cases show how the Wake-Up Detect (WUD) bit in the UART Receive Control (URC) register can be set by a match condition. However, the CPU is interrupted only if the Wake-Up Interrupt Enable (WUIE) bit in the UART Interrupt Enable (UIE) register is set to 1.

Case 1: WUENB = 1 (Wake-Up bit is enabled)

a) If Wake-Up bit match and WUMCH match (all 8 bits) is desired:

Set WUMSK = 1111 1111 (%FF)
WUMCH = ____ ____ (desired match value)

If WUMCH (bits 7-0) = UIO (bits 7-0) and
RWUVAL = RWUIN

Then Wake-Up Detect (WUD) flag is set.

b) If Wake-Up bit match and WUMCH match (selected bit, i.e., bits 5, 4, 1, 0) is desired:

Set WUMSK = 0011 0011 (%33)
WUMCH = XX__ XX__ (desired match bits 5, 4, 1, 0)

If WUMCH (bits 5, 4, 1, 0) = UIO (bits 5, 4, 1, 0) and
RWUVAL = RWUIN

Then Wake-Up Detect (WUD) flag is set.

c) If only a Wake-Up bit match is desired:

Set WUMSK = 0000 0000 (%00)
WUMCH = XXXX XXXX (don't care)

If RWUVAL = RWUIN

Then Wake-Up Detect (WUD) flag is set.

Case 2: MUENB = 0 (Wake-Up bit is ignored)

a) If a match is desired for WUMCH (all 8 bits):

Set WUMSK = 1111 1111 (%FF)
WUMCH = ____ ____ (desired match value)

If WUMCH (bits 7-0) = UIO (bits 7-0)

Then Wake-Up Detect (WUD) flag is set.

b) If a match is desired on WUMCH (selected bits only, i.e., bits 4, 3, 2):

Set WUMSK = 0001 1100 (%1C)
WUMCH = XXX_ __XX (desired match bits 4, 3, 2)

If WUMCH (bits 4, 3, 2) = UIO (bits 4, 3, 2)

Then Wake-Up Detect (WUD) flag is set.

c) If a match is always desired:

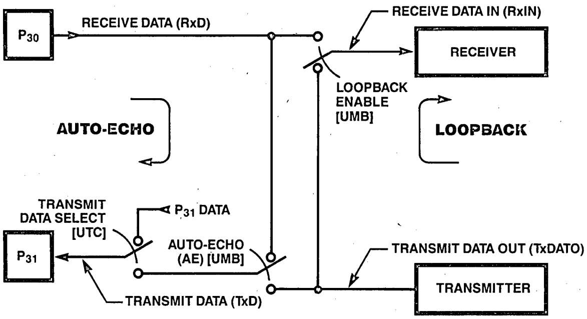
Set WUMSK = 0000 0000 (%00)
WUMCH = XXXX XXXX (don't care)

If this character is received, the Wake-Up Detect (WUD) flag is always set. However, this will be ignored if the Wake-Up Interrupt Enable (WUIE) bit in the UART Interrupt Enable (UIE) register is disabled.

10.5 AUTO-ECHO/LOOPBACK

As shown in Figure 10-3, the UART can be configured to automatically transmit any data coming in at the Receive Data input pin ($P3_0$) RXD. This auto-echo mode of operation is enabled by setting the Auto-Echo (AE) bit in the UART Mode B (UMB) register to 1. In addition, the Transmit Data Select (TXDTSEL) bit in the UART Transmit Control (UTC) register must be set to 1 for this mode to work correctly.

Similarly, the UART can be set in the local loopback mode by setting the Loopback Enable (LBENB) bit in the UMB register to 1. In loopback mode, the output of the transmitter is automatically routed to the receiver.



AE	LBENB	(TxDTSEL = 1)	
0	0	RxD → RxIN	TxDATO → TxD
0	1	TxDATO → RxIN	TxDATO → TxD
1	0	RxD → RxIN	RxD → TxD
1	1	TxDATO → RxIN	RxD → TxD

Figure 10-3. Auto-Echo/Loopback

In auto-echo mode, the transmitter can still be enabled; however, the transmitter data goes nowhere unless loopback is also enabled.

10.6 POLLED OPERATION

In a polled environment, the Receive Character Available (RCA) bit in the URC register must be monitored so the CPU can decide when to read a character. This bit is automatically cleared when the UIOR is read.

To prevent overwriting data in polled operations, the transmit buffer status must be checked before writing to the transmit buffer (UIOT). The Transmit Buffer Empty (TBE) bit in the UTC is set to 1 after completing the sending of a character.

10.7 BAUD-RATE GENERATOR

The UART has its own on-chip programmable baud-rate generator implemented as a 16-bit downcounter. The transmitter can receive its clocking signal from an external source ($P2_1$) or the baud-rate generator (BRG); the receiver clock can come from an external source ($P2_0$) or the on-chip baud-rate generator.

If $P2_1$ is not used as a Transmit Clock input, it can be used to output the transmit clock, the CPU clock, the output of the baud-rate generator, or as an I/O line.

The baud-rate generator consists of two 8-bit Time Constant registers, a 16-bit downcounter, and a flip-flop on the counter's output that produces a square wave.

On startup, the flip-flop is set to a high state, the value in the Time Constant registers is loaded into the Counter, and the Counter starts counting down. The output of the baud-rate generator toggles on reaching zero, the value in the Time Constant registers is again loaded into the Counter, and the process is repeated. The time constant can be changed at any time, but the new value does not take effect until the next load of the Counter.

As shown in Figure 10-4, the output of the baud-rate generator can be used as the receive clock, the transmit clock, or both. The transmitter and receiver can handle data at a rate of 1, 1/16th, 1/32nd, or 1/64th of the clock rate supplied to the receive and transmit clock inputs.

If $P2_1$ (Port 2, Bit 1) is not used as transmit clock input, it may be used as an output. A multiplexer (MUX) provided at $P2_1$ can be used to output various clocks or $P2_1$ data; bits 6 and 7 of the UMB register determine the function of P2 when it is used as an output.

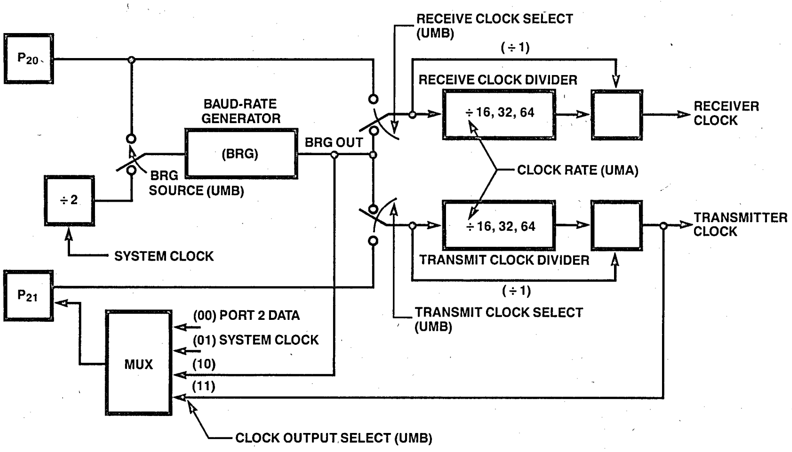


Figure 10-4. Baud-Rate Generator

10.8 UART INTERFACE PINS

The UART uses up to four Port 2 and 3 pins for interfacing with the external world. These are:

$P2_0$	Receive Clock
$P3_0$	Receive Data
$P2_1$	Transmit Clock
$P3_1$	Transmit Data

10.9 UART CONTROL/MODE AND STATUS REGISTERS

The following sections and figures describe the UART Control/Mode and Status registers.

10.9.1 UART Data Register (UIOT & UIOR)

Writing to this register automatically writes the data in the Transmit Data register (UIOT); a read from this register gets the data from the UART Receive Data register (UIOR). The format of this register is shown in Figure 10-5.

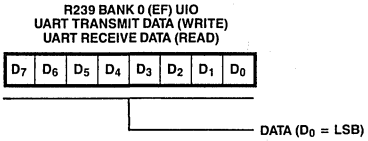


Figure 10-5. UART Data Register

10.9.2 Wake-Up Match Register (WUMCH)

Any character up to eight bits can be written into this register. The receiver detects a match between the received character and this character. The format of this register is shown in Figure 10-6.

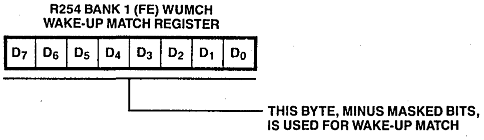


Figure 10-6. Wake-Up Match Register

10.9.3 Wake-Up Mask Register (WUMSK)

Any bit in the WUMCH register can be masked by writing a 0 into the corresponding bit in this register. The format of this register is shown in Figure 10-7.

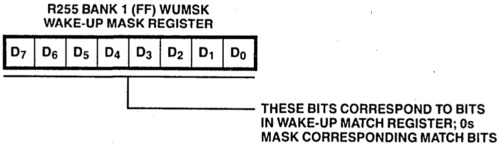

R255 BANK 1 (FF) WUMSK
WAKE-UP MASK REGISTER
D7 D6 D5 D4 D3 D2 D1 D0
THESE BITS CORRESPOND TO BITS IN WAKE-UP MATCH REGISTER; 0s MASK CORRESPONDING MATCH BITS

Figure 10-7. Wake-Up Mask Register

10.9.4 UART Receive Control Register (URC)

The fields in this register (Figure 10-8) are:

RCA. Receive Character Available (D_0). This is a status bit that is set to a 1 when data is available in the receive buffer (UIOR). When the CPU reads the receive buffer, it automatically clears this bit to 0. A write to this bit position has no effect. A hardware reset forces this bit to 0.

RENB. Receive Enable (D_1). When this bit is set to 1, the receive operation begins. This bit should be set only after all other receive parameters are established and the receiver is completely initialized. This bit is cleared to a 0 by a hardware reset, which disables the receiver.

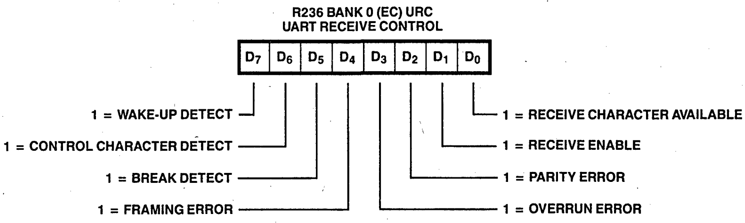


Figure 10-8. UART Receive Control Register

PERR. Parity Error (D$_2$). This is a status bit. When parity is enabled, this bit is set to 1 and buffered with the character whose parity does not match the programmed parity (even/odd). This bit is latched so that once an error occurs, it remains set until it is cleared to 0 by writing a 1 to this bit position. A hardware reset forces this bit to 0.

OVERR. Overrun Error (D$_3$). This status bit indicates that the receive buffer has not been read and another character has been received. Only the character that has been written over is flagged with this error; once set, this bit remains set until cleared to 0 by writing a 1 to this bit position. A hardware reset forces this bit to 0.

FERR. Framing Error (D$_4$). This is a status bit. If a framing error occurs (no stop bit where expected), this bit is set for the receive character in which the framing error occurred. This bit remains set until cleared to 0 by writing a 1 to this bit position. A hardware reset forces this bit to 0.

BRKD. Break Detect (D$_5$). This is a status bit that is set at the beginning and the end of a break sequence in the receive data stream. It stays set to 1 until cleared to 0 by writing a 1 to this bit position. A hardware reset forces this bit to 0. See note in section 10.9.5 for more information.

CCD. Control Character Detect (D$_6$). This status bit is set any time an ASCII control character is received in the receive data stream. It stays set until cleared to 0 by writing a 1 to this bit position. (An ASCII control character is any character that has bits 5 and 6 set to 0.) A hardware reset forces this bit to 0.

WUD. Wake-Up Detect (D$_7$). This status bit is set any time a valid wake-up condition is detected at the receiver. It stays set until cleared to 0 by writing a 1 to this bit position. The wake-up condition can be satisfied in many possible ways by the Wake-Up bit, Wake-Up Match register, and Wake-Up Mask register. See the Wake-Up Feature section (section 10.4) for a more detailed explanation. A hardware reset forces this bit to 0.

10.9.5 UART Interrupt Enable Register (UIE)

This register contains the individual status and data interrupt enables (Figure 10-9). The fields in this register are:

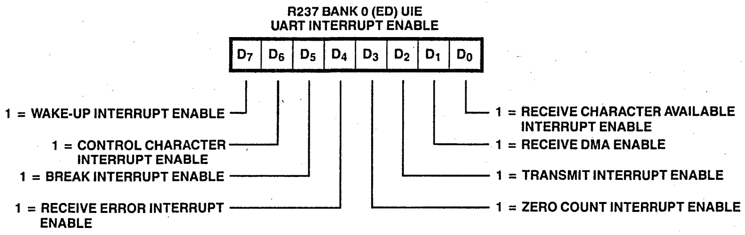

R237 BANK 0 (ED) UIE
UART INTERRUPT ENABLE
D7
D6
D5
D4
D3
D2
D1
D0
1 = WAKE-UP INTERRUPT ENABLE
1 = CONTROL CHARACTER INTERRUPT ENABLE
1 = BREAK INTERRUPT ENABLE
1 = RECEIVE ERROR INTERRUPT ENABLE
1 = RECEIVE CHARACTER AVAILABLE INTERRUPT ENABLE
1 = RECEIVE DMA ENABLE
1 = TRANSMIT INTERRUPT ENABLE
1 = ZERO COUNT INTERRUPT ENABLE

Figure 10-9. UART Interrupt Enable Register

RCAIE. Receive Character Available Interrupt Enable (D$_0$). If this bit is set to 1, then a Receive Character Available status in the URC register will cause an interrupt request. In a DMA receive operation, if this bit is set to 1, then an interrupt request will be issued only if an End-of-Process (EOP) of the DMA counter is also set. If it is not set, a Receive Character Available status causes no interrupt. A hardware reset forces this bit to 0.

RDMAENB. Receive DMA Enable (D$_1$). When this bit is set to 1, the DMA function is enabled for the UART receiver. Whenever a Receive Character Available signal in the URC register is true, a DMA request will be made. When the DMA channel gains control of the bus, it will transfer the received data to the register file or the external memory. A hardware reset forces this bit to 0.

TIE. Transmit Interrupt Enable (D$_2$). If this bit is set to 1, then a Transmit Buffer Empty signal in the UTC register will cause an interrupt request. In a DMA transmit operation, if this bit is set to 1, then an interrupt request will be issued only if an End-of-Process (EOP) of the DMA counter is also set. If it is not set, a Transmit Buffer Empty signal causes no interrupt. A hardware reset forces this bit to 0.

ZCIE. Zero Count Interrupt Enable (D$_3$). If this bit is set to 1, a baud-rate generator Zero Count status in the UTC register will cause an interrupt request. A hardware reset forces this bit to 0.

REIE. Receive Error Interrupt Enable (D_4). If this bit is set to 1, any receive error condition will cause an interrupt request. Possible receive error conditions include parity error, overrun error, and framing error. A hardware reset forces this bit to 0.

BRKIE. Break Interrupt Enable (D_5). If this bit is set to 1, a transition in either direction on the break signal will cause an interrupt request. A hardware reset forces this bit to 0.

Note: A break signal is a sequence of 0s. When all the required bits, parity bit, wake-up bit, and stop bits are 0s, the receiver immediately recognizes a break condition (not a framing error) and causes Break Detect (BRKD) to be set and an interrupt request. At the end of the break signal, a zero character is loaded into the Receive Data register (UIOR) and Break Detect (BRKD) is set again, along with another interrupt request.

CCIE. Control Character Interrupt Enable (D_6). If this bit is set to 1, then an ASCII Control Character Detect signal in the URC register will cause an interrupt. A hardware reset forces this bit to 0.

WUIE. Wake-Up Interrupt Enable (D_7). If this bit is set to 1, then any of the wake-up conditions that set the Wake-Up Detect bit (WUD) in the URC register will cause an interrupt request. A hardware reset forces this bit to 0.

10.9.6 UART Mode A Register (UMA)

This register controls the configurations of the receiver/transmitter that are not likely to change on a dynamic basis. The fields in this register (Figure 10-10) are:

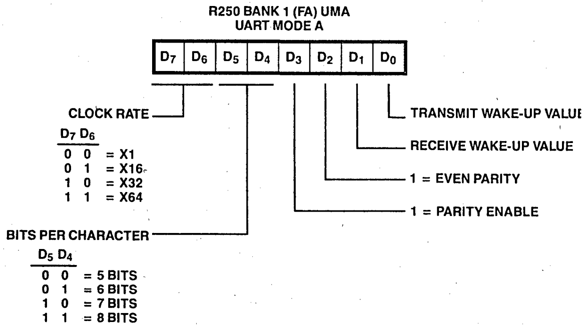


Figure 10-10. UART Mode A Register

TWUVAL. Transmit Wake-Up Value (D_0). If the wake-up mode is enabled, then the value in this bit position is transmitted along with the character at the appropriate time by the transmitter.

RWUVAL. Receive Wake-Up Value (D_1). If the wake-up mode is enabled, then the receiver expects a wake-up bit after the parity bit in the incoming data stream and the value is compared with this bit value. For further explanation of how this is used, see the Wake-Up Feature section (Section 10.4).

EVNPAR. Even Parity (D_2). This bit determines the type of parity used by both the receiver and the transmitter. If this bit is set to 0, odd parity is used; if this bit is set to 1, then even parity is used. If the Parity Enable (PARENB) bit in this register is not enabled, then this bit has no effect.

PARENB. Parity Enable (D_3). When this bit is set to 1, an additional bit position beyond those specified in the bits/character control is added to the transmitted data and is expected in the received data. The received parity bit is transferred to the CPU as a part of the data unless eight bits per character are used. If this bit is set to 0, the parity feature is disabled.

BPC1, BPC0. Bits Per Character (D$_5$, D$_4$). This field determines the number of bits per character for both the transmit and the receive sections. The character bits are right-justified with the least significant bit transmitted or received first. The field is coded as shown in Table 10-1.

Table 10-1. Character Size Field Encoding

D$_5$	D$_4$	Character Size in Bits
0	0	5
0	1	6
1	0	7
1	1	8

CR1, CR0. Clock Rate (D$_7$, D$_6$). This field specifies the multiplier between the clock and the data rates. Table 10-2 shows how this field is coded.

Table 10-2. Clock Rate Field Encoding

D$_7$	D$_6$	Mode	Description
0	0	1 x	Clock rate = 1 x data rate
0	1	16 x	Clock rate = 16 x data rate
1	0	32 x	Clock rate = 32 x data rate
1	1	64 x	Clock rate = 64 x data rate

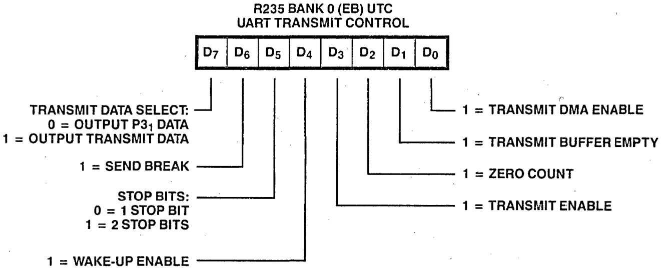


Figure 10-11. UART Transmit Control Register

10.9.7 UART Transmit Control Register (UTC)

This register contains the status and command bits needed to control the transmit section of the UART. The fields in this register (Figure 10-11) are:

TDMAENB. Transmit DMA Enable (D$_0$). When this bit is set to 1, it enables the DMA function for the UART transmit section. If this bit is set and the Transmit Buffer Empty signal becomes true, then a DMA request is made. When the DMA channel gains control of the bus, it transfers bytes from the external memory or the register file to the UART transmit section. A hardware reset forces this bit to 0.

TBE. Transmit Buffer Empty (D$_1$). This status bit is set to 1 whenever the transmit buffer is empty. It is cleared to 0 when a data byte is written in the transmit buffer. A hardware reset forces this bit to 1.

ZC. Zero Count (D$_2$). This status bit is set to 1 and latched when the Counter in the baud-rate generator reaches the count of 0. This bit can be cleared to 0 by writing a 1 to this bit position. A hardware reset forces this bit to 0.

TENB. Transmit Enable (D$_3$). Data is not transmitted until this bit is set to 1. When cleared to 0, the Transmit Data pin continuously outputs 1s unless Auto-Echo mode is selected. This bit should be cleared only after the desired transmission of data in the buffer is completed. A hardware reset forces this bit to 0.

WUENB. Wake-Up Enable (D$_4$). If this bit is set to 1, wake-up mode is enabled for both the transmitter and the receiver. The transmitter adds a bit beyond those specified by the bits/character and the parity. This added bit has the value specified in the Transmit Wake-Up Value (TWUVAL) in the UMA register. The receiver expects a Wake-Up bit value in the incoming data stream after the parity bit and compares this value with that specified in the Received Wake-Up Value (RWUVAL) bit in the UMA register. The resulting action depends on the configuration of the Wake-Up feature. A more complete description is given in the Wake-Up Feature section (section 10.4). A hardware reset forces this bit to 0.

STPBTS. Stop Bits (D$_5$). This bit determines the number of stop bits added to each character transmitted from the UART transmit section. If this bit is a 0, then one stop bit is added. If this bit is a 1, then two stop bits are added. The receiver always checks for at least one stop bit. A hardware reset forces this bit to 0.

SENBRK. Send Break (D_6). When set to 1, this bit forces the transmit section to continuously output 0s, beginning with the following transmit clock, regardless of any data being transmitted at the time. This bit functions whether or not the transmitter is enabled. When this bit is cleared to 0, the transmit section continues to send the contents of the Transmit Data register. A hardware reset forces this bit to 0.

TXDTSEL. Transmit Data Select (D_7). This bit has an effect only if port pin $P3_1$ is configured as an output. If this bit is set to 1, the serial data coming out of the transmit section is reflected on the $P3_1$ pin. If this bit is set to 0, then $P3_1$ acts as a normal port and $P3_1$ data is reflected on the $P3_1$ pin. A hardware reset forces this bit to 0.

10.9.8 UART Mode B Register (UMB)

This register (Figure 10-12) contains the necessary status and command bits for the baud-rate generator, transmit clock select, auto-echo and loopback enable. The fields are as follows:

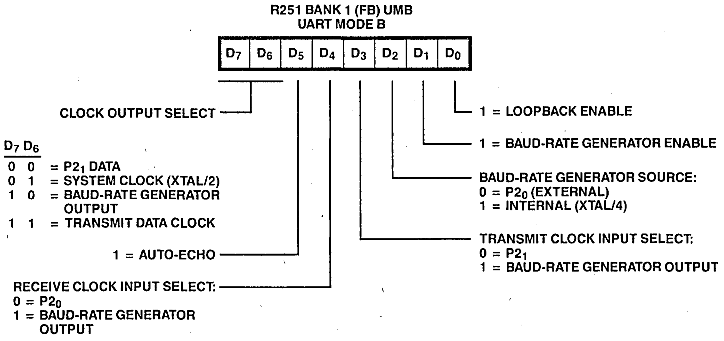

R251 BANK 1 (FB) UMB
UART MODE B
D_7
D_6
D_5
D_4
D_3
D_2
D_1
D_0
CLOCK OUTPUT SELECT
D_7 D_6
0 0 = $P2_1$ DATA
0 1 = SYSTEM CLOCK (XTAL/2)
1 0 = BAUD-RATE GENERATOR OUTPUT
1 1 = TRANSMIT DATA CLOCK
1 = AUTO-ECHO
RECEIVE CLOCK INPUT SELECT:
0 = $P2_0$
1 = BAUD-RATE GENERATOR OUTPUT
1 = LOOPBACK ENABLE
1 = BAUD-RATE GENERATOR ENABLE
BAUD-RATE GENERATOR SOURCE:
0 = $P2_0$ (EXTERNAL)
1 = INTERNAL (XTAL/4)
TRANSMIT CLOCK INPUT SELECT:
0 = $P2_1$
1 = BAUD-RATE GENERATOR OUTPUT

Figure 10-12. UART Mode B Register

LBENB. Loopback Enable (D_0). Setting this bit to 1 selects the local loopback mode of operation. In this mode, the data output from the transmit section is also routed back to the receive section. For meaningful results, the frequency of the transmit and receive clocks must be the same. A hardware reset forces this bit to 0.

BRGENB. Baud-Rate Generator Enable (D_1). This bit controls the operation of the baud-rate generator. The Counter in the baud-rate generator is enabled for counting when this bit is set to 1 and disabled for counting when this bit is set to 0. A hardware reset forces this bit to 0.

BRGSRC. Baud-Rate Generator Source (D_2). This bit selects the source of the clock for the baud-rate generator. If this bit is set to 0, the baud-rate generator clock comes from the receive clock pin ($P2_0$). If this bit is set to 1, the clock for the baud-rate generator is the CPU clock divided by two (XTAL clock divided by four). A hardware reset forces this bit to 0.

TCIS. Transmit Clock Input Select (D_3). This bit selects the source for the transmit section clock input. If TCIS is cleared to 0, the source is the transmit clock pin ($P2_1$). If it is set to 1, then the source is the baud-rate generator output. A hardware reset forces this bit to 0.

RCIS. Receive Clock Input Select (D_4). This bit selects the source for the receive section clock input. If this bit is cleared to 0, the source is the receive clock pin ($P2_0$). If it is set to 1, then the source is the baud-rate generator output. A hardware reset forces this bit to 0.

AE. Auto-Echo (D_5). Auto-echo mode of operation is enabled by setting this bit to 1. In this mode, the data coming in on the receive data pin is reflected out on the transmit data pin. The receive section still listens to the receive data input; however, the data from the transmit section goes nowhere. See section 10.6 for a more detailed description of this function. A hardware reset forces this bit to 0.

COS1, COSO. Clock Output Select (D_7-D_6). This field determines the source that drives the transmit clock pin if $P2_1$ is configured as an output. A hardware reset forces this field to 00. Table 10-3 shows the coding of this field.

Table 10-3. Transmit Clock Source Field Encoding

D_7	D_6	Output Source
0	0	$P2_1$ Data
0	1	System clock (XTAL frequency divided by 2)
1	0	Baud-rate generator output
1	1	Transmit data rate

10.9.9 UART Baud-Rate Generator Time Constant Register (UBG)

This register contains the high and low bytes (Figure 10-13) for the 16-bit time constant used to generate the desired baud rate. The time constant can be changed at any time, but the new value does not take effect until the next time constant is loaded into the downcounter.

The formula for determining the appropriate time constant for a given baud rate is shown below, with the desired rate in bits per second and the baud-rate clock period in seconds.

$$\text{time constant} = \frac{1}{(2 \times \text{baud rate} \times n \times \text{BRG input clock period})} - 1$$

where n=1,16,32,or 64 x the clock rate selected in UMA register R250

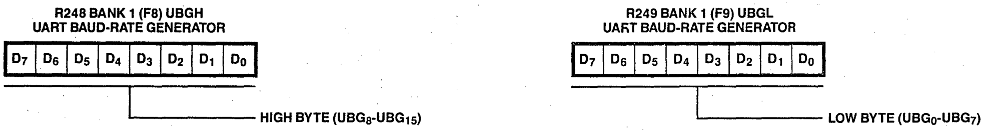


Figure 10-13. UART Baud-Rate Generator Time Constant Register

Chapter 11
DMA Channel

11.1 INTRODUCTION

The Super8 has an on-chip Direct Memory Access (DMA) channel to provide high bandwidth data transmission capabilities that can be used by the UART receive or transmit section or by Handshake Channel 0.

The DMA channel can transfer data between the peripheral device and contiguous locations in either the register file or external data memory.

UART Receiver	------>	Register file or data memory
UART Transmitter	<------	Register file or data memory
Handshake Channel 0	<------	Register file or data memory
Handshake Channel 0	------>	Register file or data memory

Prior to enabling the DMA channel, the starting register address for the block to be transferred must be present in register $C1_H$ or the starting memory address must be present in register $C0_H$ (high byte) and $C1_H$ (low byte). Registers $C0_H$ and $C1_H$ themselves can only be accessed as part of the working register group. The address is auto-incremented after each DMA-controlled transfer.

The DMA Count registers (R240 and R241, Bank 1) hold the 16-bit count that determines the number of transactions the DMA channel is to perform. The count loaded should be n-1 to perform n byte transfers. An interrupt can be generated when the count is exhausted.

DMA transfers to or from the register file take six CPU clock cycles; DMA transfers to or from memory take ten CPU clock cycles, excluding wait states.

11.2 DMA CONTROL REGISTERS

The control bits that link the DMA channel to the UART or an I/O port are the Transmit DMA Enable (TDMAENB) bit in the UART Transmit Control (UTC) register for the transmitter, the Receive DMA Enable (RDMAENB) bit in the UART Interrupt Enable (UIE) register for the receiver, and the DMA Enable bit (D_2) in the Handshake 0 Control register for the I/O ports. Only one of these three enable bits should be set at a given time. If Handshake Channel 0 is linked to the DMA channel, the data transfer direction is determined by the direction of the handshake.

A bit in the External Memory Timing register, called DMA INT/EXT, controls whether DMA transfers access the register file or external data memory. When this bit is cleared to 0, transfers are to/from the register file. When this bit is set to 1, transfers are to/from external data memory. See Figure 11-1.

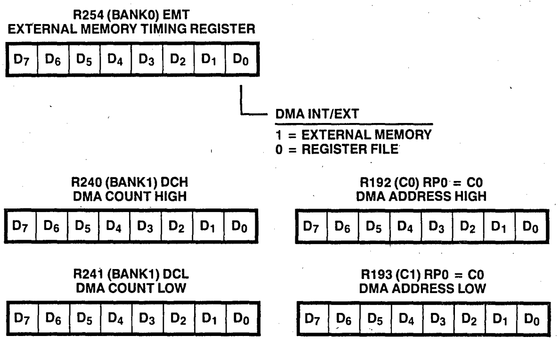


Figure 11-1. DMA Control Registers

11.3 DMA AND THE UART RECEIVER

The Receive DMA Enable bit (RDMAENB) in the UIE register (R237) of the UART is first set to 1 to link the DMA to the UART receiver.

Data received at the UART receiver is handled by the DMA as soon as the Receive Character Available (RCA) status bit of the URC register (R236) of the UART is set to 1. The DMA reads data from the UIO register of the UART and then clears the RCA bit to prepare the UART receiver to receive new data. The data is then stored at the location whose address is contained in the DMA address register (RR192). The DMA count at RR240, Bank 1, is decreased by 1 and the DMA address register is increased by 1. When the DMA count is negative, an interrupt request (IRQ6, vector address 20, 21) is generated at the UART Receive section if the Receive Character Available Interrupt Enable bit of the UIE register of the UART (R237) is set to 1.

The UART continues to receive new data and the DMA responds to the RCA bit as described above until an interrupt is generated due to a negative DMA count.

11.4 DMA AND THE UART TRANSMITTER

First, the Transmit DMA Enable (TDMAENB) bit of the UTC register (R235) of the UART is enabled to link the DMA to the UART transmitter.

Upon transmit, the Transmit Buffer Empty status bit (TBE) in the UTC register (R235) of the UART is set to 1. The DMA then transfers the data at the location whose address is contained in the DMA address register (RR192) to the UIO register (R239) of the UART.

The TBE bit is then cleared to 0. The DMA count at RR240, Bank 1, is decreased by 1 and the DMA address register is increased by 1. When the DMA count is negative, the DMA issues an End-of-Process (EOP) signal to the UART. The UART grants an interrupt request (IRQ1, vector address 26, 27) to the Super8 if the Transmit Interrupt Enable (TIE) bit of the UIE register (R237) of the UART is set to 1.

The UART transmitter continues its operation with the new data in the UIO register and the DMA responds to the TBE bit as described above until an interrupt is generated due to a negative DMA count.

11.5 DMA AND HANDSHAKE CHANNEL 0

The DMA can be configured with Handshake Channel 0 to transfer data from register file or data memory to I/O devices or vice versa through Port 1 or Port 4. Handshake Channel 0 can be in either fully interlocked mode or strobed mode as controlled by the Handshake 0 Control register (R244). The direction of DMA transfer is determined by the handshake direction, which is the direction of the chosen port.

11.5.1 DMA WRITE (INPUT HANDSHAKE CHANNEL 0)

The I/O device transfers data to register file or data memory through Handshake Channel 0 and the DMA channel.

The Handshake Channel 0 Enable and DMA Enable bits of the Handshake 0 Control (HOC) register (R244) should be first set to 1. When the I/O device puts data on the port specified in the HOC register and activates $\overline{\text{DAV}}$ to go from high to low as in Figures 8-11 and 8-13, the DMA transfers data on the port to the specified address in the DMA address register (RR192). The DMA count at RR240, Bank 1, is decreased by 1 and the DMA address register is increased by 1. When the DMA count is negative, the DMA issues an End-of-Process (EOP) signal to Handshake Channel 0. Handshake Channel 0 grants an interrupt request (IRQ4) to the Super8. The handshake output at pin 25 is the same as described in Figures 8-11 and 8-13 and the DMA is waiting for the I/O device to put data on the port and activate the $\overline{\text{DAV}}$ signal again.

11.5.2 DMA READ (OUTPUT HANDSHAKE CHANNEL 0)

Data is transferred from register file or data memory to the I/O device through the DMA channel and Handshake Channel 0.

The Handshake Channel 0 Enable and DMA Enable bits of the Handshake 0 Control (HOC) register (R244) should be first set to 1. The handshake direction should be set by choosing the direction of the port specified in the HOC register.

The DMA sequence should always begin by writing the first byte of data to the port to start the DMA. This is an important process, otherwise the DMA is not activated when Handshake Channel 0 is not yet activated. The DMA starting address in the DMA address register (RR192) should now be set at the second byte of the data block. The I/O device should then read that first byte of data and store it away as in Figures 8-12 and 8-14. The DMA is then activated.

11.5.2.1 FULLY INTERLOCKED MODE

At State 3 of Figure 8-12, the DMA reads the data at the address specified in the DMA address register (RR192) and transfers it to the port. The DMA count at RR240, Bank 1, is decreased by 1 and the DMA address register is increased by 1. When the DMA count is negative, the DMA issues an End-of-Process (EOP) signal to Handshake Channel 0. Handshake Channel 0 then grants an interrupt request (IRQ4) to the Super8.

The DMA and handshake process continues as in Figure 8-12 until an interrupt is caused by a negative DMA count.

11.5.2.2 STROBED MODE

After the first writing of the first byte of data to the port as in Figure 8-14, the DMA is activated at the end of strobe time. The DMA reads the data at the address specified in the DMA address register (RR192) and transfers it to the port. The DMA count at RR240, Bank 1, is decreased by 1 and the DMA address register is increased by 1. When the DMA count is negative, the DMA issues an End-of-Process (EOP) signal to Handshake Channel 0. Handshake Channel 0 then grants an interrupt request (IRQ4) to the Super8.

The handshake operation continues as in Figure 8-14 and the DMA transfers new data to the port only at the end of strobe time. The DMA stops when an interrupt is activated by a negative DMA count.

12.4 EXTERNAL STACKS

The Super8 architecture supports stack operations in either the register file or in data memory. A stack's location is determined by setting bit 1 in the External Memory Timing register, R254, Bank 0 (Figure 12-5).

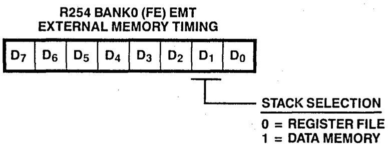


Figure 12-5. External Memory Timing

The instruction used to change the stack selection bit should not be immediately followed by an instruction that uses the stack, since this will cause indeterminate program flow. Interrupts should be disabled when changing the stack selection bit.

12.5 DATA MEMORY

The two external memory spaces, data and program, can be addressed as a single memory space or as two separate spaces. If the memory spaces are separated, program memory and data memory are logically selected by the Data Memory select output ($\overline{\text{DM}}$). $\overline{\text{DM}}$ is made available on Port 3, line 5 ($P3_5$) by setting bit D3 in the Port Mode register to 1 (Figure 12-6).

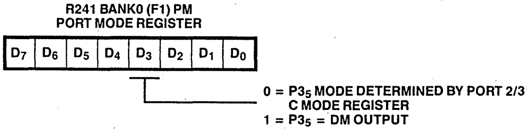


Figure 12-6. Data Memory

12.6 BUS OPERATION

Typical data transfers between the Super8 and external memory are illustrated in Figures 12-7 and 12-8. Machine cycles can vary from six to twelve external clock periods depending on the operation being performed. The notations used to describe the basic timing periods of the Super8 are machine cycles (Mn), timing states (Tn), and clock periods. All timing references are made with respect to the output signals $\overline{\text{AS}}$ and $\overline{\text{DS}}$. The clock is shown for clarity only and does not have specific timing relationships with other signals; the clock signal shown is the external clock, which has twice the frequency of the internal CPU clock.

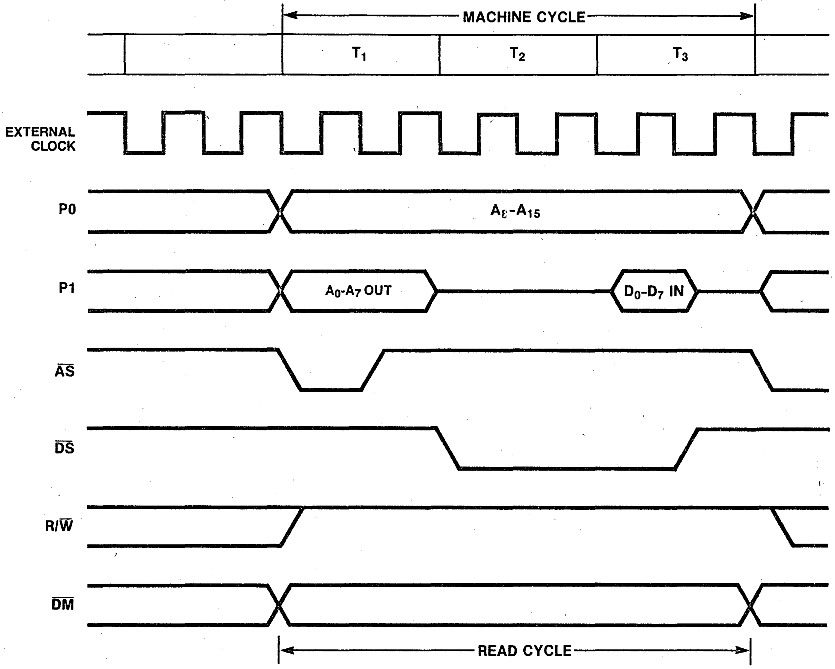


Figure 12-7. External Instruction Fetch or Memory Read Cycle

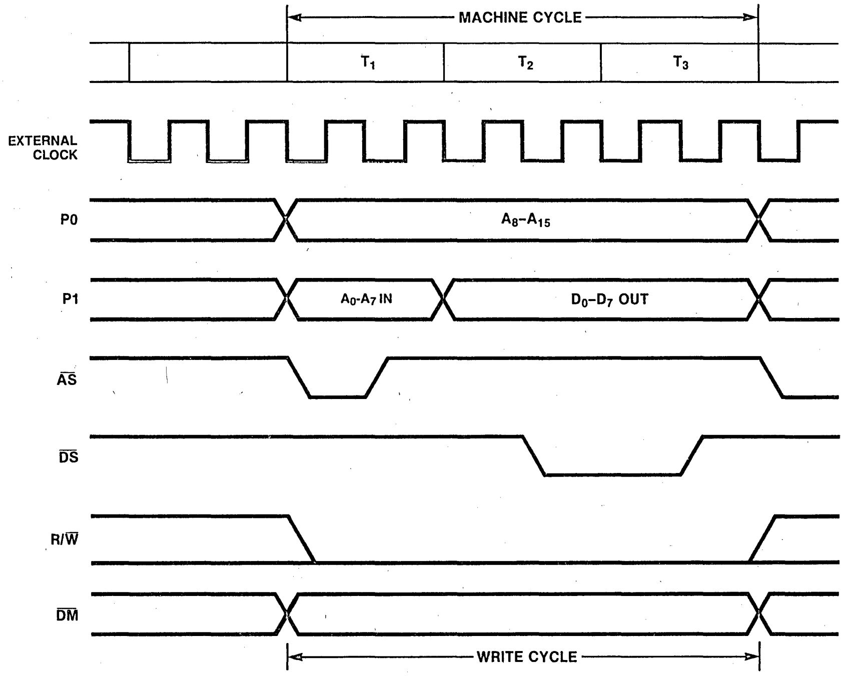


Figure 12-8. External Memory Write Cycle

12.6.1 Address Strobe (AS)

All transactions start with Address Strobe (AS) being driven low and then raised high by the Super8. The rising edge of AS indicates that Read/Write (R/W), Data Memory (DM), and the addresses output from Ports 0 and 1 are valid. The addresses output via Port 1 typically need to be latched during AS, whereas Port 0 address outputs, if used, remain stable throughout the machine cycle.

12.6.2 Data Strobe (DS)

The Super8 uses Data Strobe (DS) to time the actual data transfer. For write operations (R/W = low), a low on DS indicates that valid data is on the Port 1 AD_0-AD_7 lines. For read operations (R/W = high), the address/data bus is placed in a high-impedance state before driving DS low so that the addressed device can put its data on the bus. The Super8 samples this data prior to raising DS high.

12.6.3 External Memory Operations

Whenever the Super8 is configured for external memory operations, the addresses of all internal program memory references appear on the external bus. This should have no effect on the external system since the bus control line DS remains in its inactive high state. DS becomes active only during external memory references.

12.7 EXTENDED BUS TIMING

The Super8 can accommodate slow memory access and cycle times by three different methods that give the user much flexibility in the types of memory available.

12.7.1 Software Programmable Wait States

The Super8 can stretch the Data Strobe (DS) timing automatically by adding one, two, or three internal clock periods. This is under program control and applies only to external memory cycles. Internal memory cycles still operate at the maximum rate. The software has independent control over stretched Data Strobe for external memory (i.e., the software can set up one timing for program memory and a different timing for data memory). Thus, program and data memory may be made up of different kinds of hardware chips, each requiring its own timing.

12.7.2 Slow Memory Timing

Another feature of the Super8 that is useful in interfacing with slow memories is the Slow Memory Timing option. When this option is enabled, the normal external memory timing is slowed by a factor of two (bus clock = CPU clock divided by two). All memory times for set-up, duration, hold, and access times are essentially doubled. This feature can also be used with the programmed automatic wait states described above. Programmed wait states can still be used to stretch the Data Strobe time by one, two, or three internal clock times (not two, four, or six) when Slow Memory Timing is enabled.

12.7.3 Hardware Wait States

Still another Super8 feature is an optional external $\overline{\text{WAIT}}$ input using port pin $P3_4$. The $\overline{\text{WAIT}}$ input function can be used with either or both of the above two features. Thus the Data Strobe width will have a minimum value determined by the number of programmed wait states selected and/or by Slow Memory Timing. The $\overline{\text{WAIT}}$ input provides the means to stretch it even further. The $\overline{\text{WAIT}}$ input is sampled each internal clock time and, if held low, can stretch the Data Strobe by adding one internal clock period to the Data Strobe time for an indefinite period of time.

All of the extended bus timing features are programmed by writing the appropriate bits in the External Memory Timing register (Figure 12-9).

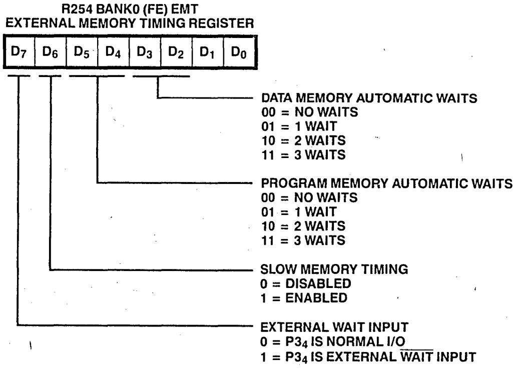


Figure 12-9. External Memory Timing Register

12.8 INSTRUCTION TIMING

The high throughput of the Super8 is due, in part, to the use of instruction pipelining, where the instruction fetch and execution cycles are overlapped. During the execution of the current instruction, the opcode of the next instruction is fetched, as illustrated in Figure 12-10.

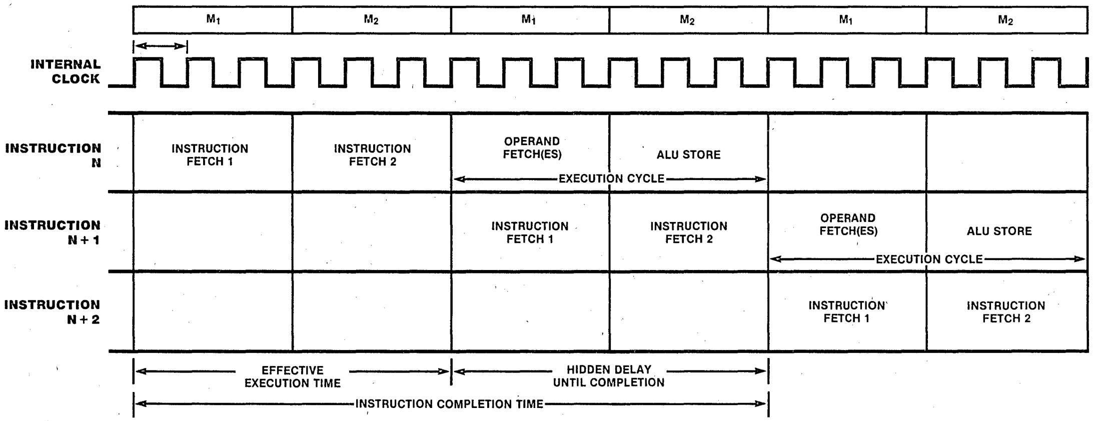


Figure 12-10. Instruction Pipelining

Figures 12-11 through 12-14 show typical instruction cycle timing for instructions fetched from external memory. All instruction fetch cycles have the same machine timing regardless of whether the memory is internal or external except when external memory timing is extended. In order to calculate the execution time of a program, the internal clock periods shown in the cycles column of the instruction formats in the Instruction Set (Chapter 5) should be added. Pipeline cycles are transparent to the user and should be ignored. Each cycle represents two cycles of the crystal or input clock.

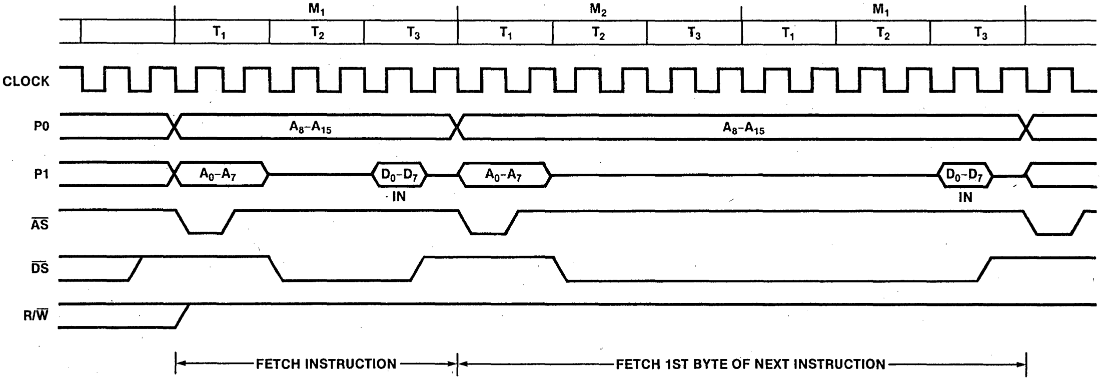


Figure 12-11. Typical Instruction Cycle Timing (One Byte Instruction)

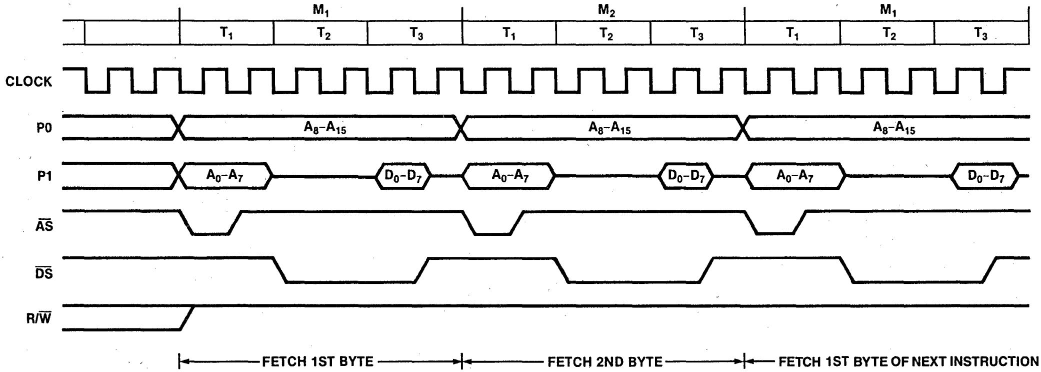


Figure 12-12. Typical Instruction Cycle Timing (Two Byte Instruction)

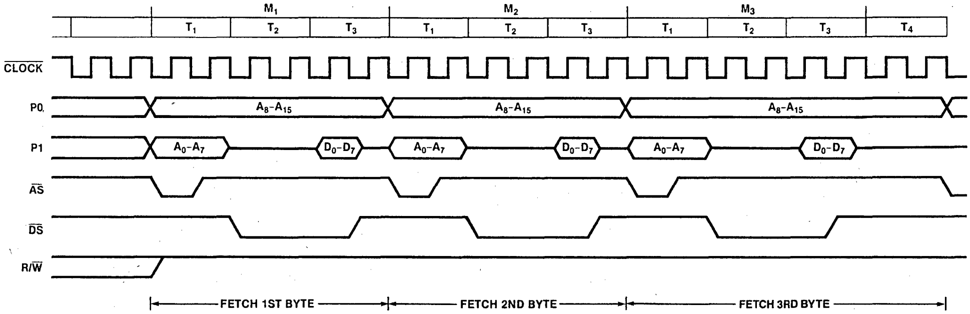


Figure 12-13. Typical Instruction Cycle Timing (Three Byte Instruction)

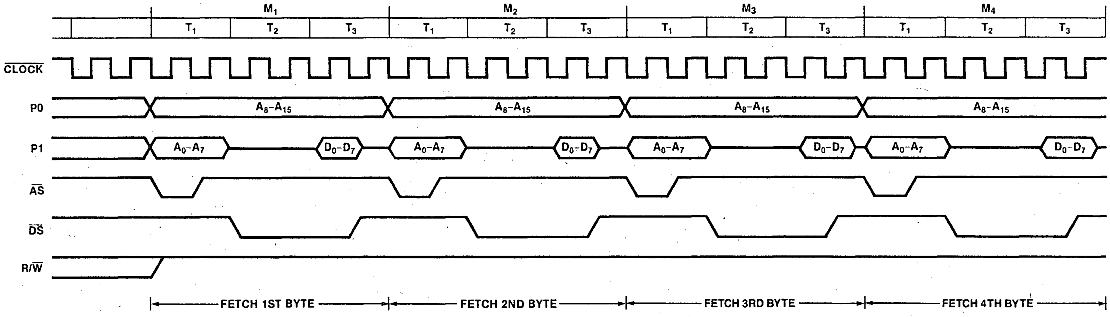


Figure 12-14. Typical Instruction Cycle Timing (Four Byte Instruction)

Chapter 12
External Interface

12.1 INTRODUCTION

The 48-pin Super8 has 40 programmable I/O pins, some of which are configurable as an external memory interface. A description of the pins and their functions follows (see Figure 12-1).

12.2 PIN DESCRIPTIONS

$\overline{AS}$. Address Strobe (output, active low, 3-state). $\overline{AS}$ is pulsed low once at the beginning of each machine cycle. For external memory accesses, the rising edge of $\overline{AS}$ indicates that addresses, R/$\overline{W}$, and $\overline{DM}$ signals are valid. Under program control, $\overline{AS}$ can be placed in a high impedance state along with Ports 0 and 1, $\overline{DS}$, R/$\overline{W}$, and $\overline{DM}$ if used.

$\overline{DS}$. Data Strobe (output, active low, 3-state). $\overline{DS}$ provides timing for data movement to or from Port 1 for each external memory transfer. During a write cycle, data out is valid at the leading edge of $\overline{DS}$; during a read cycle, data in is valid prior to the trailing edge of $\overline{DS}$. $\overline{DS}$ can be placed in a high-impedance state along with Ports 0 and 1, $\overline{AS}$, R/$\overline{W}$, and $\overline{DM}$ if used.

R/$\overline{W}$. Read/Write (output, 3-state). R/$\overline{W}$ determines the direction of data transfer for external memory transactions. R/$\overline{W}$ is low during write operations and high during all other operations. R/$\overline{W}$ can be placed in a high-impedance state along with Ports 0 and 1, $\overline{AS}$, $\overline{DS}$, and $\overline{DM}$ if used.

$P0_0$–$P0_7$, $P1_0$–$P1_7$, $P2_0$–$P2_7$, $P3_0$–$P3_7$, $P4_0$–$P4_7$. I/O Port Lines (inputs/outputs, TTL-compatible). These I/O lines provide five 8-bit I/O ports that can be configured under program control for I/O or external memory interfacing. Ports 0 and 1 can be placed in a high-impedance state under program control, along with $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$, and $\overline{DM}$ if used.

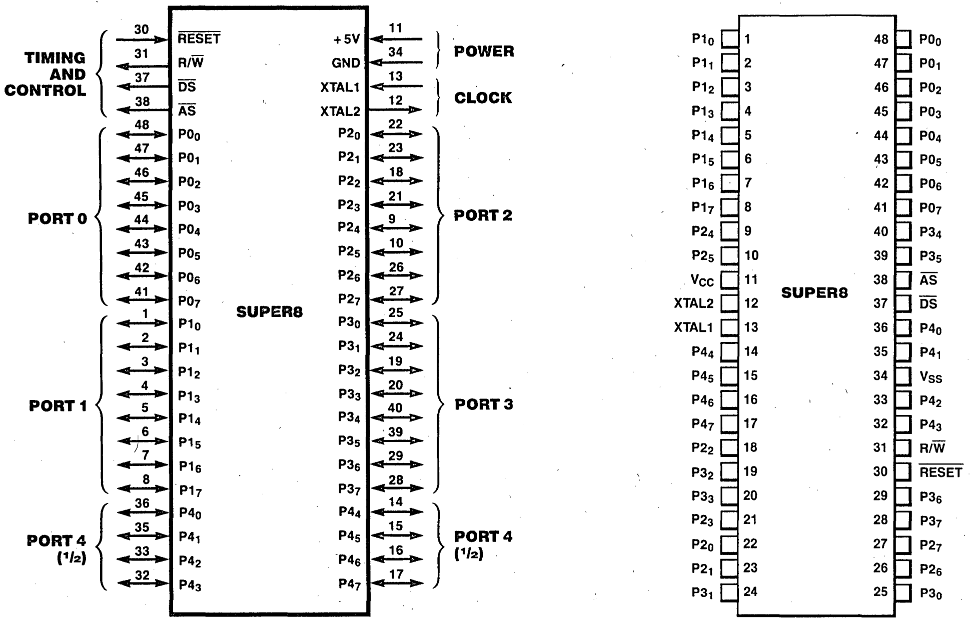


Figure 12-1. Pin Functions and Assignments

$\overline{RESET}$. Reset (input, active low). $\overline{RESET}$ is used to initialize the Super8. When $\overline{RESET}$ is deactivated, program execution begins from program address 0020_H. $\overline{RESET}$ is also used to enable the Super8 test mode.

XTAL1, XTAL2. Crystal (oscillator input/output). XTAL1 and XTAL2 are used to connect a parallel resonant crystal or external clock source to the on-board clock oscillator and buffer.

12.3 CONFIGURING FOR EXTERNAL MEMORY

Before external memory can be referenced in a ROM-based part, Ports 0 and 1 must be properly configured. The minimum bus configuration uses Port 1 as a multiplexed address/data bus (AD_0-AD_7) with access to 256 bytes of external memory. In this configuration, the eight lower order address bits (A_0-A_7) are multiplexed with the eight data bits (D_0-D_7).

Additional address lines can be output on the Port 0 pins, where bit 0 of that port corresponds to A_8, bit 1 to A_9, and so on. The pins of Port 0 can be defined as memory address lines or I/O lines on a bit-by-bit basis, via programming of the Port 0 Mode register (R240, Bank 0). This ensures the efficient use of the I/O pins, allowing the Super8 to address various sizes of external memory using no more pins than necessary. Port 0 pins not configured for address lines can be used as I/O lines.

Configuring Port 1 for external memory is accomplished by writing the appropriate bits in the Port Mode register, R241 in Bank 0 (Figure 12-2).

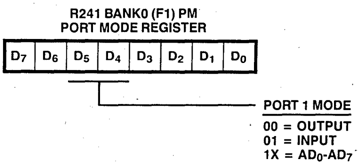


Figure 12-2. Configuring Port 1 for External Memory

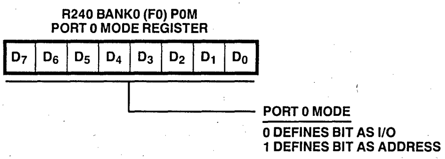


Figure 12-3. Configuring Port 0 for External Memory

Configuring Port 0 for external memory is accomplished in a similar manner, using Port 0 Mode Register, R240 in Bank 0 (Figure 12-3).

Once Port 1 is configured as an address/data port, it is no longer usable as a general-purpose I/O port. Attempting to read Port 1 returns "FF_H"; writing has no effect. Similarly, if Port 0 is configured for address lines A_8-A_{15}, it is no longer usable as a general-purpose I/O port; however, if not all of the bits are defined as address lines, the remaining bits are still accessible as an I/O port. Reading Port 0 will return the port data in those positions defined as I/O. The positions defined as address will return the value on the external pins which, under normal loading, will be the address.

> After setting the modes of Ports 0 and 1 for external memory, the next three bytes must be fetched from internal memory.

An external memory interface may be 3-stated under program control by setting bit 7 of the System Mode register, R222 (Figure 12-4).

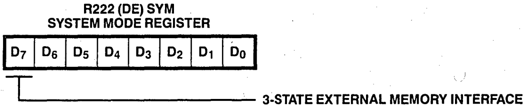


Figure 12-4. 3-State External Memory Interface

When this bit is set to 1, the external memory interface, including $\overline{AS}$, $\overline{DS}$, R/$\overline{W}$ and $\overline{DM}$, is 3-stated. A hardware reset forces this bit to a 0. The external memory interface can but should not be tri-stated in the ROMless parts.

In Super8 parts with on-chip ROM, a hardware reset configures Ports 0 and 1 as input ports and instruction execution begins at location 0020_H, which is within the on-chip ROM.

In the ROMless parts, a hardware reset configures Port 0 pins $P0_0$-$P0_4$ as address out and pins $P0_5$-$P0_7$ as inputs; Port 1 is configured as an address/data port, allowing access to 8 Kbytes of memory. If external memory greater than 8 Kbytes is desired, additional address lines must be configured in Port 0. Since Port 0 lines are initially configured as inputs, they will float and their logic state will be unknown until an initialization routine is executed that configures Port 0. This initialization routine must reside within the first 8 Kbytes of executable code and must be physically mapped into memory by externally forcing the Port 0 address lines to a known state.

12.4 EXTERNAL STACKS

The Super8 architecture supports stack operations in either the register file or in data memory. A stack's location is determined by setting bit 1 in the External Memory Timing register, R254, Bank 0 (Figure 12-5).

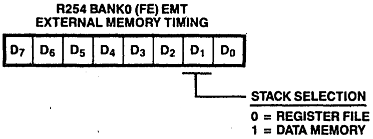


Figure 12-5. External Memory Timing

The instruction used to change the stack selection bit should not be immediately followed by an instruction that uses the stack, since this will cause indeterminate program flow. Interrupts should be disabled when changing the stack selection bit.

12.5 DATA MEMORY

The two external memory spaces, data and program, can be addressed as a single memory space or as two separate spaces. If the memory spaces are separated, program memory and data memory are logically selected by the Data Memory select output ($\overline{DM}$). $\overline{DM}$ is made available on Port 3, line 5 ($P3_5$) by setting bit D3 in the Port Mode register to 1 (Figure 12-6).

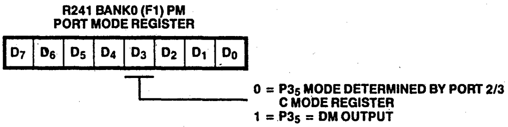


Figure 12-6. Data Memory

12.6 BUS OPERATION

Typical data transfers between the Super8 and external memory are illustrated in Figures 12-7 and 12-8. Machine cycles can vary from six to twelve external clock periods depending on the operation being performed. The notations used to describe the basic timing periods of the Super8 are machine cycles (Mn), timing states (Tn), and clock periods. All timing references are made with respect to the output signals $\overline{AS}$ and $\overline{DS}$. The clock is shown for clarity only and does not have specific timing relationships with other signals; the clock signal shown is the external clock, which has twice the frequency of the internal CPU clock.

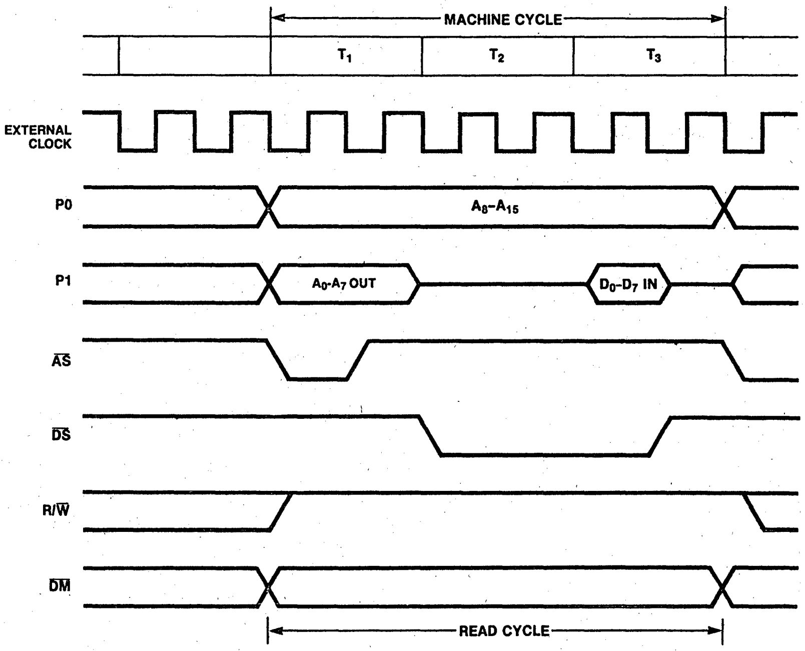


Figure 12-7. External Instruction Fetch or Memory Read Cycle

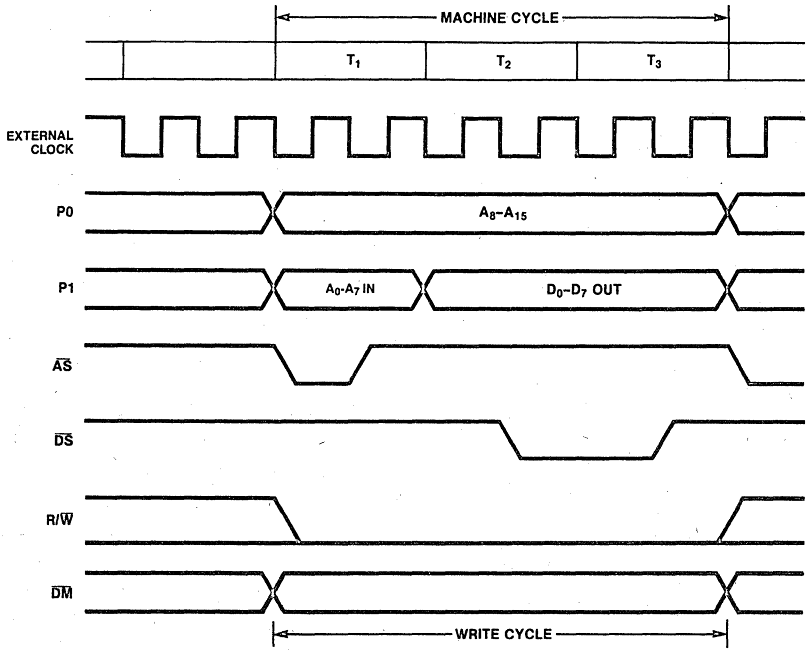


Figure 12-8. External Memory Write Cycle

12.6.1 Address Strobe (AS)

All transactions start with Address Strobe (AS) being driven low and then raised high by the Super8. The rising edge of AS indicates that Read/Write (R/W), Data Memory (DM), and the addresses output from Ports 0 and 1 are valid. The addresses output via Port 1 typically need to be latched during AS, whereas Port 0 address outputs, if used, remain stable throughout the machine cycle.

12.6.2 Data Strobe (DS)

The Super8 uses Data Strobe (DS) to time the actual data transfer. For write operations (R/W = low), a low on DS indicates that valid data is on the Port 1 AD_0-AD_7 lines. For read operations (R/W = high), the address/data bus is placed in a high-impedance state before driving DS low so that the addressed device can put its data on the bus. The Super8 samples this data prior to raising DS high.

12.6.3 External Memory Operations

Whenever the Super8 is configured for external memory operations, the addresses of all internal program memory references appear on the external bus. This should have no effect on the external system since the bus control line DS remains in its inactive high state. DS becomes active only during external memory references.

12.7 EXTENDED BUS TIMING

The Super8 can accommodate slow memory access and cycle times by three different methods that give the user much flexibility in the types of memory available.

12.7.1 Software Programmable Wait States

The Super8 can stretch the Data Strobe (DS) timing automatically by adding one, two, or three internal clock periods. This is under program control and applies only to external memory cycles. Internal memory cycles still operate at the maximum rate. The software has independent control over stretched Data Strobe for external memory (i.e., the software can set up one timing for program memory and a different timing for data memory). Thus, program and data memory may be made up of different kinds of hardware chips, each requiring its own timing.

12.7.2 Slow Memory Timing

Another feature of the Super8 that is useful in interfacing with slow memories is the Slow Memory Timing option. When this option is enabled, the normal external memory timing is slowed by a factor of two (bus clock = CPU clock divided by two). All memory times for set-up, duration, hold, and access times are essentially doubled. This feature can also be used with the programmed automatic wait states described above. Programmed wait states can still be used to stretch the Data Strobe time by one, two, or three internal clock times (not two, four, or six) when Slow Memory Timing is enabled.

12.7.3 Hardware Wait States

Still another Super8 feature is an optional external $\overline{WAIT}$ input using port pin $P3_4$. The $\overline{WAIT}$ input function can be used with either or both of the above two features. Thus the Data Strobe width will have a minimum value determined by the number of programmed wait states selected and/or by Slow Memory Timing. The $\overline{WAIT}$ input provides the means to stretch it even further. The $\overline{WAIT}$ input is sampled each internal clock time and, if held low, can stretch the Data Strobe by adding one internal clock period to the Data Strobe time for an indefinite period of time.

All of the extended bus timing features are programmed by writing the appropriate bits in the External Memory Timing register (Figure 12-9).

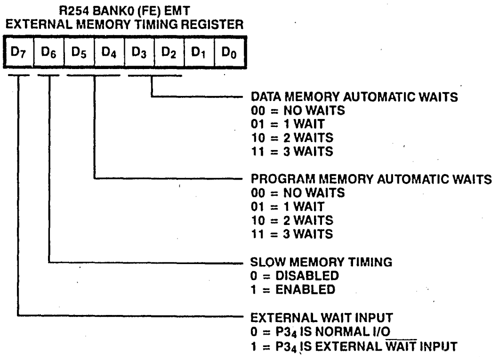


Figure 12-9. External Memory Timing Register

12.8 INSTRUCTION TIMING

The high throughput of the Super8 is due, in part, to the use of instruction pipelining, where the instruction fetch and execution cycles are overlapped. During the execution of the current instruction, the opcode of the next instruction is fetched, as illustrated in Figure 12-10.

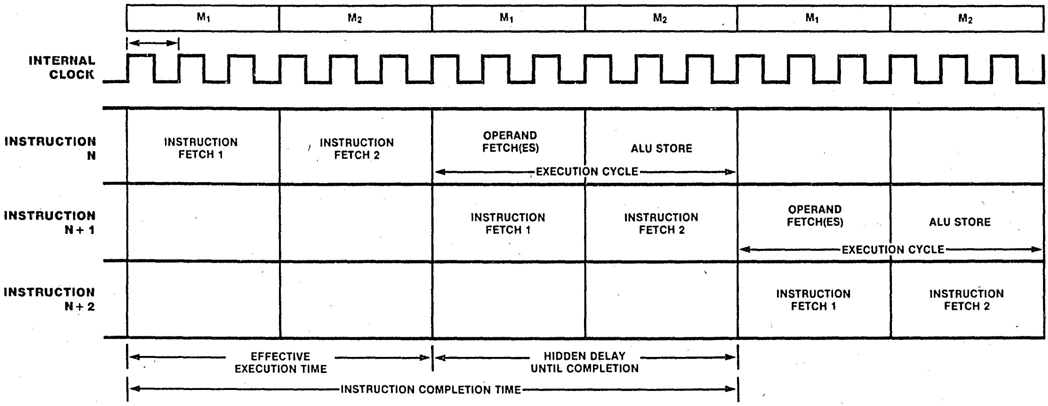


Figure 12-10. Instruction Pipelining

Figures 12-11 through 12-14 show typical instruction cycle timing for instructions fetched from external memory. All instruction fetch cycles have the same machine timing regardless of whether the memory is internal or external except when external memory timing is extended. In order to calculate the execution time of a program, the internal clock periods shown in the cycles column of the instruction formats in the Instruction Set (Chapter 5) should be added. Pipeline cycles are transparent to the user and should be ignored. Each cycle represents two cycles of the crystal or input clock.

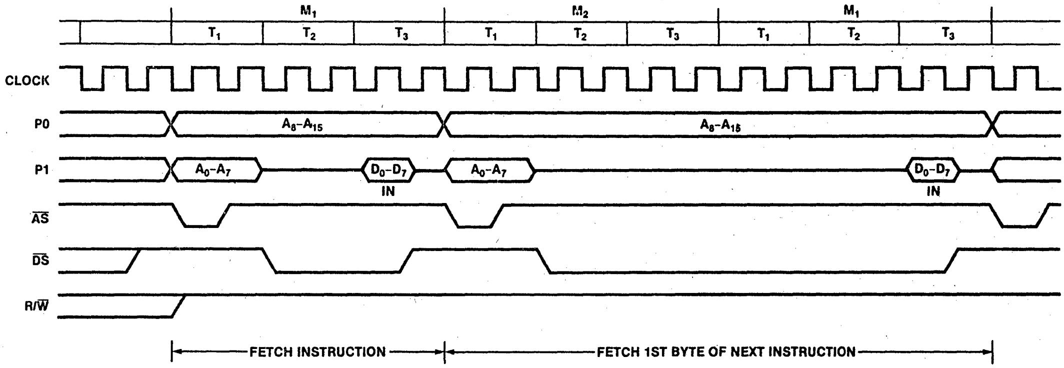


Figure 12-11. Typical Instruction Cycle Timing (One Byte Instruction)

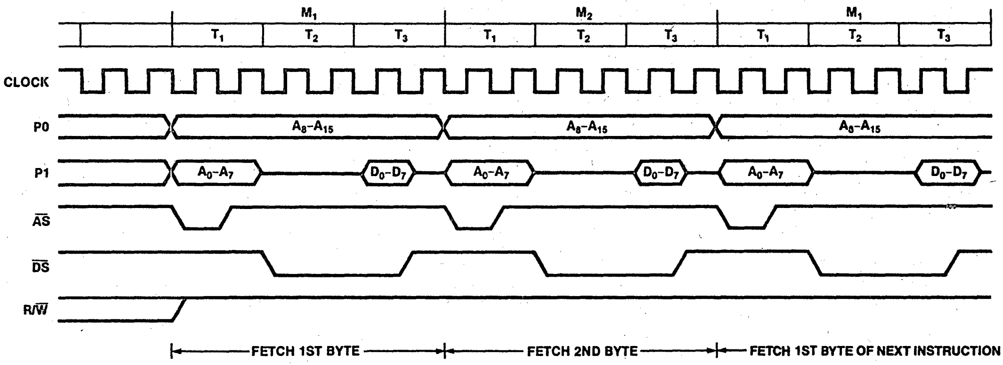


Figure 12-12. Typical Instruction Cycle Timing (Two Byte Instruction)

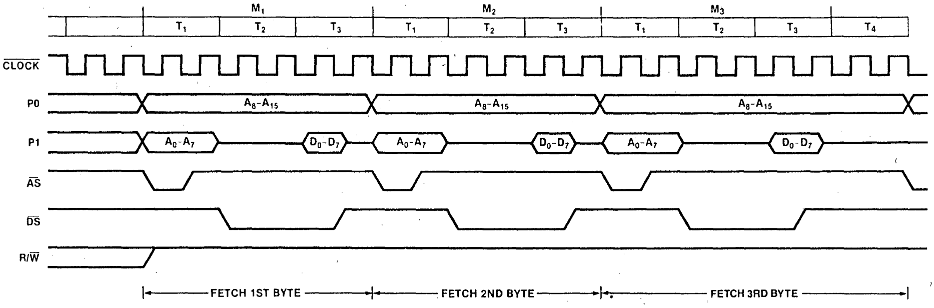


Figure 12-13. Typical Instruction Cycle Timing (Three Byte Instruction)

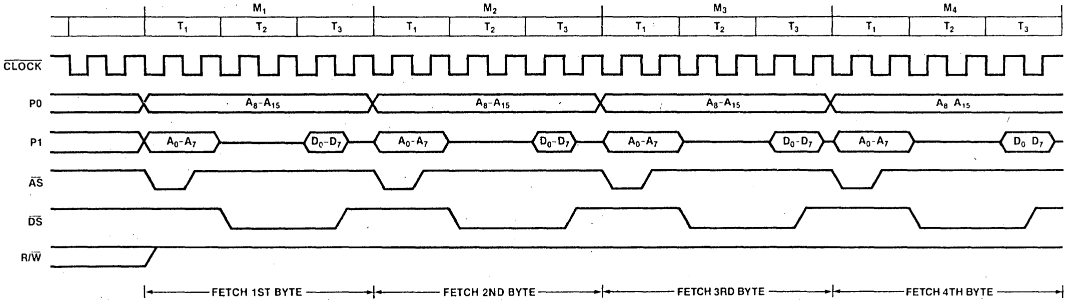


Figure 12-14. Typical Instruction Cycle Timing (Four Byte Instruction)

Glossary

addressing mode: The way in which the location of an operand is specified. There are seven addressing modes: Register, Indirect Register, Indexed, Direct Address, Indirect Address, Relative Address, and Immediate.

auto-echo mode: In this UART mode, the data coming in on the Receive Data pin is reflected out on the Transmit Data pin. The receive section still listens to the receive data input; however, the data from the transmit section goes nowhere.

base address: The address used, along with an index and/or displacement value, to calculate the effective address of an operand. The base address is located in a general-purpose register, the Program Counter, or the instruction.

baud-rate generator: The UART has its own on-chip programmable baud-rate generator that consists of two 8-bit Time Constant registers that hold the time constant value, a 16-bit Timer/Counter that counts down, and a flip-flop at the output producing a square wave.

bi-value mode: A Super8 counter/timer operating mode wherein the Time Constant and Capture registers alternate in loading the counter.

byte: A data item containing 8 contiguous bits. A byte is the basic data unit for addressing memory and peripherals.

capture: A "capture on external event" feature of the Super8 that takes a snapshot of the counter when a certain event occurs.

data memory: A memory address space that can hold only data to be read or written, not instruction code; data memory is always external to the Super8.

Deskew Counter: A 4-bit counter in each handshaking channel that is used to count processor clocks between the time that valid data is available at the port and the handshake signal indicates that data is available.

Direct Address (DA) addressing mode: In this mode, the effective address is contained in the instruction.

Direct Memory Access (DMA): An on-chip channel that provides high-speed transfers of data directly between memory and peripheral devices.

exception: A condition or event that alters the usual flow of instruction processing. The Super8 CPU supports two types of exception: reset and interrupts.

extended bus timing: The Super8 has the capability of stretching the Data Strobe timing by 1, 2, or 3 internal clock periods during external memory accesses. The software can set up one timing for program memory and a different timing for data memory.

fast interrupt processing: Fast interrupt processing completes the interrupt servicing in 6 clock periods instead of the usual 22.

Flag register: This register is used to supply the status of the Super8 CPU at any time.

Flag': A dedicated register that saves the contents of the Flag register when a fast interrupt occurs.

general-purpose registers: The 325 registers that can be used as accumulators, address pointers, index registers, data registers, or stack registers.

handshaking channels: The Super8 has two identical handshaking channels which operate in two modes--"fully interlocked" or two-wire mode, and "strobed" or single-wire mode.

Immediate (IM) addressing mode: In this mode, the operand is contained in the instruction.

Indexed (X) addressing mode: In this mode, the contents of an index register are added to the contents of a specified working register or working register pair, which holds the index value desired.

Indirect Address (IA) addressing mode: In this mode, the instruction specifies a pair of memory locations and this selected pair, in turn, contains the actual address of the instruction to be executed.

Indirect Register (IR) addressing mode: In this mode, the contents of the specified register or register pair is the address of the operand.

Instruction Pointer: A 16-bit register that acts as Program Counter for a threaded-code language, such as Forth, or can be used in the fast interrupt processing mode for special interrupt handling.

interrupt: An asynchronous exception generated by a peripheral device that needs attention. The interrupt structure of the Super8 contains 27 different interrupt sources, 16 vectors, and 8 levels.

interrupt level: Interrupt levels provide the top level of priority assignment and can be changed by programming the Interrupt Priority register.

Interrupt Priority register (IPR): This register assigns 192 different combinations of priority when more than one interrupt level is pending.

interrupt source: An interrupt source is anything that generates an interrupt, internal or external to the Super8.

interrupt vector: The vector number is used to generate the address of a particular interrupt servicing routine.

local loopback mode: In this mode, the data output from the transmit section of the UART is also routed back to the receive section.

pipelining: Instruction pipelining is a computer design technique in which the instruction fetch and execution cycles are overlapped. Thus, during the execution of the current instruction, the opcode of the next instruction is fetched, resulting in high throughput.

Program Counter (PC): The 16-bit Program Counter controls the sequence of instructions in the currently executing program and is not an addressable register.

program memory: A memory address space that can hold code or data; program memory can be internal or external to the Super8.

read access: The type of memory access used by the CPU for fetching data operands and instructions.

Register (R) addressing mode: In this mode, the operand value is the contents of the specified register or register pair.

register file: One of the three types of address spaces supported by the Super8 CPU. Register file address space is an internal register file composed of 325 8-bit wide registers that are logically divided into 32 working register groups of eight registers each.

Register Pointer (RP): The two register pointers are system registers that contain the base address of the two active working register groups of the register file.

Relative Address (RA) addressing mode: In this mode, the displacement in the instruction is added to the contents of the Program Counter to obtain the effective address.

reset: A CPU operating state or exception that results when a reset request is signaled on the $\overline{\text{RESET}}$ line. A reset initializes the Program Status registers.

Slow Memory timing: An optional feature of the Super8 in which normal external memory timing is slowed by a factor of two.

Stack Pointer (SP): A 16-bit register pair indicating the top (lowest address) of the processor stack and used by the Call instruction and interrupts to hold the return address.

system registers: System registers govern the operation of the CPU and may be accessed using any of the instructions that reference the register file using the Direct addressing mode.

Universal Asynchronous Receiver/Transmitter (UART): A full duplex asynchronous channel that transmits and receives independently with 5 to 8 bits per character, options for even or odd parity, and an optional wake-up feature.

wake-up feature: A feature of the UART wherein pattern match logic detects a pre-specified data pattern at the receiver; the pattern can include both the received character and a special wake-up bit.

write access: The type of memory access used by the CPU for storing data operands.

Zilog

PRELIMINARY INFORMATION

Product Specification

June 1987

Z8®Z8611 MCU
Military Electrical Specification

Z8603 Prototyping Device with 2K EPROM Interface

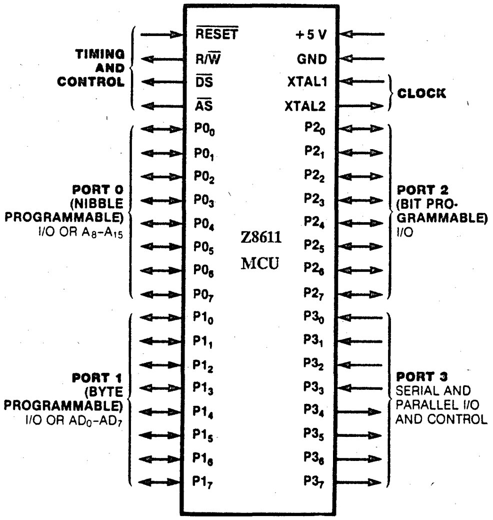
Features

- Complete microcomputer, 2K (8601) or 4K (8611) bytes of ROM, 128 bytes of RAM, 32 I/O lines, and up to 62K (8601) or 60K (8611) bytes addressable external space each for program and data memory.
- 144-byte register file, including 124 general-purpose registers, four I/O port registers, and 16 status and control registers.
- Average instruction execution time of 1.5 μs, maximum of 1 μs.
- Vectored, priority interrupts for I/O, counter/timers, and UART.
- Full-duplex UART and two programmable 8-bit counter/timers, each with a 6-bit programmable prescaler.
- Register Pointer so that short, fast instructions can access any of nine working register groups in 1 μs.
- On-chip oscillator which accepts crystal or external clock drive.
- Single +5 V power supply—all pins TTL compatible.
- 12.5 MHz.

General Description

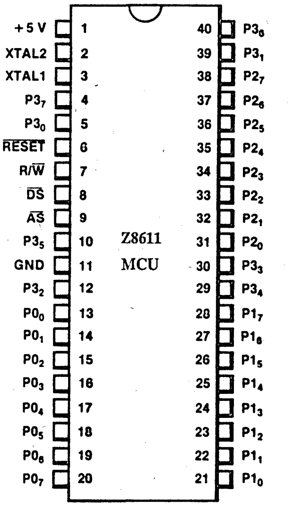
The Z8 microcomputer introduces a new level of sophistication to single-chip architecture. Compared to earlier single-chip microcomputers, the Z8 offers faster execution; more efficient use of memory; more sophisticated interrupt, input/output and bit-manipulation capabilities; and easier system expansion.

Under program control, the Z8 can be tailored to the needs of its user. It can be configured as a stand-alone microcomputer with 4K bytes of internal ROM, a traditional microprocessor that manages up to 124K bytes of external memory, or a parallel-processing element in a system with other processors and peripheral controllers linked by the Z-BUS® bus. In all configurations, a large number of pins remain available for I/O.



TIMING AND CONTROL
RESET
R/W
DS
AS
+5 V
GND
XTAL1
XTAL2
CLOCK
PORT 0 (NIBBLE PROGRAMMABLE) I/O OR A8-A15
P0₀ P0₁ P0₂ P0₃ P0₄ P0₅ P0₆ P0₇
Z8611 MCU
P2₀ P2₁ P2₂ P2₃ P2₄ P2₅ P2₆ P2₇
PORT 2 (BIT PROGRAMMABLE) I/O
PORT 1 (BYTE PROGRAMMABLE) I/O OR AD0-AD7
P1₀ P1₁ P1₂ P1₃ P1₄ P1₅ P1₆ P1₇
P3₀ P3₁ P3₂ P3₃ P3₄ P3₅ P3₆ P3₇
PORT 3 SERIAL AND PARALLEL I/O AND CONTROL

Figure 1. Pin Functions



Figuro 2a. 40-pin Dual-In-Lino Packago (DIP), Pin Assignments

Pin Description

$\overline{\text{AS}}$. *Address Strobe* (output, active Low). Address Strobe is pulsed once at the beginning of each machine cycle. Addresses output via Port 1 for all external program or data memory transfers are valid at the trailing edge of $\overline{\text{AS}}$. Under program control, $\overline{\text{AS}}$ can be placed in the high-impedance state along with Ports 0 and 1, Data Strobe and Read/Write.

$\overline{\text{DS}}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each external memory transfer.

$P0_0$-$P0_7$, $P1_0$-$P1_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. *I/O Port Lines* (input/outputs, TTL-compatible). These 32 lines are divided into four 8-bit I/O ports that can be configured under program control for I/O or external memory interface.

$\overline{\text{RESET}}$. *Reset* (input, active Low). $\overline{\text{RESET}}$ initializes the Z8. When $\overline{\text{RESET}}$ is deactivated, program execution begins from internal program location $000C_H$.

$\overline{\text{ROMless}}$. (input, active LOW). This pin is only available on the 44 pin versions of the Z8611. When connected to GND disables the internal ROM and forces the part to function as a Z8681 ROMless Z8. When left unconnected or pulled high to V_{CC} the part will function normally as a Z8611.

R/$\overline{\text{W}}$. *Read/Write* (output). R/$\overline{\text{W}}$ is Low when the Z8 is writing to external program or data memory.

XTAL1, XTAL2. *Crystal 1, Crystal 2* (time-base input and output). These pins connect a parallel resonant 12.5 MHz crystal or an external single-phase 12.5 MHz clock to the on-chip clock oscillator and buffer.

Architecture

Z8 architecture is characterized by a flexible I/O scheme, an efficient register and address space structure and a number of ancillary features that are helpful in many applications.

Microcomputer applications demand powerful I/O capabilities. The Z8 fulfills this with 32 pins dedicated to input and output. These lines are grouped into four ports of eight lines each and are configurable under software control to provide timing, status signals, serial or parallel I/O with or without handshake, and an address/ data bus for interfacing external memory.

Because the multiplexed address/data bus is merged with the I/O-oriented ports, the Z8 can assume many different memory and I/O configurations. These configurations range from a self-contained microcomputer to a microprocessor that can address 124K (Z8601) or 120K (Z8611) bytes of external memory.

Three basic address spaces are available to support this wide range of configurations: program memory (internal and external), data memory (external) and the register file (internal). The 144-byte random-access register file is composed of 124 general-purpose registers, four I/O port registers, and 16 control and status registers.

To unburden the program from coping with real-time problems such as serial data communication and counting/timing, an asynchronous receiver/transmitter (UART) and two counter/timers with a large number of userselectable modes are offered on-chip. Hardware support for the UART is minimized because one of the on-chip timers supplies the bit rate.

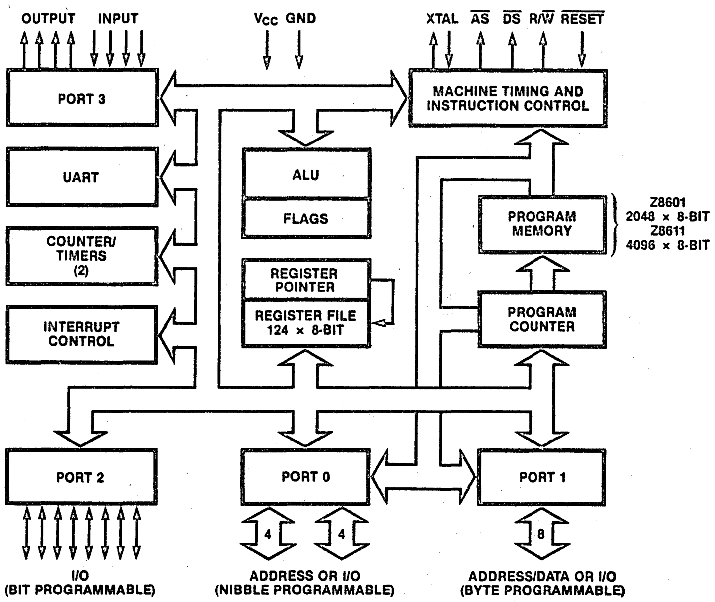


Figure 3. Functional Block Diagram

Address Spaces

Program Memory. The 16-bit program counter addresses 64K bytes of program memory space. Program memory can be located in two areas: one internal and the other external (Figure 4). The first 4096 (Z8611) bytes consist of on-chip mask-programmed ROM. At addresses 4096 (Z8611) and greater, the Z8 executes external program memory fetches.

The first 12 bytes of program memory are reserved for the interrupt vectors. These locations contain six 16-bit vectors that correspond to the six available interrupts.

Data Memory. The Z8 can address 60K (Z8611) bytes of external data memory beginning at location 4096 (Z8611) (Figure 5). External data memory may be included with or separated from the external program memory space. $\overline{DM}$, an optional I/O function that can be programmed to appear on pin $P3_4$, is used to distinguish between data and program memory space.

Register File. The 144-byte register file includes four I/O port registers (R0–R3), 124 general-purpose registers (R4–R127) and 16 control and status registers (R240–R255). These registers are assigned the address locations shown in Figure 6.

Z8 instructions can access registers directly or indirectly with an 8-bit address field. The Z8 also allows short 4-bit register addressing using the Register Pointer (one of the control registers). In the 4-bit mode, the register file is

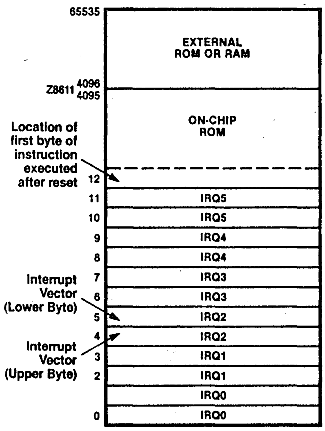


Figure 4. Program Memory Map

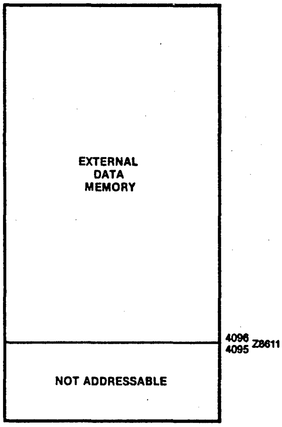
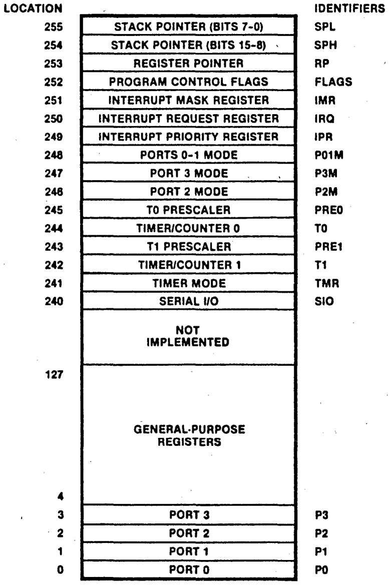


Figure 5. Data Memory Map



LOCATION		IDENTIFIERS
255	STACK POINTER (BITS 7-0)	SPL
254	STACK POINTER (BITS 15-8)	SPH
253	REGISTER POINTER	RP
252	PROGRAM CONTROL FLAGS	FLAGS
251	INTERRUPT MASK REGISTER	IMR
250	INTERRUPT REQUEST REGISTER	IRQ
249	INTERRUPT PRIORITY REGISTER	IPR
248	PORTS 0-1 MODE	P01M
247	PORT 3 MODE	P3M
246	PORT 2 MODE	P2M
245	T0 PRESCALER	PRE0
244	TIMER/COUNTER 0	T0
243	T1 PRESCALER	PRE1
242	TIMER/COUNTER 1	T1
241	TIMER MODE	TMR
240	SERIAL I/O	SIO
	NOT IMPLEMENTED	
127	GENERAL-PURPOSE REGISTERS	
4		
3	PORT 3	P3
2	PORT 2	P2
1	PORT 1	P1
0	PORT 0	P0

Figure 6. The Register File

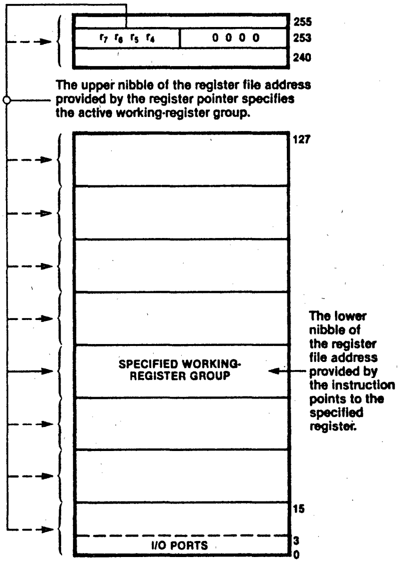

255
253
240
r7 r6 r5 r4
0 0 0 0
The upper nibble of the register file address provided by the register pointer specifies the active working-register group.
127
SPECIFIED WORKING-REGISTER GROUP
The lower nibble of the register file address provided by the instruction points to the specified register.
15
I/O PORTS
3
0

Figure 7. The Register Pointer

divided into nine working-register groups, each occupying 16 continguous locations (Figure 6). The Register Pointer addresses the starting location of the active working-register group (Figure 7).

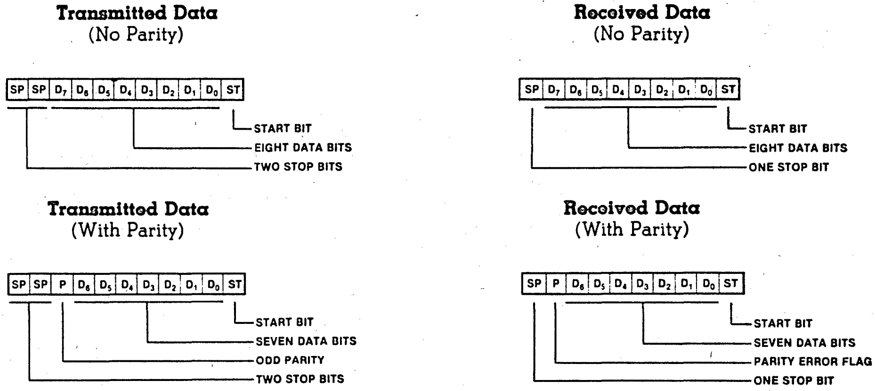
Stacks. Either the internal register file or the external data memory can be used for the stack. A 16-bit Stack Pointer (R254 and R255) is used for the external stack, which can reside anywhere in data memory between locations 2048 (8601) or 4096 (8611) and 65535. An 8-bit Stack Pointer (R255) is used for the internal stack that resides within the 124 general-purpose registers (R4–R127).

Serial Input/ Output

Port 3 lines $P3_0$ and $P3_7$ can be programmed as serial I/O lines for full-duplex serial asynchronous receiver/transmitter operation. The bit rate is controlled by Counter/Timer 0, at 12 MHz.

The Z8 automatically adds a start bit and two stop bits to transmitted data (Figure 8). Odd parity is also available as an option. Eight data bits are always transmitted, regardless of parity selection. If parity is enabled, the eighth bit is the odd parity bit. An interrupt request (IRQ_4) is generated on all transmitted characters.

Received data must have a start bit, eight data bits and at least one stop bit. If parity is on, bit 7 of the received data is replaced by a parity error flag. Received characters generate the IRQ_3 interrupt request.


Transmitted Data
(No Parity)

Received Data
(No Parity)

Transmitted Data
(With Parity)

Received Data
(With Parity)

Figure 8. Serial Data Formats

Counter/ Timers

The Z8 contains two 8-bit programmable counter/timers (T_0 and T_1), each driven by its own 6-bit programmable prescaler. The T_1 prescaler can be driven by internal or external clock sources; however, the T_0 prescaler is driven by the internal clock only.

The 6-bit prescalers can divide the input frequency of the clock source by any number from 1 to 64. Each prescaler drives its counter, which decrements the value (1 to 256) that has been loaded into the counter. When the counter reaches the end of count, a timer interrupt request—IRQ_4 (t_0) or IRQ_5 (T_1)—is generated.

The counters can be started, stopped, restarted to continue, or restarted from the initial value. The counters can also be programmed to stop upon reaching zero (single-pass mode) or to automatically reload the initial value and continue counting (modulo-n continuous mode). The counters, but not the prescalers, can be read any time without disturbing their value or count mode.

The clock source for T_1 is user-definable and can be the internal microprocessor clock divided by four, or an external signal input via Port 3. The Timer Mode register configures the external timer input as an external clock, a trigger input that can be retriggerable or non-retriggerable, or as a gate input for the internal clock. The counter/timers can be programmably cascaded by connecting the T_0 output to the input of T_1. Port 3 line $P3_6$ also serves as a timer output (T_{OUT}) through which T_0, T_1 or the internal clock can be output.

I/O Ports

The Z8 has 32 lines dedicated to input and output. These lines are grouped into four ports of eight lines each and are configurable as input, output or address/data. Under software control, the ports can be programmed to provide address outputs, timing, status signals, serial I/O, and parallel I/O with or without handshake. All ports have active pull-ups and pull-downs compatible with TTL loads.

Port 1 can be programmed as a byte I/O port or as an address/data port for interfacing external memory. When used as an I/O port, Port 1 may be placed under handshake control. In this configuration, Port 3 lines $P3_3$ and $P3_4$ are used as the handshake controls RDY_1 and $\overline{DAV}_1$ (Ready and Data Available).

Memory locations greater than 2048 (Z8601) or 4096 (Z8611) are referenced through Port 1. To interface external memory, Port 1 must be programmed for the multiplexed Address/Data mode. If more than 256 external locations are required, Port 0 must output the additional lines.

Port 1 can be placed in the high-impedance state along with Port 0, $\overline{AS}$, $\overline{DS}$ and $R/\overline{W}$, allowing the Z8 to share common resources in multiprocessor and DMA applications. Data transfers can be controlled by assigning $P3_3$ as a Bus Acknowledge input and $P3_4$ as a Bus Request output.

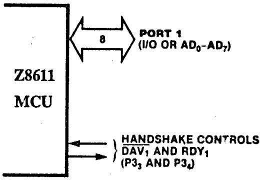


Figure 9a. Port 1

Port 0 can be programmed as a nibble I/O port, or as an address port for interfacing external memory. When used as an I/O port, Port 0 may be placed under handshake control. In this configuration, Port 3 lines $P3_2$ and $P3_5$ are used as the handshake controls $\overline{DAV}_0$ and RDY_0. Handshake signal assignment is dictated by the I/O direction of the upper nibble $P0_4$-$P0_7$.

For external memory references, Port 0 can provide address bits A_8-A_{11} (lower nibble) or A_8-A_{15} (lower and upper nibble) depending on the required address space. If the address range requires 12 bits or less, the upper nibble of Port 0 can be programmed independently as I/O while the lower nibble is used for addressing. When Port 0 nibbles are defined as address bits, they can be set to the highimpedance state along with Port 1 and the control signals $\overline{AS}$, $\overline{DS}$ and $R/\overline{W}$.

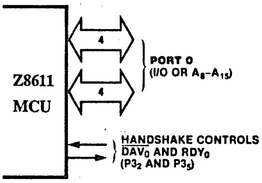


Figure 9b. Port 0

Port 2 bits can be programmed independently as input or output. The port is always available for I/O operations. In addition, Port 2 can be configured to provide open-drain outputs.

Like Ports 0 and 1, Port 2 may also be placed under handshake control. In this configuration, Port 3 lines $P3_1$ and $P3_6$ are used as the handshake controls lines $\overline{DAV}_2$ and RDY_2. The handshake signal assignment for Port 3 lines $P3_1$ and $P3_6$ is dictated by the direction (input or output) assigned to bit 7 of Port 2.

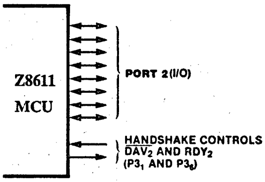


Figure 9c. Port 2

Port 3 lines can be configured as I/O or control lines. In either case, the direction of the eight lines is fixed as four input ($P3_0$-$P3_3$) and four output ($P3_4$-$P3_7$). For serial I/O, lines $P3_0$ and $P3_7$ are programmed as serial in and serial out respectively.

Port 3 can also provide the following control functions: handshake for Ports 0, 1 and 2 ($\overline{DAV}$ and RDY); four external interrupt request signals (IRQ_0-IRQ_3); timer input and output signals (T_{IN} and T_{OUT}) and Data Memory Select ($\overline{DM}$).

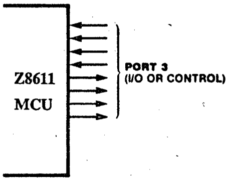


Figure 9d. Port 3

Interrupts

The Z8 allows six different interrupts from eight sources: the four Port 3 lines $P3_0$–$P3_3$, Serial In, Serial Out, and the two counter/timers. These interrupts are both maskable and prioritized. The Interrupt Mask register globally or individually enables or disables the six interrupt requests. When more than one interrupt is pending, priorities are resolved by a programmable priority encoder that is controlled by the Interrupt Priority register.

All Z8 interrupts are vectored. When an interrupt request is granted, an interrupt machine cycle is entered. This disables all subsequent interrupts, saves the Program Counter and status flags, and branches to the program memory vector location reserved for that interrupt. This memory location and the next byte contain the 16-bit address of the interrupt service routine for that particular interrupt request.

Polled interrupt systems are also supported. To accommodate a polled structure, any or all of the interrupt inputs can be masked and the Interrupt Request register polled to determine which of the interrupt requests needs service.

Clock

The on-chip oscillator has a high-gain, parallel-resonant amplifier for connection to a crystal or to any suitable external clock source (XTAL1 = Input, XTAL2 = Output).

The crystal source is connected across XTAL1 and XTAL2, using the recommended capacitors ($C_l \leq 15$ pF) from each pin to ground. The specifications for the crystal are as follows:

- AT cut, parallel resonant
- Fundamental type, 12.5 MHz maximum
- Series resistance, $R_s \leq 100\,\Omega$

Instruction Set Notation

Addressing Modes. The following notation is used to describe the addressing modes and instruction operations as shown in the instruction summary.

IRR	Indirect register pair or indirect working-register pair address
Irr	Indirect working-register pair only
X	Indexed address
DA	Direct address
RA	Relative address
IM	Immediate
R	Register or working-register address
r	Working-register address only
IR	Indirect-register or indirect working-register address
Ir	Indirect working-register address only
RR	Register pair or working register pair address

Symbols. The following symbols are used in describing the instruction set.

dst	Destination location or contents
src	Source location or contents
cc	Condition code (see list)
@	Indirect address prefix
SP	Stack pointer (control registers 254-255)
PC	Program counter
FLAGS	Flag register (control register 252)
RP	Register pointer (control register 253)
IMR	Interrupt mask register (control register 251)

Assignment of a value is indicated by the symbol "←". For example,

dst ← dst + src

indicates that the source data is added to the destination data and the result is stored in the destination location. The notation "addr(n)" is used to refer to bit "n" of a given location. For example,

dst (7)

refers to bit 7 of the destination operand.

Flags. Control Register R252 contains the following six flags:

C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag

Affected flags are indicated by:

0	Cleared to zero
1	Set to one
*****	Set or cleared according to operation
–	Unaffected
X	Undefined

Condition Codes

Value	Mnemonic	Meaning	Flags Set
1000		Always true	---
0111	C	Carry	C = 1
1111	NC	No carry	C = 0
0110	Z	Zero	Z = 1
1110	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110	EQ	Equal	Z = 1
1110	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	[Z OR (S XOR V)] = 0
0010	LE	Less than or equal	[Z OR (S XOR V)] = 1
1111	UGE	Unsigned greater than or equal	C = 0
0111	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1
0000		Never true	---

Instruction Formats

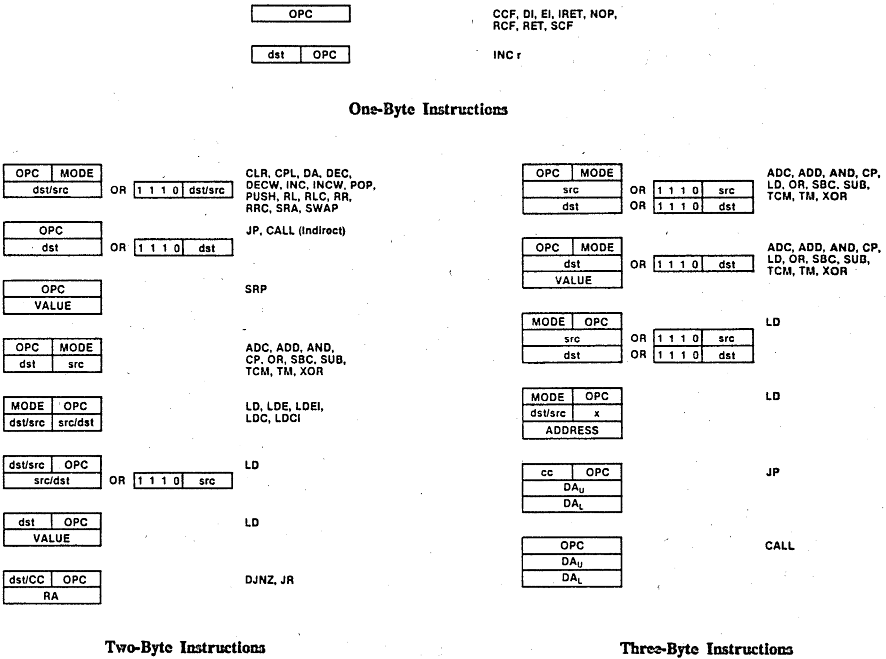
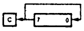
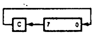
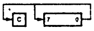
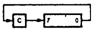
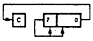
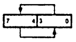


Figure 12. Instruction Formats

Instruction Summary

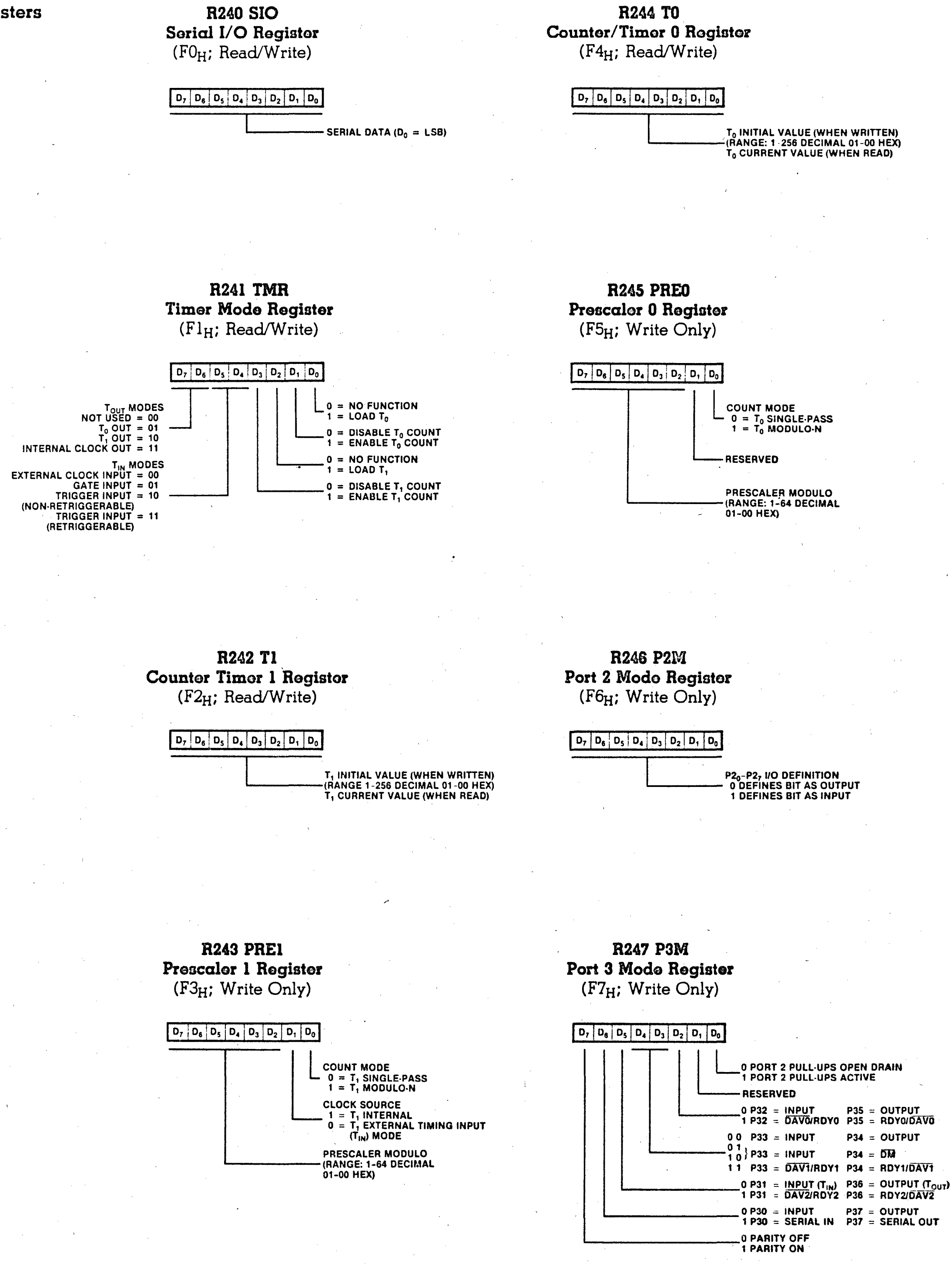
Instruction and Operation	Addr Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
ADC dst,src	(Note 1)		1□	*	*	*	*	0	*
dst ← dst + src + C									
ADD dst,src	(Note 1)		0□	*	*	*	*	0	*
dst ← dst + src									
AND dst,src	(Note 1)		5□	–	*	*	0	–	–
dst ← dst AND src									
CALL dst	DA		D6	–	–	–	–	–	–
SP ← SP - 2	IRR		D4						
@SP ← PC; PC ← dst									
CCF			EF	*	–	–	–	–	–
C ← NOT C									
CLR dst	R		B0	–	–	–	–	–	–
dst ← 0	IR		B1						
COM dst	R		60	–	*	*	0	–	–
dst ← NOT dst	IR		61						
CP dst,src	(Note 1)		A□	*	*	*	*	–	–
dst - src									
DA dst	R		40	*	*	*	X	–	–
dst ← DA dst	IR		41						
DEC dst	R		00	–	*	*	*	–	–
dst ← dst - 1	IR		01						
DECW dst	RR		80	–	*	*	*	–	–
dst ← dst - 1	IR		81						
DI									
IMR (7) ← 0			8F	–	–	–	–	–	–
DJNZ r,dst	RA		rA	–	–	–	–	–	–
r ← r - 1			r=0-F						
if r ≠ 0									
PC ← PC + dst									
Range: +127, -128									
EI			9F	–	–	–	–	–	–
IMR (7) ← 1									
INC dst	r		rE	–	*	*	*	–	–
dst ← dst + 1			r=0-F						
	R		20						
	IR		21						
INCW dst	RR		A0	–	*	*	*	–	–
dst ← dst + 1	IR		A1						
IRET			BF	*	*	*	*	*	*
FLAGS ← @SP; SP ← SP + 1									
PC ← @SP; SP ← SP + 2; IMR (7) ← 1									
JP cc,dst	DA		cD	–	–	–	–	–	–
if cc is true			c=0-F						
PC ← dst	IRR		30						
JR cc,dst	RA		cB	–	–	–	–	–	–
if cc is true,			c=0-F						
PC ← PC + dst									
Range: +127, -128									
LD dst,src	r	Im	rC	–	–	–	–	–	–
dst ← src	r	R	r8						
	R	r	r9						
			r=0-F						
	r	X	C7						
	X	r	D7						
	r	Ir	E3						
	Ir	r	F3						
	R	R	E4						
	R	IR	E5						
	R	Im	E6						
	IR	Im	E7						
	IR	R	F5						
LDC dst,src	r	Irr	C2	–	–	–	–	–	–
dst ← src	Irr	r	D2						
LDCI dst,src	Ir	Irr	C3	–	–	–	–	–	–
dst ← src	Irr	Ir	D3						
r ← r + 1; rr ← rr + 1									
LDE dst,src	r	Irr	82	–	–	–	–	–	–
dst ← src	Irr	r	92						
LDEI dst,src	Ir	Irr	83	–	–	–	–	–	–
dst ← src	Irr	Ir	93						
r ← r + 1; rr ← rr + 1									
NOP			FF	–	–	–	–	–	–
OR dst,src	(Note 1)		4□	–	*	*	0	–	–
dst ← dst OR src									
POP dst	R		50	–	–	–	–	–	–
dst ← @SP	IR		51						
SP ← SP + 1									
PUSH src		R	70	–	–	–	–	–	–
SP ← SP - 1; @SP ← src		IR	71						
RCF			CF	0	–	–	–	–	–
C ← 0									
RET			AF	–	–	–	–	–	–
PC ← @SP; SP ← SP + 2									
RL dst 	R		90	*	*	*	*	–	–
	IR		91						
RLC dst 	R		10	*	*	*	*	–	–
	IR		11						
RR dst 	R		E0	*	*	*	*	–	–
	IR		E1						
RRC dst 	R		C0	*	*	*	*	–	–
	IR		C1						
SBC dst,src	(Note 1)		3□	*	*	*	*	1	*
dst ← dst - src - C									
SCF			DF	1	–	–	–	–	–
C ← 1									
SRA dst 	R		D0	*	*	*	0	–	–
	IR		D1						
SRP src		Im	31	–	–	–	–	–	–
RP ← src									
SUB dst,src	(Note 1)		2□	*	*	*	*	1	*
dst ← dst - src									
SWAP dst 	R		F0	X	*	*	X	–	–
	IR		F1						
TCM dst,src	(Note 1)		6□	–	*	*	0	–	–
(NOT dst) AND src									
TM dst, src	(Note 1)		7□	–	*	*	0	–	–
dst AND src									
XOR dst,src	(Note 1)		B□	–	*	*	0	–	–
dst ← dst XOR src									

Note 1

These instructions have an identical set of addressing modes, which are encoded for brevity. The first opcode nibble is found in the instruction set table above. The second nibble is expressed symbolically by a □ in this table, and its value is found in the following table to the right of the applicable addressing mode pair.

For example, to determine the opcode of a ADC instruction use the addressing modes r (destination) and Ir (source). The result is 13.

Addr Mode		Lower Opcode Nibble
dst	src	
r	r	$\boxed{2}$
r	Ir	$\boxed{3}$
R	R	$\boxed{4}$
R	IR	$\boxed{5}$
R	IM	$\boxed{6}$
IR	IM	$\boxed{7}$



R240 SIO
Serial I/O Register
($F0_H$; Read/Write)

R244 T0
Counter/Timer 0 Register
($F4_H$; Read/Write)

D7 D6 D5 D4 D3 D2 D1 D0
T_0 INITIAL VALUE (WHEN WRITTEN)
(RANGE: 1-256 DECIMAL 01-00 HEX)
T_0 CURRENT VALUE (WHEN READ)

R241 TMR
Timer Mode Register
($F1_H$; Read/Write)

R245 PRE0
Prescaler 0 Register
($F5_H$; Write Only)

R242 T1
Counter Timer 1 Register
($F2_H$; Read/Write)

D7 D6 D5 D4 D3 D2 D1 D0
T_1 INITIAL VALUE (WHEN WRITTEN)
(RANGE 1-256 DECIMAL 01-00 HEX)
T_1 CURRENT VALUE (WHEN READ)

R246 P2M
Port 2 Mode Register
($F6_H$; Write Only)

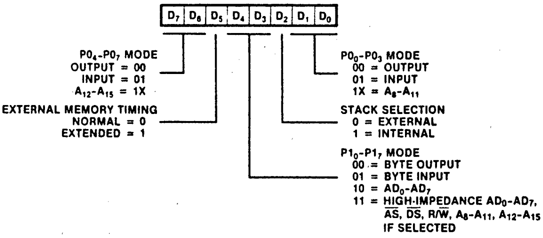
R243 PRE1
Prescaler 1 Register
($F3_H$; Write Only)

R247 P3M
Port 3 Mode Register
($F7_H$; Write Only)

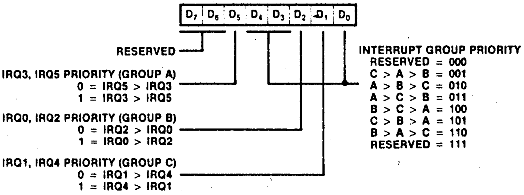
D7 D6 D5 D4 D3 D2 D1 D0
0 PORT 2 PULL-UPS OPEN DRAIN
1 PORT 2 PULL-UPS ACTIVE
RESERVED
0 P32 = INPUT P35 = OUTPUT
1 P32 = $\overline{DAV0}$/RDY0 P35 = RDY0/$\overline{DAV0}$
0 0 P33 = INPUT P34 = OUTPUT
0 1 / 1 0 } P33 = INPUT P34 = $\overline{DM}$
1 1 P33 = $\overline{DAV1}$/RDY1 P34 = RDY1/$\overline{DAV1}$
0 P31 = INPUT (T_{IN}) P36 = OUTPUT (T_{OUT})
1 P31 = $\overline{DAV2}$/RDY2 P36 = RDY2/$\overline{DAV2}$
0 P30 = INPUT P37 = OUTPUT
1 P30 = SERIAL IN P37 = SERIAL OUT
0 PARITY OFF
1 PARITY ON

Figure 13. Control Registers

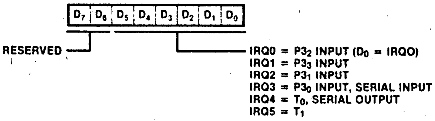
R248 P01M
Port 0 and 1 Mode Register
($F8_H$; Write Only)



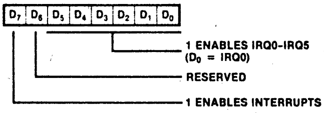
R249 IPR
Interrupt Priority Register
($F9_H$; Write Only)



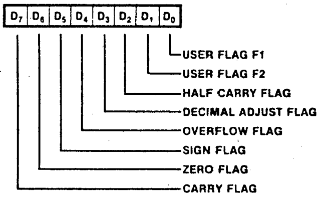
R250 IRQ
Interrupt Request Register
(FA_H; Read/Write)



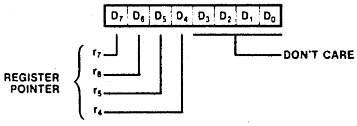
R251 IMR
Interrupt Mask Register
(FB_H; Read/Write)



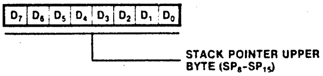
R252 FLAGS
Flag Register
(FC_H; Read/Write)



R253 RP
Register Pointer
(FD_H; Read/Write)



R254 SPH
Stack Pointer
(FE_H; Read/Write)



R255 SPL
Stack Pointer
(FF_H; Read/Write)

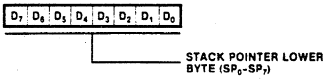
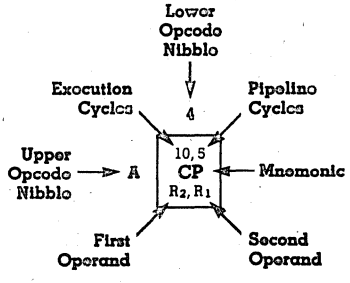


Figure 13. Control Registers (Continued)

Opcode Map

Lower Nibble (Hex)

Upper Nibble (Hex)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6,5 **DEC** R_1	6,5 **DEC** IR_1	6,5 **ADD** r_1, r_2	6,5 **ADD** r_1, Ir_2	10,5 **ADD** R_2, R_1	10,5 **ADD** IR_2, R_1	10,5 **ADD** R_1, IM	10,5 **ADD** IR_1, IM	6,5 **LD** r_1, R_2	6,5 **LD** r_2, R_1	12/10,5 **DJNZ** r_1, RA	12/10,0 **JR** cc, RA	6,5 **LD** r_1, IM	12/10,0 **JP** cc, DA	6,5 **INC** r_1	
1	6,5 **RLC** R_1	6,5 **RLC** IR_1	6,5 **ADC** r_1, r_2	6,5 **ADC** r_1, Ir_2	10,5 **ADC** R_2, R_1	10,5 **ADC** IR_2, R_1	10,5 **ADC** R_1, IM	10,5 **ADC** IR_1, IM								
2	6,5 **INC** R_1	6,5 **INC** IR_1	6,5 **SUB** r_1, r_2	6,5 **SUB** r_1, Ir_2	10,5 **SUB** R_2, R_1	10,5 **SUB** IR_2, R_1	10,5 **SUB** R_1, IM	10,5 **SUB** IR_1, IM								
3	8,0 **JP** IRR_1	6,1 **SRP** IM	6,5 **SBC** r_1, r_2	6,5 **SBC** r_1, Ir_2	10,5 **SBC** R_2, R_1	10,5 **SBC** IR_2, R_1	10,5 **SBC** R_1, IM	10,5 **SBC** IR_1, IM								
4	8,5 **DA** R_1	8,5 **DA** IR_1	6,5 **OR** r_1, r_2	6,5 **OR** r_1, Ir_2	10,5 **OR** R_2, R_1	10,5 **OR** IR_2, R_1	10,5 **OR** R_1, IM	10,5 **OR** IR_1, IM								
5	10,5 **POP** R_1	10,5 **POP** IR_1	6,5 **AND** r_1, r_2	6,5 **AND** r_1, Ir_2	10,5 **AND** R_2, R_1	10,5 **AND** IR_2, R_1	10,5 **AND** R_1, IM	10,5 **AND** IR_1, IM								
6	6,5 **COM** R_1	6,5 **COM** IR_1	6,5 **TCM** r_1, r_2	6,5 **TCM** r_1, Ir_2	10,5 **TCM** R_2, R_1	10,5 **TCM** IR_2, R_1	10,5 **TCM** R_1, IM	10,5 **TCM** IR_1, IM								
7	10/12,1 **PUSH** R_2	12/14,1 **PUSH** IR_2	6,5 **TM** r_1, r_2	6,5 **TM** r_1, Ir_2	10,5 **TM** R_2, R_1	10,5 **TM** IR_2, R_1	10,5 **TM** R_1, IM	10,5 **TM** IR_1, IM								
8	10,5 **DECW** RR_1	10,5 **DECW** IR_1	12,0 **LDE** r_1, Irr_2	18,0 **LDEI** Ir_1, Irr_2												6,1 **DI**
9	6,5 **RL** R_1	6,5 **RL** IR_1	12,0 **LDE** Irr_1	18,0 **LDEI** Ir_2, Irr_1												6,1 **EI**
A	10,5 **INCW** RR_1	10,5 **INCW** IR_1	6,5 **CP** r_1, r_2	6,5 **CP** r_1, Ir_2	10,5 **CP** R_2, R_1	10,5 **CP** IR_2, R_1	10,5 **CP** R_1, IM	10,5 **CP** IR_1, IM								14,0 **RET**
B	6,5 **CLR** R_1	6,5 **CLR** IR_1	6,5 **XOR** r_1, r_2	6,5 **XOR** r_1, Ir_2	10,5 **XOR** R_2, R_1	10,5 **XOR** IR_2, R_1	10,5 **XOR** R_1, IM	10,5 **XOR** IR_1, IM								16,0 **IRET**
C	6,5 **RRC** R_1	6,5 **RRC** IR_1	12,0 **LDC** r_1, Irr_2	18,0 **LDCI** Ir_1, Irr_2				10,5 **LD** r_1, x, R_2								6,5 **RCF**
D	6,5 **SRA** R_1	6,5 **SRA** IR_1	12,0 **LDC** r_2, Irr_1	18,0 **LDCI** Ir_2, Irr_1	20,0 **CALL*** IRR_1		20,0 **CALL** DA	10,5 **LD** r_2, x, R_1								6,5 **SCF**
E	6,5 **RR** R_1	6,5 **RR** IR_1		6,5 **LD** r_1, Ir_2	10,5 **LD** R_2, R_1	10,5 **LD** IR_2, R_1	10,5 **LD** R_1, IM	10,5 **LD** IR_1, IM								6,5 **CCF**
F	8,5 **SWAP** R_1	8,5 **SWAP** IR_1		6,5 **LD** Ir_1, r_2		10,5 **LD** R_2, IR_1										6,0 **NOP**
Bytes per Instruction	2				3				2					3	1	



Legend:
R = 8-Bit Address
r = 4-Bit Address
R_1 or r_1 = Dst Address
R_2 or r_2 = Src Address

Sequence:
Opcode, First Operand, Second Operand

Note: The blank areas are not defined.

*2-byte instruction; fetch cycle appears as a 3-byte instruction

Absolute Maximum Ratings

Voltages on all pins
with respect to GND..........-0.3 V to +7.0 V
Operating Ambient
Temperature........See Ordering Information
Storage Temperature........-65°C to +150°C

Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Standard Test Conditions

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND. Positive current flows into the reference pin.

Standard conditions are:

□ $+4.75\ V \le V_{CC} \le +5.25\ V$
□ GND = 0 V
□ $0°C \le T_A \le +70°C$

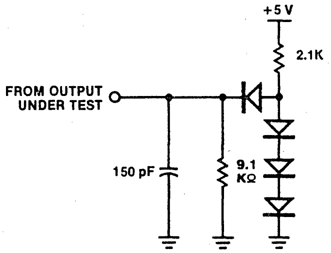


Figure 14. Test Load 1

DC Characteristics

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8	V_{CC}	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	-0.3	0.8	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{RH}	Reset Input High Voltage	3.8	V_{CC}	V	
V_{RL}	Reset Input Low Voltage	-0.3	0.8	V	
V_{OH}	Output High Voltage	2.4		V	$I_{OH} = -250\ \mu A$
V_{OL}	Output Low Voltage		0.4	V	$I_{OL} = +2.0$ mA
I_{IL}	Input Leakage	-10	10	μA	$0\ V \le V_{IN} \le +5.25\ V$
I_{OL}	Output Leakage	-10	10	μA	$0\ V \le V_{IN} \le +5.25\ V$
I_{IR}	Reset Input Current		-50	μA	$V_{CC} = +5.25\ V,\ V_{RL} = 0\ V$
I_{CC}	V_{CC} Supply Current		150	mA	

External I/O or Memory Read and Write Timing

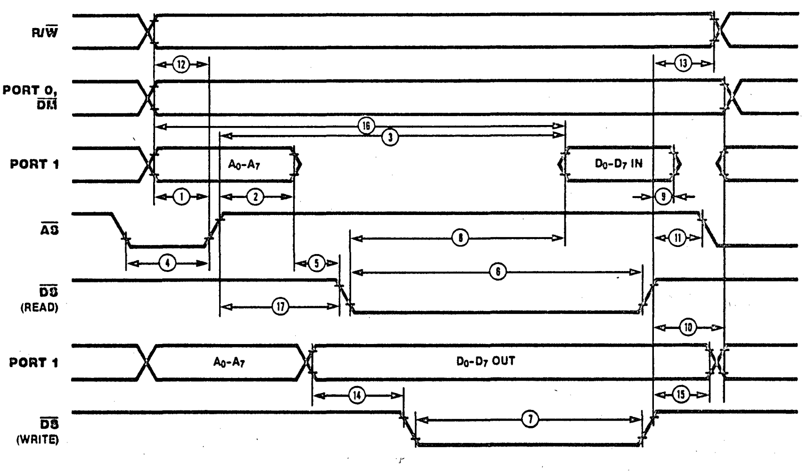


Figure 15. External I/O or Memory Read/Write

No.	Symbol	Parameter	Min	Max	Notes*†°
1	TdA(AS)	Address Valid to $\overline{AS}$ ↑ Delay	35		2,3
2	TdAS(A)	$\overline{AS}$ ↑ to Address Float Delay	45		2,3
3	TdAS(DR)	$\overline{AS}$ ↑ to Read Data Required Valid		220	1,2,3
4	TwAS	$\overline{AS}$ Low Width	55		1,2,3
5	TdAz(DS)	Address Float to $\overline{DS}$ ↓	0		
6	TwDSR	$\overline{DS}$ (Read) Low Width	185		1,2,3
7	TwDSW	$\overline{DS}$ (Write) Low Width	110		1,2,3
8	TdDSR(DR)	$\overline{DS}$ ↓ to Read Data Required Valid		130	1,2,3
9	ThDR(DS)	Read Data to $\overline{DS}$ ↑ Hold Time	0		
10	TdDS(A)	$\overline{DS}$ ↑ to Address Active Delay	45		2,3
11	TdDS(AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	55		2.3
12	TdR/W(AS)	R/$\overline{W}$ Valid to $\overline{AS}$ ↑ Delay	30		2,3
13	TdDS(R/W)	$\overline{DS}$ ↑ to R/$\overline{W}$ Not Valid	35		2,3
14	TdDW(DSW)	Write Data Valid to $\overline{DS}$ (Write) ↓ Delay	35		2,3
15	TdDS(DW)	$\overline{DS}$ ↑ to Write Data Not Valid Delay	45		2,3
16	TdA(DR)	Address Valid to Read Data Required Valid		255	1,2,3
17	TdAS(DS)	$\overline{AS}$ ↑ to $\overline{DS}$ ↓ Delay	55		2,3

NOTES:
1. When using extended memory timing add 2 TpC.
2. Timing numbers given are for minimum TpC.
3. See clock cycle time dependent characteristics table.

† Test Load 1.
° All timing references use 2.0 V for a logic "1" and 0.8 V for a logic "0".
* All units in nanoseconds (ns).

Additional Timing Table

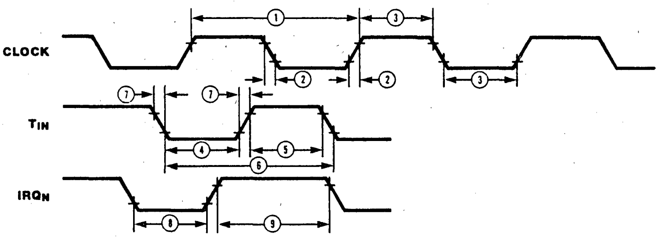


Figure 16. Additional Timing

No.	Symbol	Parameter	Min	Max	Notes*
1	TpC	Input Clock Period	80	1000	1
2	TrC,TfC	Clock Input Rise And Fall Times		15	1
3	TwC	Input Clock Width	26		1
4	TwTinL	Time Input Low Width	70		2
5	TwTinH	Timer Input High Width	3TpC		2
6	TpTin	Timer Input Period	8TpC		2
7	TrTin,TfTin	Timer Input Rise And Fall Times		100	2
8a	TwIL	Interrupt Request Input Low Time	70		2,3
8b	TwIL	Interrupt Request Input Low Time	3TpC		2,4
9	TwIH	Interrupt Request Input High Time	3TpC		2,3

NOTES:
1. Clock timing references uses 3.8 V for a logic "1" and 0.8 V for a logic "0".
2. Timing reference uses 2.0 V for a logic "1" and 0.8 V for a logic "0".
3. Interrupt request via Port 3 ($P3_1$-$P3_3$).
4. Interrupt request via Port 3 ($P3_0$).
* Units in nanoseconds (ns).

Memory Port Timing

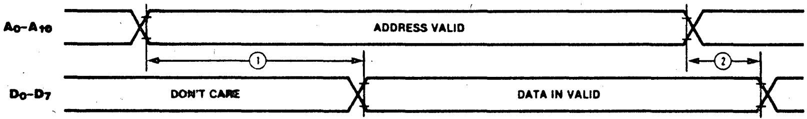


Figure 17. Memory Port Timing

No.	Symbol	Parameter	Min	Max	Notes*
1	TdA(DI)	Address Valid to Data Input Delay		320	1,2
2	ThDI(A)	Data In Hold time	0		1

NOTES:
1. Test Load 2.
2. This is a Clock-Cycle-Dependent parameter. For clock frequencies other than the maximum, use the following formula: 5 TpC – 95

*Units are nanoseconds unless otherwise specified.

Handshake Timing

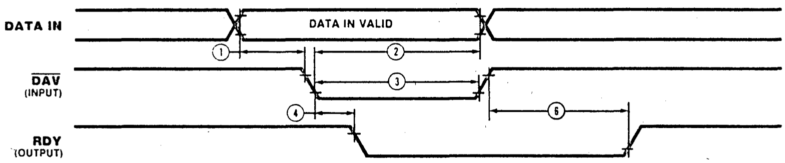


Figure 18a. Input Handshake

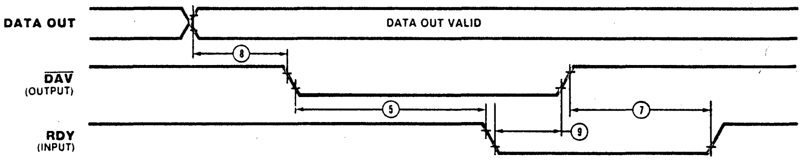


Figure 18b. Output Handshake

No.	Symbol	Parameter	Min	Max	Notes†*
1	TsDI(DAV)	Data In Setup Time	0		
2	ThDI(DAV)	Data In Hold time	160		
3	TwDAV	Data Available Width	120		
4	TdDAVIf(RDY)	$\overline{\text{DAV}}$ ↓ Input to RDY ↓ Delay		120	1,2
5	TdDAVOf(RDY)	$\overline{\text{DAV}}$ ↓ Output to RDY ↓ Delay	0		1,3
6	TdDAVIr(RDY)	$\overline{\text{DAV}}$ ↑ Input to RDY ↑ Delay		120	1,2
7	TdDAV0r(RDY)	$\overline{\text{DAV}}$ ↑ Output to RDY ↑ Delay	0		1,3
8	TdDO(DAV)	Data Out to $\overline{\text{DAV}}$ ↓ Delay	30		1
9	TdRDY(DAV)	Rdy ↓ Input to $\overline{\text{DAV}}$ ↑ Delay	0	140	1

NOTES:
1. Test load 1
2. Input handshake
3. Output handshake
† All timing references use 2.0 V for a logic "1" and 0.8 V for a logic "0".

* Units in nanoseconds (ns).

Clock-Cycle-Time-Dependent Characteristics

Number	Symbol	Equation
1	TdA(AS)	TpC-50
2	TdAS(A)	TpC-40
3	TdAS(DR)	4TpC-110*
4	TwAS	TpC-30
5	TwDSR	3TpC-65*
7	TwDSW	2TpC-55*
8	TdDSR(DR)	3TpC-120*
10	Td(DS)A	TpC-40
11	TdDS(AS)	TpC-30
12	TdR/W(AS)	TpC-55
13	TdDS(R/W)	TpC-50
14	TdDW(DSW)	TpC-50
15	TdDS(DW)	TpC-40
16	TdA(DR)	5TpC-160*
17	TdAS(DS)	TpC-30

*Add 2TpC when using extended memory timing.

MIL-STD-883 MILITARY PROCESSED PRODUCT

- Mil-Std-883 establishes uniform methods and procedures for testing microelectronic devices to insure the electrical, mechanical, and environmental integrity and reliability that is required for military applications.
- Mil-Std-883 Class B is the industry standard product assurance level for military ground and aircraft application.
- The total reliability of a system depends upon tests that are designed to stress specific quality and reliability concerns that affect microelectronic products.
- The following tables detail the 100% screening and electrical tests, sample electrical tests, and Qualification/ Quality Conformance testing required.

Zilog Military Product Flow

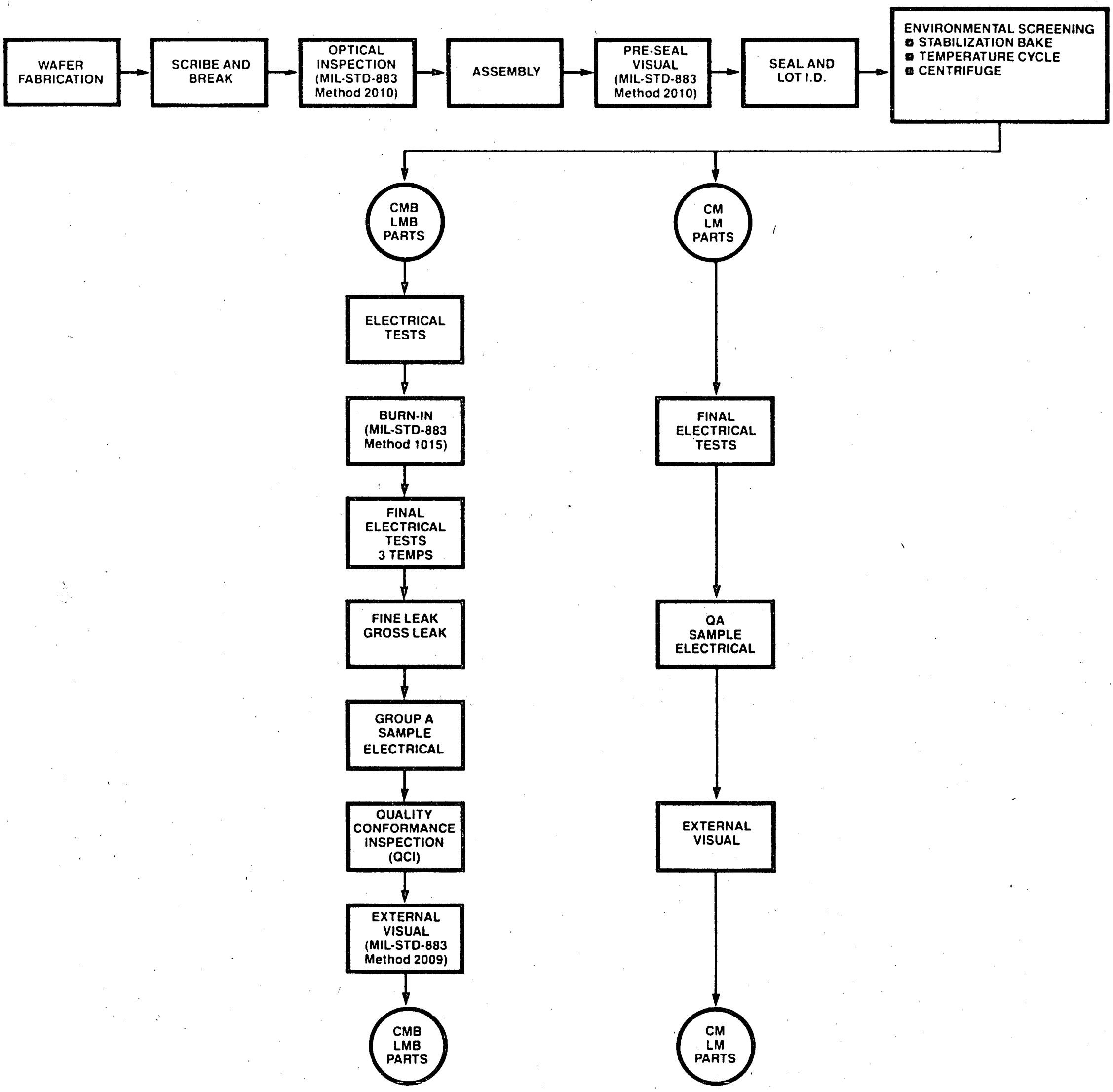


Table I
MIL-STD-883 Class B Screening Requirements
Method 5004

Test		Mil-Std-883 Method	Test Condition	Requirement
Internal Visual		2010	Condition B	100%
Stabilization Bake		1008	Condition C	100%
Temperature Cycle		1010	Condition C	100%
Constant Acceleration (Centrifuge)		2001	Condition E or D$^{(\text{Note 1})}$, Y_1 Axis Only	100%
Initial Electrical Tests			Zilog Military Electrical Specification Static/DC $T_C = +25°C$	100%
Burn-In		1015	Condition D$^{(\text{Note 2})}$, 160 hours, $T_A = +125°C$	100%
Interim Electrical Tests			Zilog Military Electrical Specification Static/DC $T_C = +25°C$	100%
PDA Calculation			PDA = 5%	100%
Final Electrical Tests			Zilog Military Electrical Specification Static/DC $T_C = +125°C, -55°C$ Functional, Switching/AC $T_C = +25°C$	100%
Fine Leak		1014	Condition A_2	100%
Gross Leak		1014	Condition C	100%
Quality Conformance Inspection (QCI)				
Group A	Each Inspection Lot	5005	(See Table II)	Sample
Group B	Every Week	5005	(See Table III)	Sample
Group C	Periodically (Note 3)	5005	(See Table IV)	Sample
Group D	Periodically (Note 3)	5005	(See Table V)	Sample
External Visual		2009		100%
QA—Ship				100%

NOTES:

1. Applies to larger packages which have an inner seal or cavity perimeter of two inches or more in total length or have a package mass of ≥5 grams.
2. In process of fully implementing of Condition D Burn-In Circuits. Contact factory for copy of specific burn-in circuit available.
3. Performed periodically as required by Mil-Std-883, paragraph 1.2.1 b(17).

Table II Group A

Sample Electrical Tests
MIL-STD-883 Method 5005

Subgroup	Tests	Temperature (T_C)	LTPD Max Accept = 2
Subgroup 1	Static/DC	+25°C	2
Subgroup 2	Static/DC	+125°C	3
Subgroup 3	Static/DC	−55°C	5
Subgroup 7	Functional	+25°C	2
Subgroup 8	Functional	−55°C and +125°C	5
Subgroup 9	Switching/AC	+25°C	2
Subgroup 10	Switching/AC	+125°C	3
Subgroup 11	Switching/AC	−55°C	5

NOTES:

- The specific parameters to be included for tests in each subgroup shall be as specified in the applicable detail electrical specification. Where no parameters have been identified in a particular subgroup or test within a subgroup, no Group A testing is required for that subgroup or test.
- A single sample may be used for all subgroup testing. Where required size exceeds the lot size, 100% inspection shall be allowed.
- Group A testing by subgroup or within subgroups may be performed in any sequence unless otherwise specified.

Table III Group B

Sample Test Performed Every Week to Test Construction and Insure Integrity of Assembly Process. MIL-STD-883 Method 5005

Subgroup	Mil-Std-883 Method	Test Condition	Quantity or LTPD/Max Accept
Subgroup 1			
Physical Dimensions	2016		2/0
Subgroup 2			
Resistance to Solvents	2015		4/0
Subgroup 3			
Solderability	2003	Solder Temperature +245°C ± 5°C	15(Note 1)
Subgroup 4			
Internal Visual and Mechanical	2014		1/0
Subgroup 5			
Bond Strength	2011	C	15(Note 2)
Subgroup 6(Note 3)			
Internal Water Vapor Content	1018	1000 ppm. maximum at +100°C	3/0 or 5/1
Subgroup 7(Note 4)			
Seal	1014		5
7a) Fine Leak		7a) A_2	
7b) Gross Leak		7b) C	
Subgroup 8(Note 5)			
Electrostatic Discharge Sensitivity	3015	Zilog Military Electrical Specification Static/DC T_C = +25°C A = 20-2000V B = >2000V Zilog Military Electrical Specification Static/DC T_C = +25°C	15/0

NOTES:

1. Number of leads inspected selected from a minimum of 3 devices.
2. Number of bond pulls selected from a minimum of 4 devices.
3. Test applicable only if the package contains a dessicant.
4. Test not required if either 100% or sample seal test is performed between final electrical tests and external visual during Class B screening.
5. Test required for initial qualification and product redesign.

Table IV Group C

Sample Test Performed Periodically to Verify Integrity of the Die.
MIL-STD-883 Method 5005

Subgroup	Mil-Std-883 Method	Test Condition	Quantity or LTPD/Max Accept
Subgroup 1			
Steady State Operating Life	1005	Condition D$^{(\text{Note } 1)}$, 1000 hours at +125°C	5
End Point Electrical Tests		Zilog Military Electrical Specification T_C = +25°C, +125°C, −55°C	
Subgroup 2			
Temperature Cycle	1010	Condition C	
Constant Acceleration (Centrifuge)	2001	Condition E or D$^{(\text{Note } 2)}$, Y_1 Axis Only	
Seal 2a) Fine Leak 2b) Gross Leak	1014	 2a) Condition A_2 2b) Condition C	15
Visual Examination	1010 or 1011		
End Point Electrical Tests		Zilog Military Electrical Specification T_C = +25°C, +125°C, −55°C	

NOTE:

1. In process of fully implementing Condition D Burn-In Circuits. Contact factory for copy of specific burn-in circuit available.
2. Applies to larger packages which have an inner seal or cavity perimeter of two inches or more in total length or have a package mass of ≥5 grams.

Table V Group D

Sample Test Performed Periodically to Insure Integrity of the Package.
MIL-STD-883 Method 5005

Subgroup	Mil-Std-883 Method	Test Condition	Quantity or LTPD/Max Accept
Subgroup 1			
Physical Dimensions	2016		15
Subgroup 2			
Lead Integrity	2004	Condition B_2 or D(Note 1)	15
Subgroup 3			
Thermal Shock	1011	Condition B minimum, 15 cycles minimum	
Temperature Cycling	1010	Condition C, 100 cycles minimum	15
Moisture Resistance	1004		
Seal	1014		
3a) Fine Leak		3a) Condition A_2	
3b) Gross Leak		3b) Condition C	
Visual Examination	1004 or 1010		
End Point Electrical Tests		Zilog Military Electrical Specification T_C = +25°C, +125°C, −55°C	
Subgroup 4			
Mechanical Shock	2002	Condition B minimum	
Vibration Variable Frequency	2007	Condition A minimum	
Constant Acceleration (Centrifuge)	2001	Condition E or D(Note 2), Y_1 Axis Only	15
Seal	1014		
4a) Fine Leak		4a) Condition A_2	
4b) Gross Leak		4b) Condition C	
Visual Examination	1010 or 1011		
End Point Electrical Tests		Zilog Military Electrical Specification T_C = +25°C, +125°C, −55°C	
Subgroup 5			
Salt Atmosphere	1009	Condition A minimum	
Seal	1014		15
5a) Fine Leak		5a) Condition A_2	
5b) Gross Leak		5b) Condition C	
Visual Examination	1009		
Subgroup 6			
Internal Water Vapor Content	1018	5,000 ppm. maximum water content at +100°C	3/0 or 5/1
Subgroup 7(Note 3)			
Adhesion of Lead Finish	2025		15(Note 4)
Subgroup 8(Note 5)			
Lid Torque	2024		5/0

NOTES:

1. Lead Integrity Condition D for leadless chip carriers.
2. Applies to larger packages which have an inner seal or cavity perimeter of two inches or more in total length or have a package mass of ⩾5 grams.
3. Not applicable to leadless chip carriers.
4. LTPD based on number of leads.
5. Not applicable for solder seal packages.

Zilog **Product Specification**

Z8® Z8681 Military ROMless Microcomputer

June 1987

FEATURES

- Complete microcomputer, 24 I/O lines, and up to 64K bytes of addressable external space each for program and data memory.
- 143-byte register file, including 124 general-purpose registers, three I/O port registers, and 16 status and control registers.
- Vectored, priority interrupts for I/O, counter/timers, and UART.
- On-chip oscillator that accepts crystal or external clock drive.
- Full-duplex UART and two programmable 8-bit counter/timers, each with a 6-bit programmable prescaler.
- Register Pointer so that short, fast instructions can access any one of the nine working-register groups.
- Single +5V power supply—all I/O pins TTL-compatible.
- Available in 8 MHz.

GENERAL DESCRIPTION

The Z8681 is the ROMless version of the Z8 single-chip microcomputer. The Z8681 offers all the outstanding features of the Z8 family architecture except an on-chip program ROM. Use of external memory rather than a preprogrammed ROM enables this Z8 microcomputer to be used in low volume applications or where code flexibility is required.

The Z8681 can provide up to 16 output address lines, thus permitting an address space of up to 64K bytes of data or program memory. Eight address outputs (AD_0-AD_7) are provided by a multiplexed, 8-bit, Address/Data bus. The remaining 8 bits can be provided by the software configuration of Port 0 to output address bits A_8-A_{15}.

Available address space can be doubled (up to 128K bytes) by programming bit 4 of Port 3 ($P3_4$) to act as a data memory select output (DM). The two states of DM together with the 16 address outputs can define separate data and memory address spaces of up to 64Kbytes each.

There are 143 bytes of RAM located on-chip and organized as a register file of 124 general-purpose registers, 16 control and status registers, and three I/O port registers. This register file can be divided into nine groups of 16 working registers each. Configuring the register file in this manner allows the use of short format instructions; in addition, any of the individual registers can be accessed directly.

ABSOLUTE MAXIMUM RATINGS

Guaranteed by characterization/design

Voltages on all pins except $\overline{\text{RESET}}$ with respect to GND	−0.3V to +7.0V
Operating Case Temperature	−55°C to +125°C
Storage Temperature Range	−65°C to +150°C
Absolute Maximum Power Dissipation	1.7 W

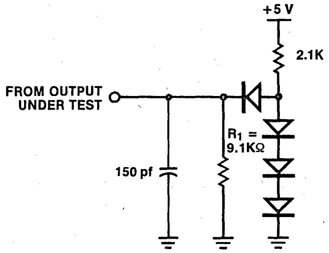
Stresses greater than those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; operation of the device at any condition above those indicated in the operational sections of these specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

STANDARD TEST CONDITIONS

The DC characteristics listed below apply for the following standard test conditions, unless otherwise noted. All voltages are referenced to GND (0V). Positive current flows into the referenced pin.

Military Operating Temperature Range (T_C)
−55°C to +125°C

Standard Military Test Condition
$+4.5 \leqslant V_{CC} \leqslant +5.5V$



Test Load

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Unit	Condition
V_{CH}	Clock Input High Voltage	3.8[a]	V_{CC}[b]	V	Driven by External Clock Generator
V_{CL}	Clock Input Low Voltage	−0.3[b]	0.8[a]	V	Driven by External Clock Generator
V_{IH}	Input High Voltage	2.0[a]	V_{CC}[b]	V	
V_{IL}	Input Low Voltage	−0.3[b]	0.8[a]	V	
V_{RH}	Reset Input High Voltage	3.8[a]	V_{CC}[b]	V	
V^{RL}	Reset Input Low Voltage	−0.3[b]	0.8[a]	V	
V_{OH}	Output High Voltage	2.4[a]		V	$I_{OH} = -250\,\mu A$
V_{OL}	Output Low Voltage		0.4[a]	V	$I_{OL} = +2.0$ mA
I_{IL}	Input Leakage	−10[a]	10[a]	μA	V_{IN} = 0V, 5.5V
I_{OL}	Output Leakage	−10[a]	10[a]	μA	V_{IN} = 0V, 5.5V
I_{IR}	Reset Input Current		−50[a]	μA	V_{CC} = MAX, V_{RL} = 0V
I_{CC}	V_{CC} Supply Current		230[a]	mA	All outputs and I/O pins floating

CAPACITANCE

Symbol	Parameter	Max	Unit
C_{MAX}	Maximum Capacitance	15[c]	pf

T_A = 25°C, f = 1 MHz.

Parameter Test Status:

[a] Tested
[b] Guaranteed
[c] Guaranteed by Characterization/Design

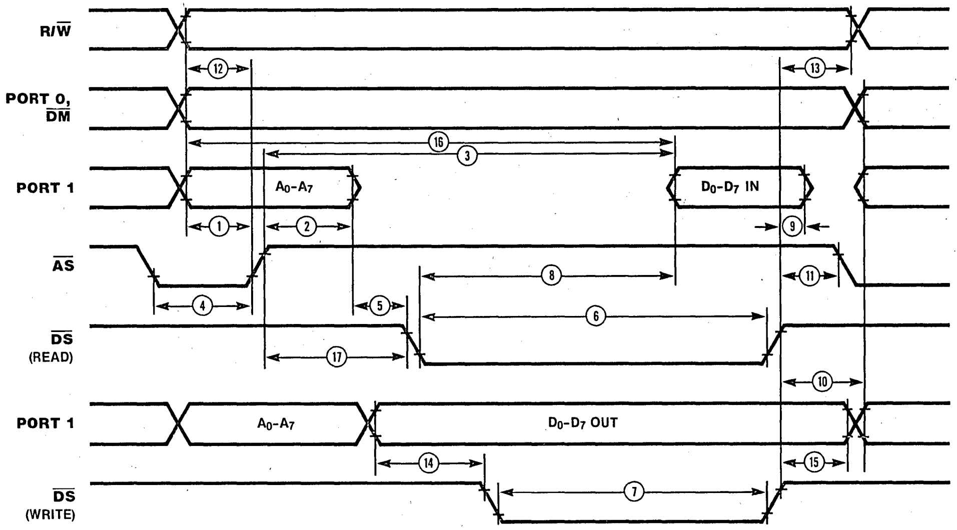


Figure 1. External I/O or Memory Read/Write Timing

AC CHARACTERISTICS

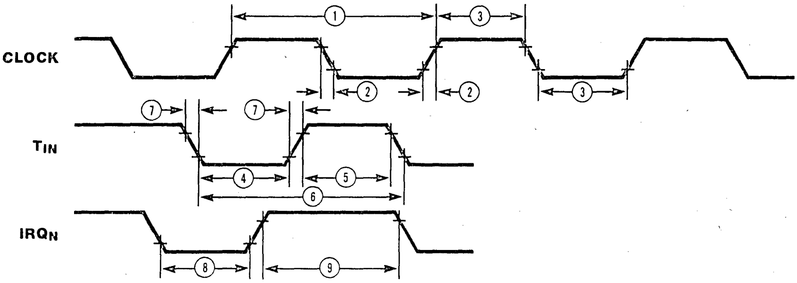
External I/O or Memory Read and Write Timing

			Z8681 8 MHz		
Number	Symbol	Parameter	Min	Max	Notes* °
1	TdA(AS)	Address Valid to $\overline{AS}$ ↑ Delay	50a		2,3
2	TdAS(A)	$\overline{AS}$ ↑ to Address Float Delay	70a		2,3
3	TdAS(DR)	$\overline{AS}$ ↑ to Read Data Required Valid		420a	1,2,3
4	TwAS	$\overline{AS}$ Low Width	80a		2,3
5	TdAz(DS)	Address Float to $\overline{DS}$ ↓	0b		
6	TwDSR	$\overline{DS}$ (Read) Low Width	250a		1,2,3
7	TwDSW	$\overline{DS}$ (Write) Low Width	160a		1,2,3
8	TdDSR(DR)	$\overline{DS}$ ↓ to Read Data Required Valid		200a	1,2,3
9	ThDR(DS)	Read Data to $\overline{DS}$ ↑ Hold Time	0a		
10	TdDS(A)	$\overline{DS}$ ↑ to Address Active Delay	70a		2,3
11	TdDS(AS)	$\overline{DS}$ ↑ to $\overline{AS}$ ↓ Delay	70a		2,3
12	TdR/W(AS)	R/$\overline{W}$ Valid to $\overline{AS}$ ↑ Delay	50a		2,3
13	TdDS(R/W)	$\overline{DS}$ ↑ to R/$\overline{W}$ Not Valid	60a		2,3
14	TdDW(DSW)	Write Data Valid to $\overline{DS}$ (Write) ↓ Delay	50a		2,3
15	TdDS(DW)	$\overline{DS}$ ↑ to Write Data Not Valid Delay	60a		2,3
16	TdA(DR)	Address Valid to Read Data Required Valid		410a	1,2,3
17	TdAS(DS)	$\overline{AS}$ ↑ to $\overline{DS}$ ↓ Delay	80a		2,3

NOTES:
1. When using extended memory timing add 2 TpC.
2. Timing numbers given are for minimum TpC.
3. See clock cycle time dependent characteristics table.

* All units in nanoseconds (ns).
° All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".

Parameter Test Status:
a Tested
b Guaranteed
c Guaranteed by Characterization/Design



AC CHARACTERISTICS

Additional Timing Table

			Z8681 8 MHz		
Number	**Symbol**	**Parameter**	**Min**	**Max**	**Notes***
1	TpC	Input Clock Period	125[a]	1000[a]	1
2	TrC,TfC	Clock Input Rise and Fall Times		25[b]	1
3	TwC	Input Clock Width	37[b]		1
4	TwTinL	Timer Input Low Width	100[b]		2
5	TwTinH	Timer Input High Width	3TpC[b]		2
6	TpTin	Timer Input Period	8TpC[b]		2
7	TrTin,TfTin	Timer Input Rise and Fall Times		100[b]	2
8A	TwIL	Interrupt Request Input Low Time	100[b]		2,3,4
8B	TwIL	Interrupt Request Input Low Time	3TpC[b]		2,3,5
9	TwIH	Interrupt Request Input High Time	3TpC[b]		2,3

NOTES:

1. Clock timing references use 3.8V for a logic "1" and 0.8V for a logic "0".
2. Timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
3. Interrupt request via Port 3.
4. Interrupt request via Port 3 ($P3_1$-$P3_3$)
5. Interrupt request via Port 3 ($P3_0$)

* Units in nanoseconds (ns).

Parameter Test Status:

[a] Tested
[b] Guaranteed
[c] Guaranteed by Characterization/Design

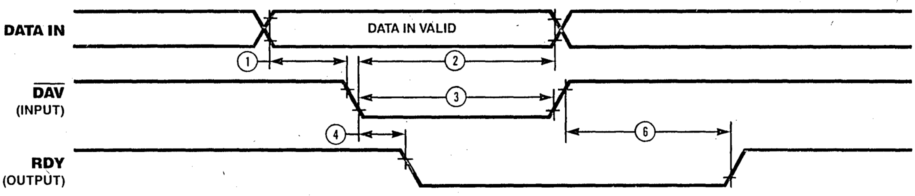


Figure 3a. Input Handshake Timing

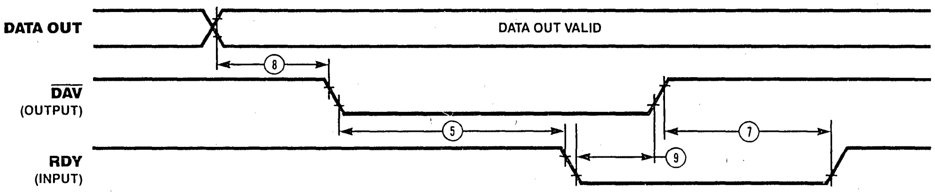


Figure 3b. Output Handshake Timing

AC CHARACTERISTICS

Handshake Timing

			Z8681		
Number	**Symbol**	**Parameter**	**Min**	**Max**	**Notes†***
1	TsDI(DAV)	Data In Setup Time	0[a]		
2	ThDI(DAV)	Data In Hold Time	230[a]		
3	TwDAV	Data Available Width	175[a]		
4	TdDAVIf(RDY)	$\overline{\text{DAV}}$ ↓ Input to RDY ↓ Delay		175[a]	1
5	TdDAVOf(RDY)	$\overline{\text{DAV}}$ ↓ Output to RDY ↓ Delay	0[a]		2
6	TdDAVIr(RDY)	$\overline{\text{DAV}}$ ↑ Input to RDY ↑ Delay		175[a]	1
7	TdDAVOr(RDY)	$\overline{\text{DAV}}$ ↑ Output to RDY ↑ Delay	0[a]		2
8	TdDO(DAV)	Data Out to $\overline{\text{DAV}}$ ↓ Delay	50[a]		
9	TdRDY(DAV)	Rdy ↓ Input to $\overline{\text{DAV}}$ ↑ Delay	0[b]	200[a]	

NOTES:
1. Input handshake
2. Output handshake
† All timing references use 2.0V for a logic "1" and 0.8V for a logic "0".
* Units in nanoseconds (ns).

Parameter Test Status:

[a] Tested
[b] Guaranteed
[c] Guaranteed by Characterization/Design

CLOCK CYCLE TIME-DEPENDENT CHARACTERISTICS

Number	Symbol	Z8681 8 MHz Equation
1	TdA(AS)	TpC-75
2	TdAS(A)	TpC-55
3	TdAS(DR)	4TpC-140 *
4	TwAS	TpC-45
6	TwDSR	3TpC-125 *
7	TwDSW	2TpC-90 *
8	TdDSR(DR)	3TpC-175 *
10	Td(DS)A	TpC-55
11	TdDS(AS)	TpC-55
12	TdR/W(AS)	TpC-75
13	TdDS(R/W)	TpC-65
14	TdDW(DSW)	TpC-75
15	TdDS(DW)	TpC-55
16	TdA(DR)	5TpC-215 *
17	TdAS(DS)	TpC-45

* Add 2TpC when using extended memory timing

PIN DESCRIPTION

$\overline{\text{AS}}$. *Address Strobe* (output, active Low). Address Strobe is pulsed once at the beginning of each machine cycle. Addresses output via Port 1 for all external program or data memory transfers are valid at the trailing edge of $\overline{\text{AS}}$.

$\overline{\text{DS}}$. *Data Strobe* (output, active Low). Data Strobe is activated once for each external memory transfer.

$P0_0$-$P0_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$. *I/O Port Lines* (input/outputs, TTL-compatible). These 24 lines are divided into three 8-bit I/O ports that can be configured under program control for I/O or external memory interface.

$P1_0$-$P1_7$. *Address/Data Port* (bidirectional). Multiplexed address (A_0-A_7) and data (D_0-D_7) lines used to interface with program and data memory.

$\overline{\text{RESET}}$. *Reset* (input, active Low). $\overline{\text{RESET}}$ initializes the Z8681. After $\overline{\text{RESET}}$ the Z8681 is in the extended memory mode. When $\overline{\text{RESET}}$ is deactivated, program execution begins from program location $000C_H$.

R/$\overline{\text{W}}$. *Read/Write* (output). R/$\overline{\text{W}}$ is Low when the Z8681 is writing to external program or data memory.

XTAL1, XTAL2. *Crystal 1, Crystal 2* (time-base input and output). These pins connect a parallel-resonant crystal to the on-chip clock oscillator and buffer.

PACKAGE PINOUTS

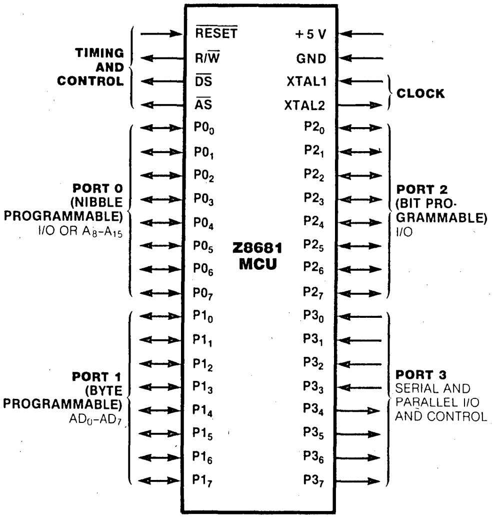


Figure 4. Pin Functions

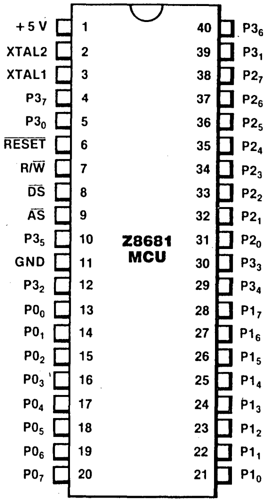


Figure 5. 40-pin Dual-In-Line Package (DIP), Pin Assignments

MIL-STD-883 MILITARY PROCESSED PRODUCT

- Mil-Std-883 establishes uniform methods and procedures for testing microelectronic devices to insure the electrical, mechanical, and environmental integrity and reliability that is required for military applications.
- Mil-Std-883 Class B is the industry standard product assurance level for military ground and aircraft application.
- The total reliability of a system depends upon tests that are designed to stress specific quality and reliability concerns that affect microelectronic products.
- The following tables detail the 100% screening and electrical tests, sample electrical tests, and Qualification/ Quality Conformance testing required.

Zilog Military Product Flow

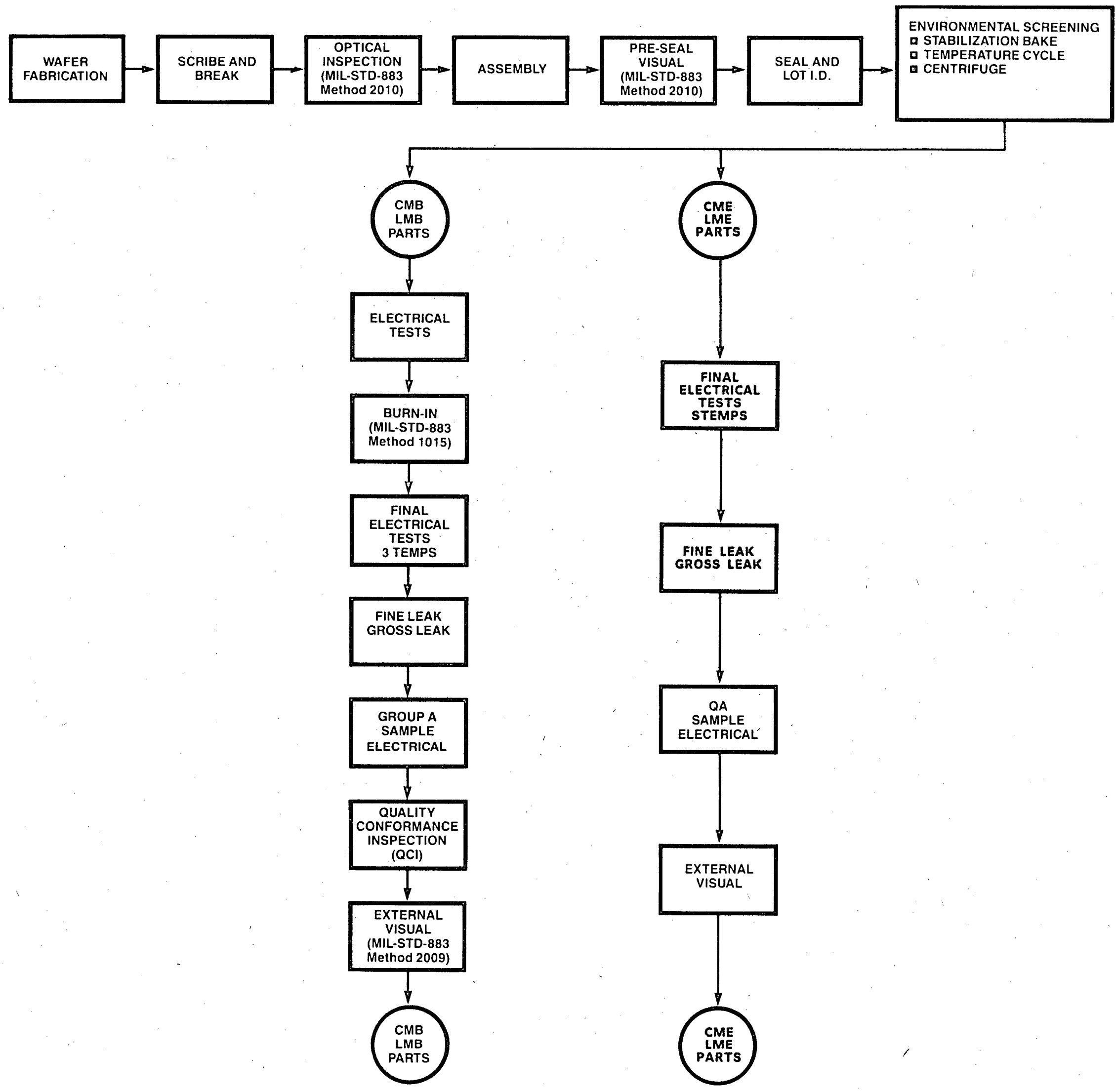

WAFER FABRICATION
SCRIBE AND BREAK
OPTICAL INSPECTION (MIL-STD-883 Method 2010)
ASSEMBLY
PRE-SEAL VISUAL (MIL-STD-883 Method 2010)
SEAL AND LOT I.D.
ENVIRONMENTAL SCREENING
□ STABILIZATION BAKE
□ TEMPERATURE CYCLE
□ CENTRIFUGE
CMB LMB PARTS
ELECTRICAL TESTS
BURN-IN (MIL-STD-883 Method 1015)
FINAL ELECTRICAL TESTS 3 TEMPS
FINE LEAK GROSS LEAK
GROUP A SAMPLE ELECTRICAL
QUALITY CONFORMANCE INSPECTION (QCI)
EXTERNAL VISUAL (MIL-STD-883 Method 2009)
CMB LMB PARTS
CME LME PARTS
FINAL ELECTRICAL TESTS STEMPS
FINE LEAK GROSS LEAK
QA SAMPLE ELECTRICAL
EXTERNAL VISUAL
CME LME PARTS

Table I
MIL-STD-883 Class B Screening Requirements
Method 5004

Test		Mil-Std-883 Method	Test Condition	Requirement
Internal Visual		2010	Condition B	100%
Stabilization Bake		1008	Condition C	100%
Temperature Cycle		1010	Condition C	100%
Constant Acceleration (Centrifuge)		2001	Condition E or D(Note 1), Y_1 Axis Only	100%
Initial Electrical Tests			Zilog Military Electrical Specification Static/DC $T_C = +25°C$	100%
Burn-In		1015	Condition D(Note 2), 160 hours, $T_A = +125°C$	100%
Interim Electrical Tests			Zilog Military Electrical Specification Static/DC $T_C = +25°C$	100%
PDA Calculation			PDA = 5%	100%
Final Electrical Tests			Zilog Military Electrical Specification Static/DC $T_C = +125°C, -55°C$ Functional, Switching/AC $T_C = +25°C$	100%
Fine Leak		1014	Condition B	100%
Gross Leak		1014	Condition C	100%
Quality Conformance Inspection (QCI)				
Group A	Each Inspection Lot	5005	(See Table II)	Sample
Group B	Every Week	5005	(See Table III)	Sample
Group C	Periodically (Note 3)	5005	(See Table IV)	Sample
Group D	Periodically (Note 3)	5005	(See Table V)	Sample
External Visual		2009		100%
QA—Ship				100%

NOTES:

1. Applies to larger packages which have an inner seal or cavity perimeter of two inches or more in total length or have a package mass of ≥5 grams.
2. In process of fully implementing of Condition D Burn-In Circuits. Contact factory for copy of specific burn-in circuit available.
3. Performed periodically as required by Mil-Std-883, paragraph 1.2.1 b(17).

Table II Group A

Sample Electrical Tests
MIL-STD-883 Method 5005

Subgroup	Tests	Temperature (T_C)	LTPD Max Accept = 2
Subgroup 1	Static/DC	+25°C	2
Subgroup 2	Static/DC	+125°C	3
Subgroup 3	Static/DC	−55°C	5
Subgroup 7	Functional	+25°C	2
Subgroup 8	Functional	−55°C and +125°C	5
Subgroup 9	Switching/AC	+25°C	2
Subgroup 10	Switching/AC	+125°C	3
Subgroup 11	Switching/AC	−55°C	5

NOTES:

- The specific parameters to be included for tests in each subgroup shall be as specified in the applicable detail electrical specification. Where no parameters have been identified in a particular subgroup or test within a subgroup, no Group A testing is required for that subgroup or test.
- A single sample may be used for all subgroup testing. Where required size exceeds the lot size, 100% inspection shall be allowed.
- Group A testing by subgroup or within subgroups may be performed in any sequence unless otherwise specified.

Table III Group B

Sample Test Performed Every Week to Test Construction and Insure Integrity of Assembly Process. MIL-STD-883 Method 5005

Subgroup	Mil-Std-883 Method	Test Condition	Quantity or LTPD/Max Accept
Subgroup 1			
Physical Dimensions	2016		2/0
Subgroup 2			
Resistance to Solvents	2015		4/0
Subgroup 3			
Solderability	2003	Solder Temperature +245°C ± 5°C	15(Note 1)
Subgroup 4			
Internal Visual and Mechanical	2014		1/0
Subgroup 5			
Bond Strength	2011	C	15(Note 2)
Subgroup 6(Note 3)			
Internal Water Vapor Content	1018	1000 ppm. maximum at +100°C	3/0 or 5/1
Subgroup 7(Note 4)			
Seal	1014		5
7a) Fine Leak		7a) B	
7b) Gross Leak		7b) C	
Subgroup 8(Note 5)			
Electrostatic Discharge Sensitivity	3015	Zilog Military Electrical Specification Static/DC T_C = +25°C A = 20-2000V B = >2000V Zilog Military Electrical Specification Static/DC T_C = +25°C	15/0

NOTES:

1. Number of leads inspected selected from a minimum of 3 devices.
2. Number of bond pulls selected from a minimum of 4 devices.
3. Test applicable only if the package contains a dessicant.
4. Test not required if either 100% or sample seal test is performed between final electrical tests and external visual during Class B screening.
5. Test required for initial qualification and product redesign.

Table IV Group C

Sample Test Performed Periodically to Verify Integrity of the Die.
MIL-STD-883 Method 5005

Subgroup	Mil-Std-883 Method	Test Condition	Quantity or LTPD/Max Accept
Subgroup 1			
Steady State Operating Life	1005	Condition D(Note 1), 1000 hours at +125°C	5
End Point Electrical Tests		Zilog Military Electrical Specification T_C = +25°C, +125°C, −55°C	
Subgroup 2			
Temperature Cycle	1010	Condition C	
Constant Acceleration (Centrifuge)	2001	Condition E or D(Note 2), Y_1 Axis Only	
Seal	1014		15
2a) Fine Leak		2a) Condition B	
2b) Gross Leak		2b) Condition C	
Visual Examination	1010 or 1011		
End Point Electrical Tests		Zilog Military Electrical Specification T_C = +25°C, +125°C, −55°C	

NOTE:

1. In process of fully implementing Condition D Burn-In Circuits. Contact factory for copy of specific burn-in circuit available.
2. Applies to larger packages which have an inner seal or cavity perimeter of two inches or more in total length or have a package mass of ⩾5 grams.

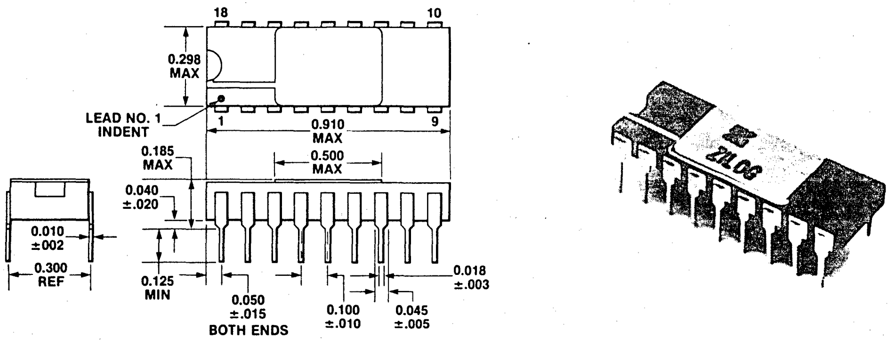
Table V Group D

Sample Test Performed Periodically to Insure Integrity of the Package.
MIL-STD-883 Method 5005

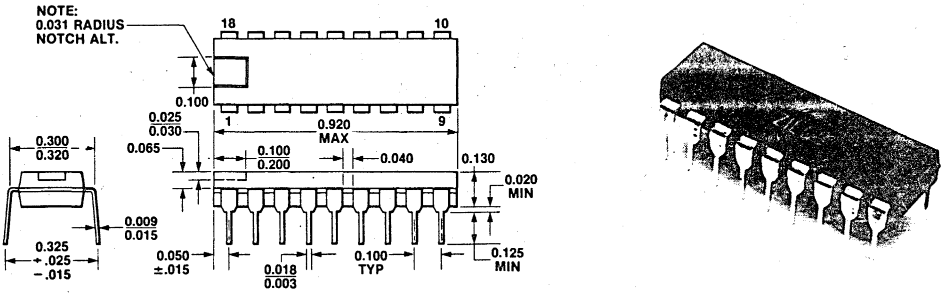
Subgroup	Mil-Std-883 Method	Test Condition	Quantity or LTPD/Max Accept
Subgroup 1			
Physical Dimensions	2016		15
Subgroup 2			
Lead Integrity	2004	Condition B_2 or D(Note 1)	15
Subgroup 3			
Thermal Shock	1011	Condition B minimum, 15 cycles minimum	
Temperature Cycling	1010	Condition C, 100 cycles minimum	15
Moisture Resistance	1004		
Seal	1014		
3a) Fine Leak		3a) Condition B	
3b) Gross Leak		3b) Condition C	
Visual Examination	1004 or 1010		
End Point Electrical Tests		Zilog Military Electrical Specification T_C = +25°C, +125°C, −55°C	
Subgroup 4			
Mechanical Shock	2002	Condition B minimum	
Vibration Variable Frequency	2007	Condition A minimum	
Constant Acceleration (Centrifuge)	2001	Condition E or D(Note 2), Y_1 Axis Only	15
Seal	1014		
4a) Fine Leak		4a) Condition B	
4b) Gross Leak		4b) Condition C	
Visual Examination	1010 or 1011		
End Point Electrical Tests		Zilog Military Electrical Specification T_C = +25°C, +125°C, −55°C	
Subgroup 5			
Salt Atmosphere	1009	Condition A minimum	
Seal	1014		15
5a) Fine Leak		5a) Condition B	
5b) Gross Leak		5b) Condition C	
Visual Examination	1009		
Subgroup 6			
Internal Water Vapor Content	1018	5,000 ppm. maximum water content at +100°C	3/0 or 5/1
Subgroup 7(Note 3)			
Adhesion of Lead Finish	2025		15(Note 4)
Subgroup 8(Note 5)			
Lid Torque	2024		5/0

NOTES:

1. Lead Integrity Condition D for leadless chip carriers.
2. Applies to larger packages which have an inner seal or cavity perimeter of two inches or more in total length or have a package mass of ≥5 grams.
3. Not applicable to leadless chip carriers.
4. LTPD based on number of leads.
5. Not applicable for solder seal packages.

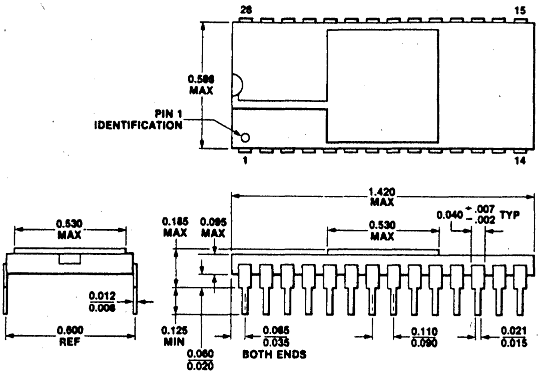


18-Pin Ceramic Package

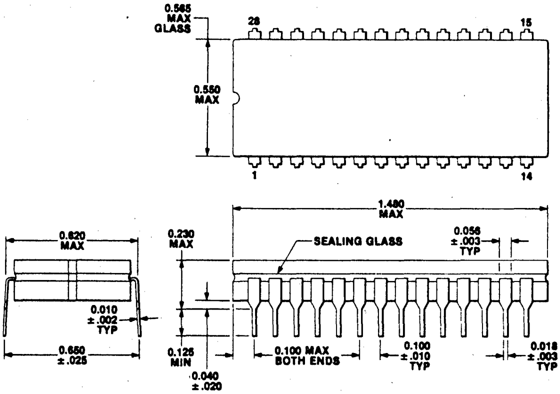


18-Pin Plastic Package

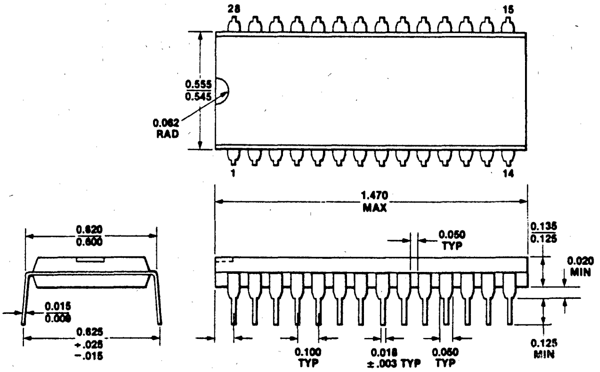
NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4



28-Pin Ceramic Package

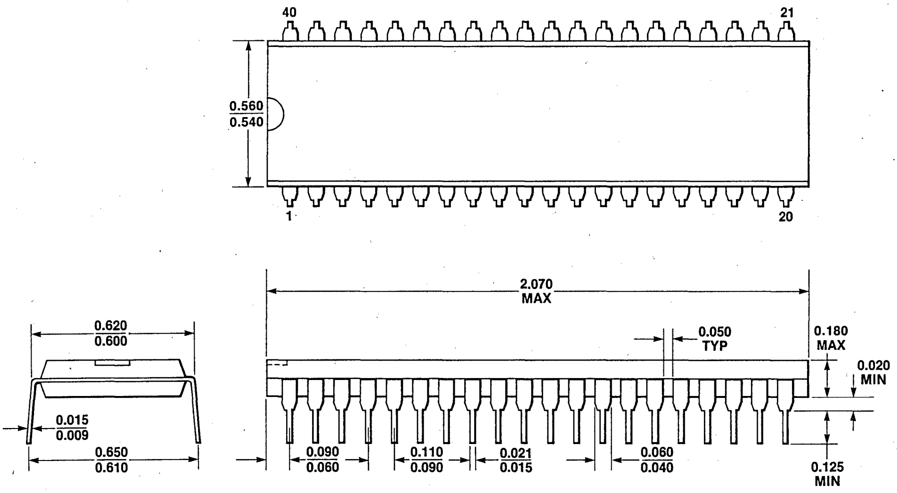


28-Pin Cerdip Package

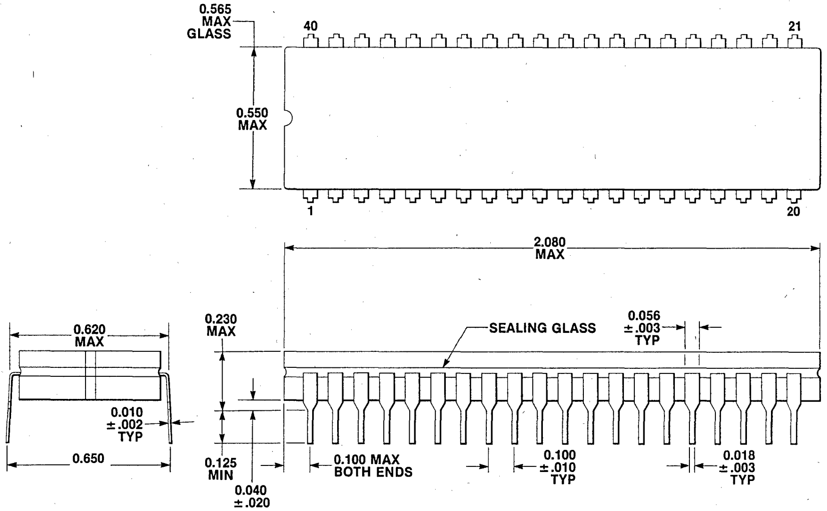


28-Pin Plastic Package

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

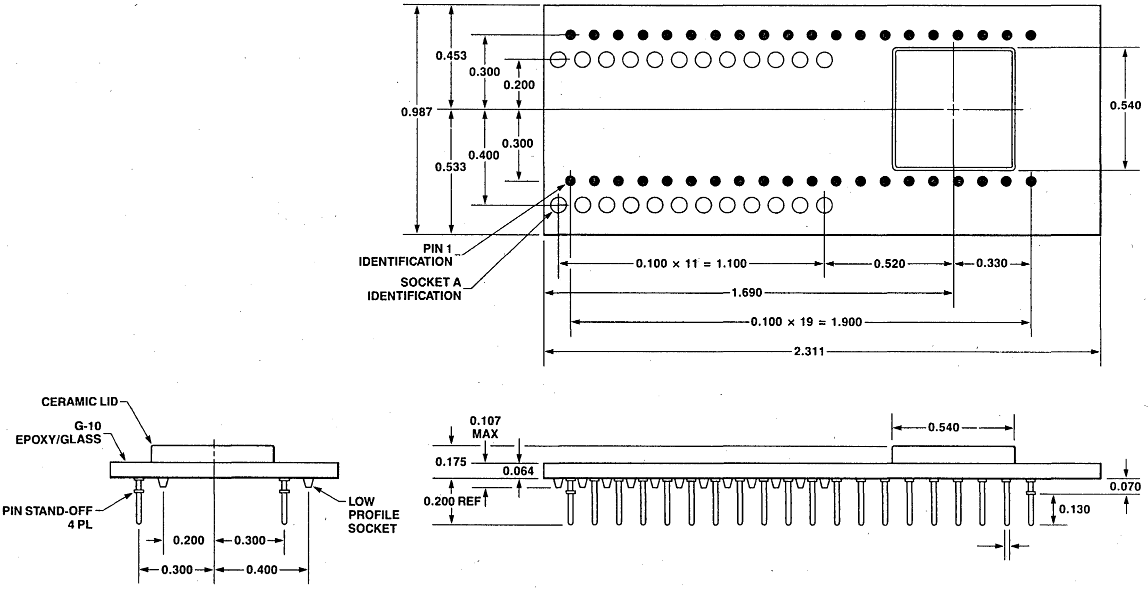


40-pin Plastic DIP



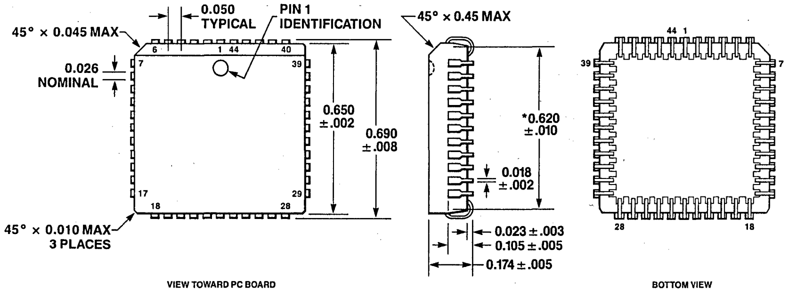
40-pin Cerdip Package

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.



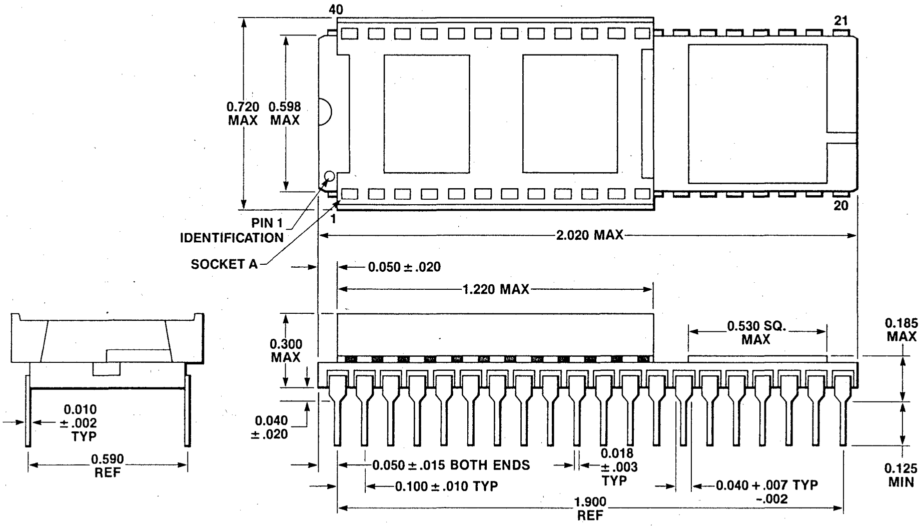
40-pin Low Profile Protopack

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.



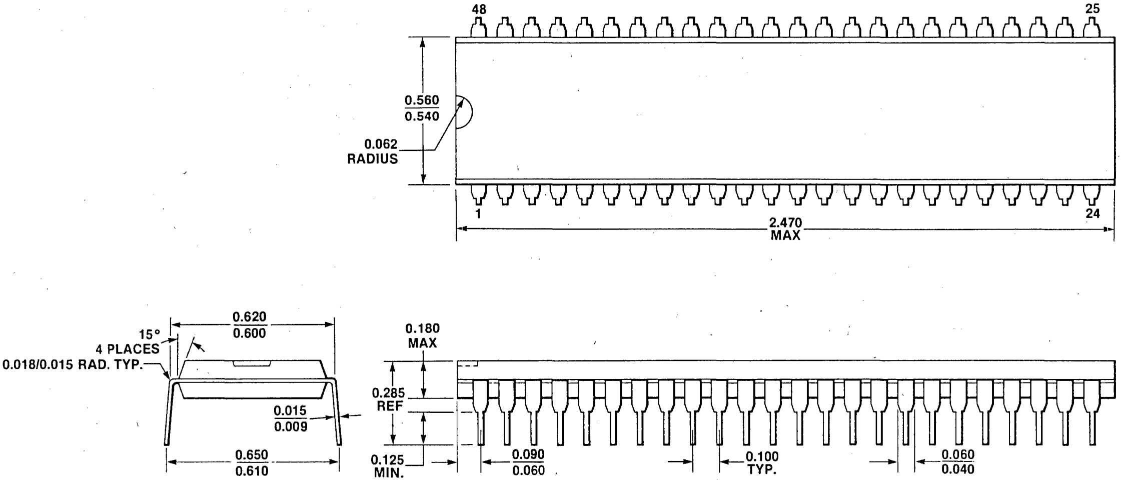
*DIMENSION FROM CENTER TO CENTER OF RADII

44-pin PCC

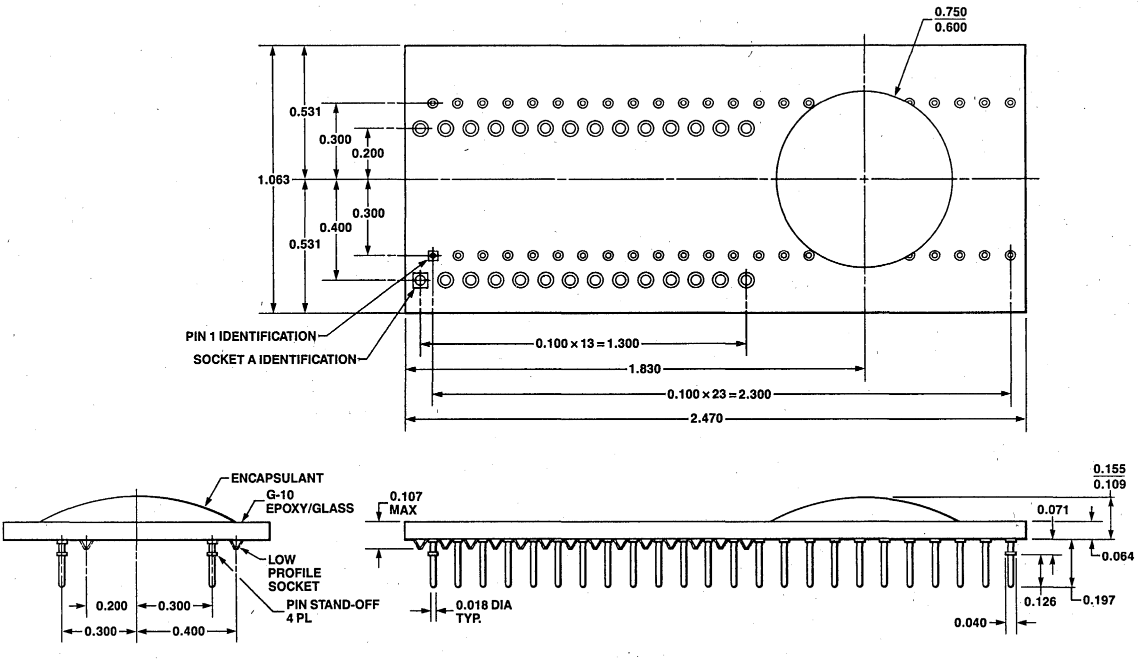


40-pin Protopack

PACKAGING INFORMATION (Continued)



48-Pin Dual-in-Line Package (DIP), Plastic



48-Pin Low Profile Protopack (T)

NOTE: Package dimensions are given in inches. To convert to millimeters, multiply by 25.4.

ORDERING INFORMATION

Z8 MCU, 2K ROM, 8 MHz
28-pin DIP

Z0860008PSCRXX
Z0860008PECRXX

Z8MCU

40-pin DIP	44-pin PCC	40-pin Protopak
2K ROM		2K XROM
Z0860112PSCRXXX	Z0860112VSCRXXX	Z0860312TSF
Z0860112DSERXXX		
Z0860112PECRXXX		
Z0860112DEERXXX		
4K ROM		4K XROM
Z0861112PSCRXXX	Z0861112VSCRXXX	Z0861312TSF
Z0861112PECRXXX		
Z0861112DSERXXX		

Z8 MCU with BASIC/Debug Interpreter, 8 MHz
40-pin DIP

Z0867108PSCR002
Z0867108PECR002

Z8681 ROMless MCU

40-pin DIP	44-pin PCC
8 MHz	
Z0868108PSC	Z0868108VSC
Z0868108DSE	
Z0868108PEC	
Z0868108DEE	
12 MHz	
Z0868112PEC	Z0868112VSC
Z0868112PSC	Z0868112VEC
Z0868112DSE	
Z0868112DEE	
16 MHz	
Z0868116PSC	Z0868116VSC

Low Cost ROMless MCU, 8 MHz
Z0868208PSC
Z0868408PSC

Low Power ROMless MCU, 8 MHz

40-pin DIP	44-pin PCC
Z0869108PSC	Z0869108VSC

Z8 ROMless MCU, 12 MHz

40-pin DIP	44-pin PCC
Z0869112PSC	Z0869112VSC
Z0869112PEC	

Z8 ROMless MCU, 16 MHz

40-pin DIP	44-pin PCC
Z0869116PSC	Z0869116VSC

Z8 MCU, 4K ROM, 12 MHz 40-pin DIP	Z8 MCU, 4K ROM, 16 MHz 40-pin DIP
Z86C1112PECRXXX	Z86C1116PSCRXXX
44-pin PLCC	44-pin PLCC
Z86C1112VECRXXX	Z86C1116VSCRXXX

Z8 MCU, 8K ROM
40-pin DIP

Z86C2112PECRXXX
Z86C2116PSCRXXX
Z86C2112CEARXXX

Z8 MCU, 8K PROM

40-pin DIP	44-pin PLCC
Z86E2112PEC	Z86C2112VECRXXX
Z86E2116PSC	Z86C2116VSCRXXX
Z86E2112CEA	

44-pin PLCC

Z86E2112VEC
Z86E2116VSC

Z8 ROMless MCU

40-pin DIP	44-pin PCC
Z86C9112PEC	Z86C9112VEC
Z86C9116PSC	Z86C9116VSC

Z8 4K ROM MCU, 12 MHz

Z0861112CMBRXXX

Z8 ROMless MCU, 8 MHz
40-pin DIP

Z0868108CMB

Z8 MCU, 4K ROM, 12 MHz
28-pin DIP

Z86C1012PSC

Z8 MCU, 8K ROM, 12 MHz
28-pin DIP

Z86C2012PSC

Codes

PACKAGE
Preferred
D = Cerdip
P = Plastic
V = Plastic Chip Carrier

Longer Lead Time
C = Ceramic
F = Plastic Quad Flat Pack
G = Ceramic PGA (Pin Grid Array)
L = Ceramic LCC
Q = Ceramic Quad-in-Line
R = Protopack
T = Low Profile Protopack

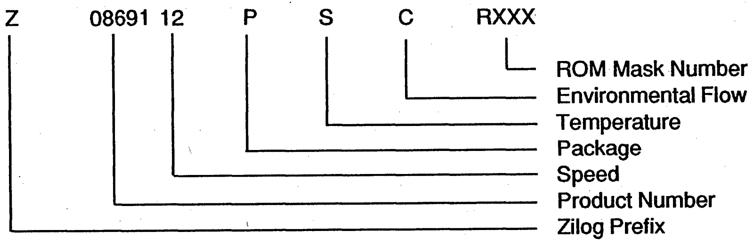
TEMPERATURE
Preferred
S = 0°C to +70°C

Longer Lead Time
E = -40°C to +85°C
M = -55°C to +125°C

ENVIRONMENTAL
Preferred
C = Plastic Standard
E = Hermetic Standard
F = Protopack Standard

Longer Lead Time
A = Hermetic Stressed
B = 833 Class B Military
D = Plastic Stressed
J = JAN 38510 Military

Example:
Z0869112PSC is a 12 MHz 8691 (ROMless Z8) in a plastc DIP, 0° C to +70° C, Standard Flow.



Z 08691 12 P S C RXXX

ROM Mask Number
Environmental Flow
Temperature
Package
Speed
Product Number
Zilog Prefix

ZILOG DOMESTIC SALES OFFICES AND TECHNICAL CENTERS

CALIFORNIA	
Agoura	818-707-2160
Campbell	408-370-8120
Tustin	714-838-7800
COLORADO	
Boulder	303-494-2905
FLORIDA	
Largo	813-585-2533
GEORGIA	
Norcross	404-923-8500
ILLINOIS	
Schaumburg	312-885-8080
NEW HAMPSHIRE	
Nashua	603-888-8590
MINNESOTA	
Edina	612-831-7611
NEW JERSEY	
Hasbrouck Hts.	201-288-3737
OHIO	
Seven Hills	216-447-1480
PENNSYLVANIA	
Ambler	215-653-0230
TEXAS	
Richardson	214-231-9090

INTERNATIONAL SALES OFFICES

CANADA	
Toronto	416-673-0634
GERMANY	
Munich	49-89-672-045
JAPAN	
Tokyo	81-3-5870528
HONG KONG	
Kowloon	852-3-7238979
R.O.C.	
Taiwan	886-2-7312420
UNITED KINGDOM	
Maidenhead	44-628-39200

The information contained herein is subject to change without notice. Zilog assumes no responsibility for the use of any circuitry embodied in a Zilog product. No other circuit patent licenses are implied.

All specifications (parameters) are subject to change without notice. The applicable Zilog test documentation wil l specify which parameters are tested.

Zilog, Inc. 210 Hacienda Ave., Campbell, CA 95008-6609
Telephone (408) 370-8000 TWX 910-338-7621

03-8275-02 Printed in USA