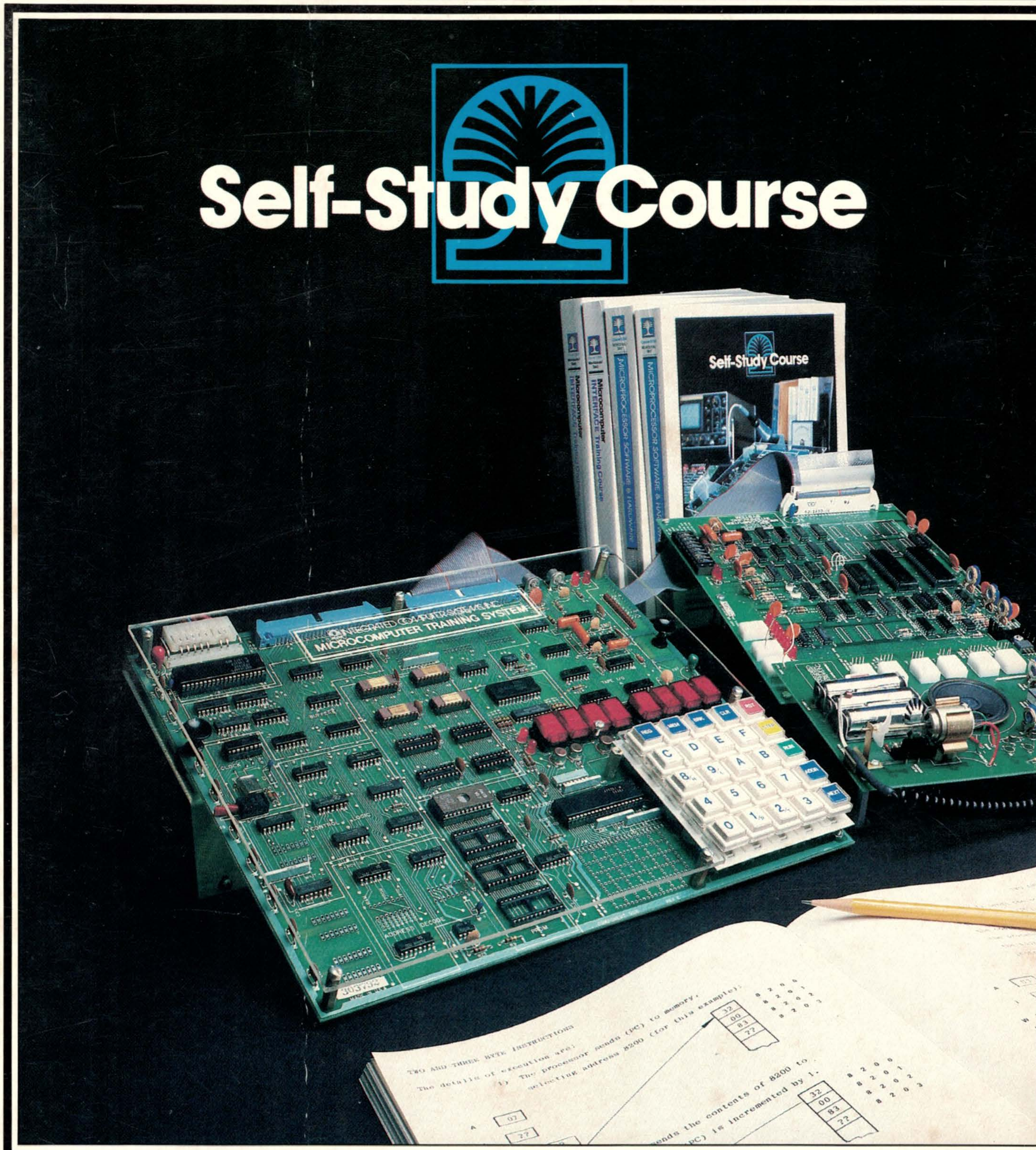


MICROPROCESSOR REAL-TIME INTERFACING

Workbook/Text

Volume 2

Self-Study Course

Course 536A:

MICROPROCESSOR REAL-TIME INTERFACING

Workbook/Text

Volume 2

DEVELOPED & PUBLISHED BY:
INTEGRATED COMPUTER SYSTEMS
Course Development Division
© Copyright 1980

SENIOR AUTHOR:
Edward Dillingham, M.E., M.S.E.E.

ASSISTED BY:
Dr. Daniel M. Forsyth
Dr. Rudolf Hirschmann
Ms. Ruth H. Savoie
Dr. David C. Collins

EDUCATION IS OUR BUSINESS™

D/9/80

MICROPROCESSOR REAL-TIME INTERFACING

Two Volumes
ISBN 0-89438-003-6

Volume I
ISBN 0-89438-004-4

Volume II
ISBN 0-89438-005-2

VOLUME I

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

VOLUME II

TABLE OF CONTENTS

VOLUME I

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS

VOLUME II

This page intentionally left blank.

MICROCOMPUTER INTERFACING WORKBOOK

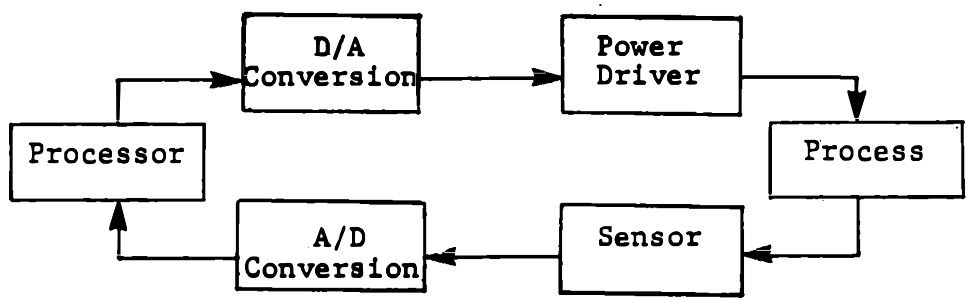
CHAPTER 6

CLOSED LOOP CONTROL

This page intentionally left blank.

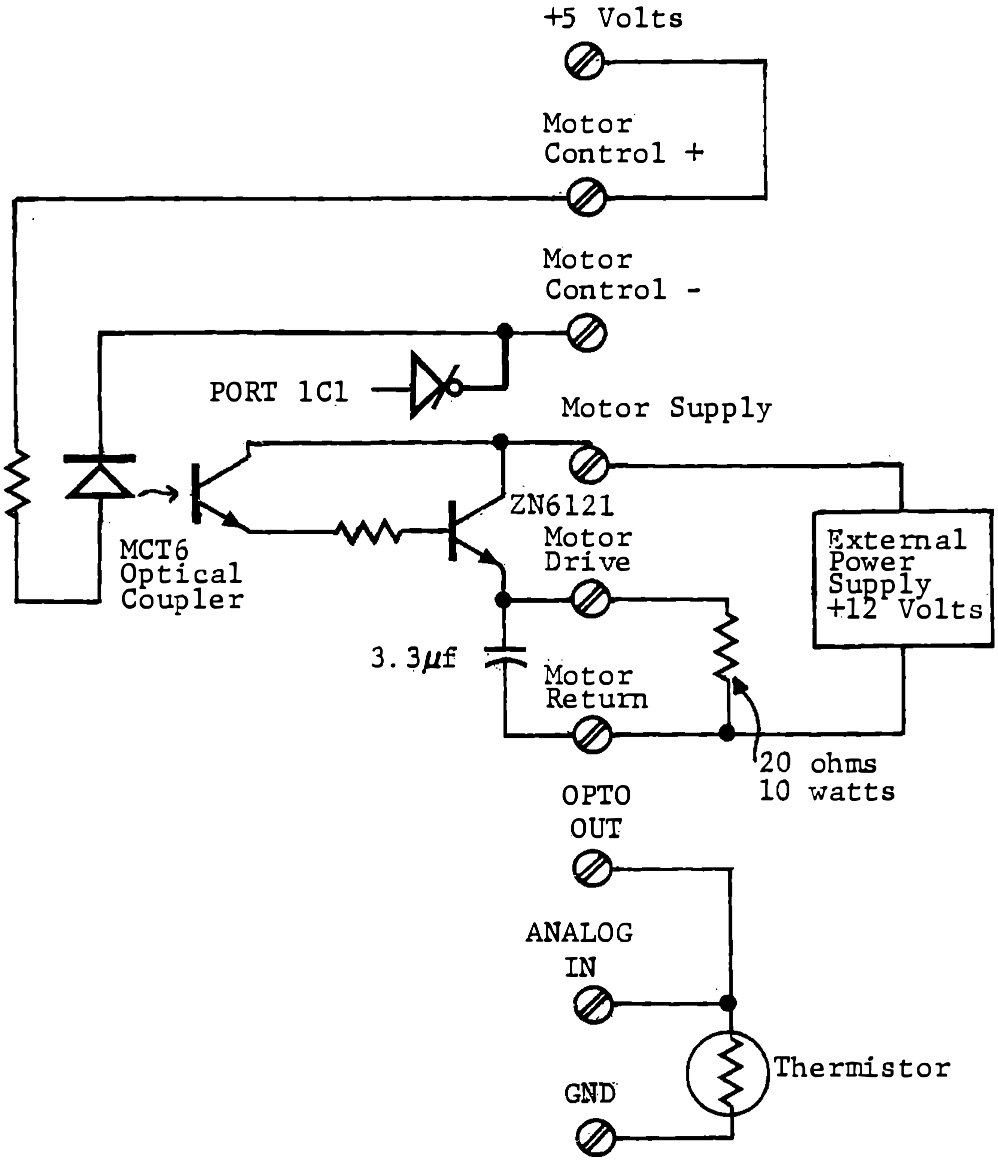
6. CLOSED LOOP CONTROL

As we have seen, a computer can generate control signals and receive sensed inputs. When the input is used to determine the control output, we have "closed the loop".



In this chapter we will deal with three closed loop problems: on-off (thermostat) control, proportional voltage or temperature control, and speed control.

Voltage and temperature control are essentially similar problems and will use the same forms of analog to digital input and digital to analog output. The automatic A/D input function of the Ferranti 425 will be used to sense the condition of the external "process", and output switching or pulse width modulation will be used for control. In speed control we will determine the speed by a frequency measurement, and we will try both PWM and the Ferranti D/A conversion for control.



Connections for Thermostat Exercise

Figure 6-1

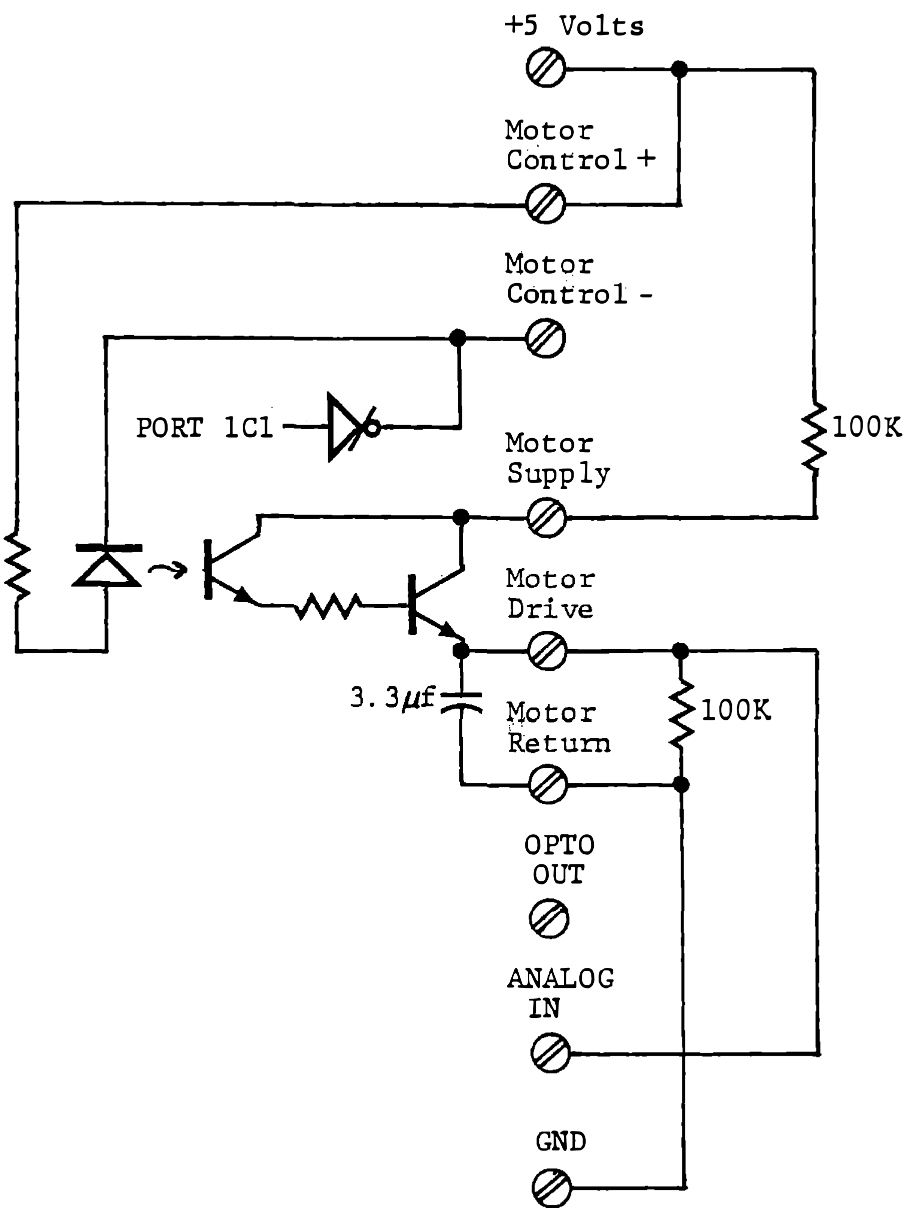
6.1 ON-OFF CONTROL

The simplest closed loop control system is exemplified by the familiar household thermostat. The temperature is measured, and if it is too low (i.e., below some preset value) the furnace is turned on. When the temperature reaches the desired value the furnace is turned off.

An on-off control system usually has a "dead band" in which the present condition of the output is not changed - if the furnace is on it stays on until an upper temperature limit is reached; if it is off it stays off until a low limit is reached.

Without the dead band the output will switch on and off too frequently, resulting in inefficient operation (and annoying noise in the household situation). In a simple thermostat, the dead band is provided by mechanical hysteresis in the bimetal switch. In more sophisticated systems the upper and lower limits can be programmed independently. It is also possible to let system time constants provide the dead band, as we shall see in the first exercise.

The interface board includes a Fairchild 2N6121 power transistor driven by a Monsanto MCT6 optical coupler. This circuit can be used directly to heat a power resistor, as shown in Figure 6-1. The thermistor must be coupled to the heater as closely as possible, since the heating available is not very great. One way of achieving close coupling is to wrap the resisitor and thermistor together with electrical tape.



Connections for On-Off Voltage Control

Figure 6-2

Alternately, the power transistor can drive a relay controlling power to an AC heater, with the thermistor in water that is being heated. (Note: Exercise 6.1.1 should not be done with a relay, because the fast switching would damage the relay contacts).

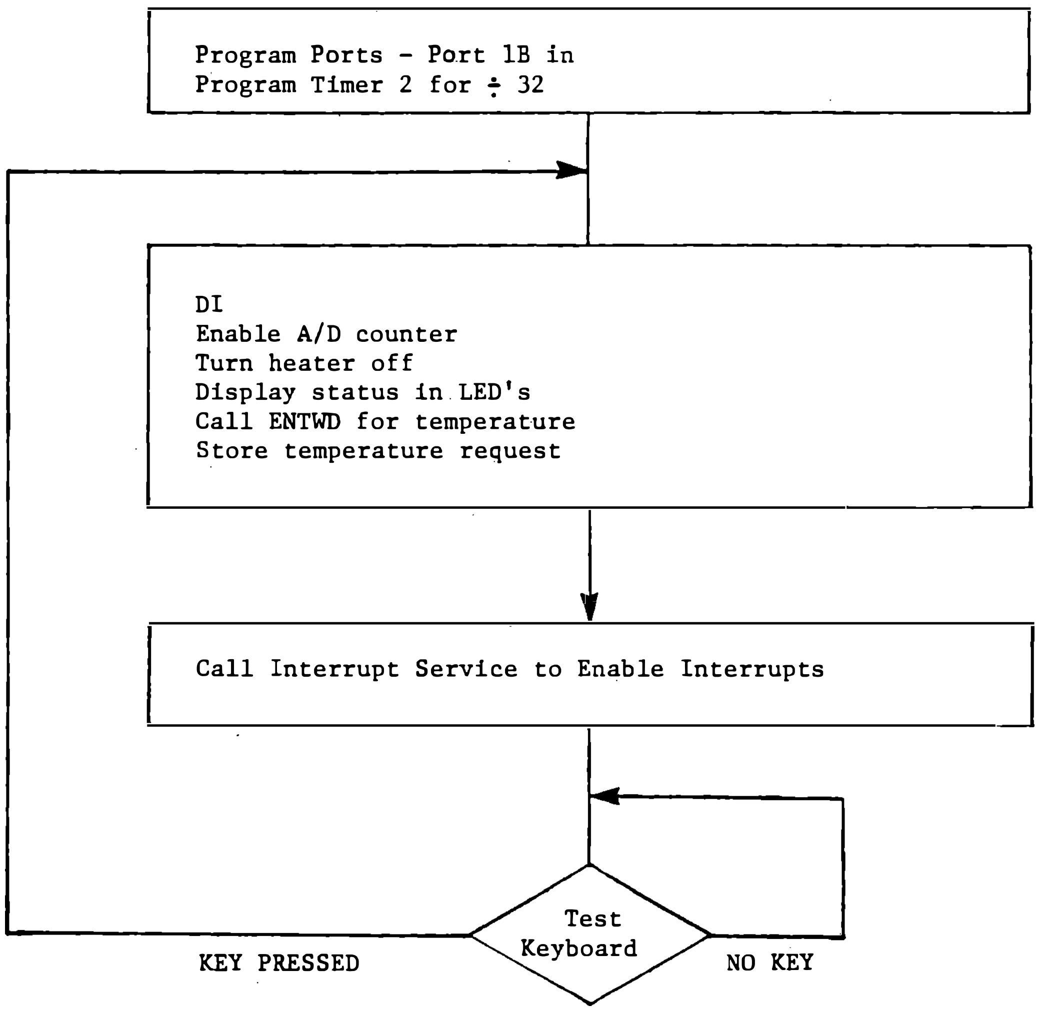
The experiments can be performed without heating anything, but simply simulating heat by the charge on a capacitor and measuring its voltage, as shown in Figure 6-2.

6.1.1 On-Off Control Without Deadband

EXERCISE

The on-off control experiment will use subroutines TEMP and FILTR, developed in Chapter 5. The program of Figure 6-3 accepts a temperature request (by keyboard entry) and attempts to heat the load to that temperature as measured by the thermistor. Whenever the temperature is less than that requested it turns the heater on by setting Port 1C1 low; when the temperature is greater it sets Port 1C1 high to turn the heater off. The temperature is calculated and displayed by subroutine TEMP. The measurement and control functions are contained entirely in the RST6 interrupt service, with the main program performing initialization and then calling ENTWD for a temperature request. The desired temperature must be entered as degrees and tenths, to permit comparison with the temperature returned by TEMP. For instance, enter 315 to request a temperature of 31.5^{o} C.

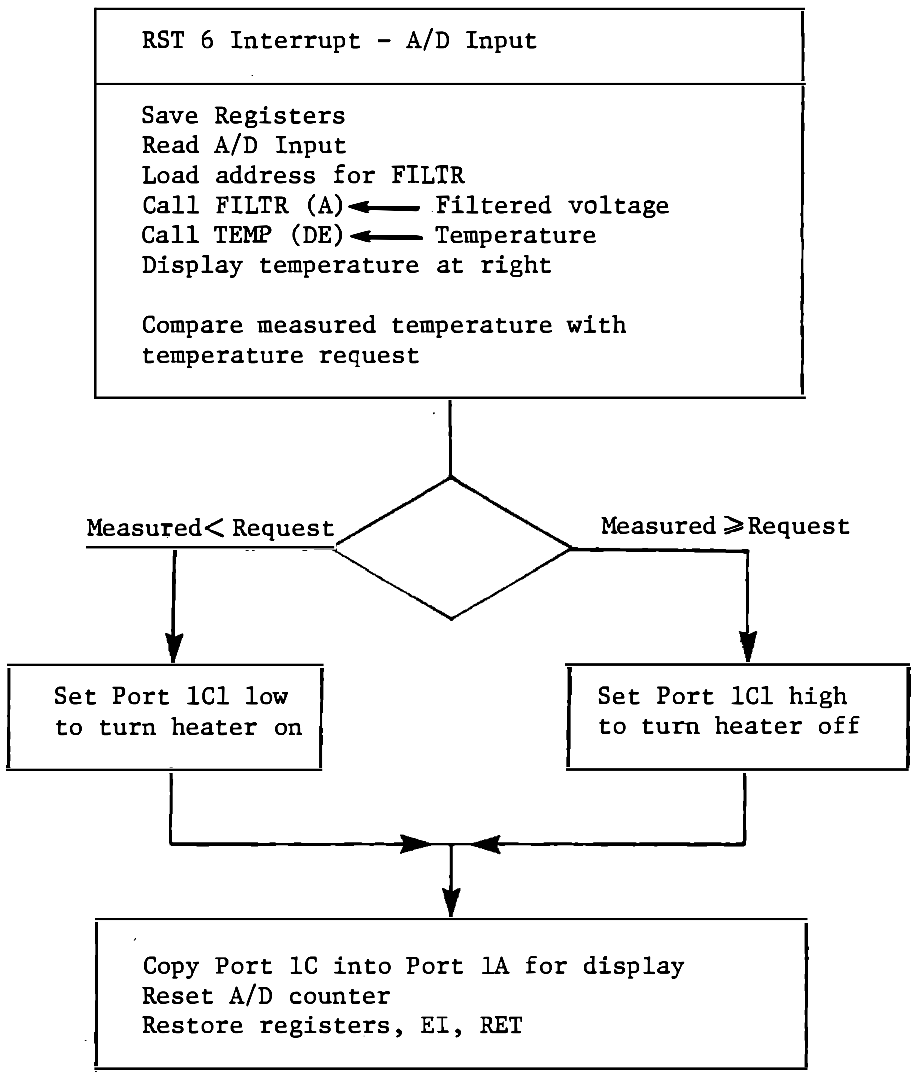
Clearly this program does nothing very exciting. Its main purpose is to demonstrate the effect of having no deadband. Observe the high frequency at which it switches the power on and off when the desired temperature is reached. This is no problem to the power transistor, nor to an SCR, but would damage the contacts of a relay. Moreover, many heating devices are less efficient when being switched on and off repeatedly, and some may be damaged.



Initialization

On-Off Control - No Deadband

Figure 6-3a



Interrupt Service

On-Off Control - No Deadband

Figure 6-3b

THERMOSTAT WITHOUT DEADBAND

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

A D D R	CODE		Label	Opcode	Operand	Comments
8 20 0	3E			MVI	A, 82	Program Ports
1	82					Port 1B In
2	D3			OUT	CNT1	
3	07					
4	3E			MVI	A, 92	
5	92					
6	D3			OUT	CNT2	
7	0F					
8	3E			MVI	A, 94	Program Timer 2
9	94					Low byte
A	D3			OUT	TIMCT	Mode 2
B	17					Binary
C	3E			MVI	A, 20	Load Timer 2
D	20					for ÷ 32
E	D3			OUT	TIM2	
F	16					
8 21 0	F3			DI		No interrupts while
1	3E			MVI	A, 03	in ENTWD
2	03					PIC0 high for A/D
3	D3			OUT	PORT1C	PIC1 high to
4	06					turn power off
5	D3			OUT	PORT1A	Display in LED's
6	04					
7	CD			CALL	ENTWD	Enter temperature
8	46					limit in degrees
9	03					and tenths --
A	22			SHLD	8306	(e.g., 273 for 27.3°C)
B	06					Store temperature
C	83					limit
D	F7			RST	6	Enable A/D interrupt
821 E	DB			IN	PORT0A	Test keyboard
F	00					(FF if no key)
8 22 0	3C			INR	A	
1	CA			JZ	821E	Loop until
2	1E					key pressed
3	82					
4	C3			JMP	8210	Goto DI and
5	10					turn power off
6	82					and accept data
7						
8						Figure 6-4a

THERMOSTAT - RST 6 INTERRUPT

ADDR	CODE	Instruction	Comment
8230	F5	PUSH PSW	
1	E5	PUSH H	
2	D5	PUSH D	
3	C5	PUSH B	
4	21	LXI H, 8300	Data address
5	00		for FILTR
6	83		
7	DB	IN PORT 1B	Read A/D
8	05		
9	CD	CALL FILTR	(A) ← filtered
A	70		voltage
B	82		
C	CD	CALL TEMP	Calculate and
D	B0		display temperature
E	82		(DE) ← temperature
F	2A	LHLD 8306	(HL) ← requested
8240	06		temperature
1	83		
2	7D	MOV A, L	Compare temperature
3	93	SUB E	to request
4	7C	MOV A, H	Sets CY if
5	9A	SBB D	temperature high
6	3E	MVI A, 01	(A) ← 03 if high
7	01		← 02 if low
8	17	RAL	
9	D3	OUT CNT1	Set Port 1C1 low
A	07		(power on) if
B	DB	IN PORT 1C	temperature low
C	06		
D	D3	OUT PORT 1A	Read and display
E	04		power control.
F	00	NOP	
8 0			
1			
2			
3			
4			
5			
6			
7			
8			Figure 6-4b

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET.

THERMOSTAT - EXIT FROM INTERRUPT

ADDR			CODE												
8	25	0	3	E	M	V	I	A	,	0	7				Enable A/D and
		1	0	7											reset counter
		2	D	3	O	U	T	C	N	T	2				
		3	0	F											
		4	C	1	P	O	P	B							
		5	D	1	P	O	P	D							
		6	E	1	P	O	P	H							
		7	F	1	P	O	P	P	S	W					
		8	F	B	E	I									
		9	C	9	R	E	T								
		A													
		B													
		C													
		D													
		E													
		F													
8		0			S	U	B	R	O	U	T	I	N	E	S REQUIRED
		1				F	I	L	T	R		8	2	7	0 - AF
		2				T	E	M	P			8	2	B	0 - EF
		3			D	A	T	A		R	E	Q	U	I	RED
		4				T	E	M	P		T	A	B	L	E 8310 - 836F
		5													
		6					8	3	0	0		0	4		n
		7					8	3	0	1		0	0		Preload for
		8					8	3	0	2		0	0		FILTR
		9					8	3	0	3		0	0		
		A					8	3	0	6					Temperature request
		B					8	3	0	7					stored by MAIN
		C													
		D													
		E													
		F													
8		0													
		1													
		2													
		3													
		4													
		5													
		6													
		7													
		8													Figure 6-4c

SUBROUTINE FILTR

ADDR	CODE		INSTRUCTION	COMMENT
8270	D5		PUSH D	Save (DE)
1	C5		PUSH B	Save (BC)
2	46		MOV B, M	(B) ← n
3	48		MOV C, B	(C) ← n
4	23		INX H	Address $2^n E_{i-1}$
5	E5		PUSH H	Save address
6	5E		MOV E, M	}
7	23		INX H	} (DE) ← $2^n E_{i-1}$
8	56		MOV D, M	}
9	6B		MOV L, E	(HL) ← $2^n E_{i-1}$
A	62		MOV H, D	
827B	29		DAD H	}
C	0D		DCR C	}
D	C2		JNZ 827B	} (HL) ← $2^{2n} E_{i-1}$
E	7B			}
F	82			}
8280	4F		MOV C, A	(C) ← V_i
1	7D		MOV A, L	}
2	93		SUB E	} (DE) ← (HL) − (DE)
3	5F		MOV E, A	} $= 2^n (2^n - 1) E_{i-1}$
4	7C		MOV A, H	}
5	9A		SBB D	}
6	57		MOV D, A	}
7	69		MOV L, C	}
8	26		MVI H, 00	} (HL) ← V_i
9	00			}
A	CD		CALL SHFTN	Divide by 2^n
B	A0			(DE) ← $(2^n - 1) E_{i-1}$
C	82			
D	EB		XCHG	}
E	19		DAD D	} (DE) ← $V_i + (2^n - 1) E_{i-1}$
F	EB		XCHG	} $= 2^n E_i$
8 _ 0			CONTINUED	AT 8290
1				
2			ENTER (A) =	V_i (new value)
3			((HL)) =	n (filter constant)
4			((HL) + 1)	} $2^n E_{i-1}$
5			((HL) + 2)	}
6			RETURN	
7			((HL) + 3) =	(A) = (H) = E_i
8				(L) = V_i

Figure 6-4d

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR	CODE	Instruction	Operand	Comment
8 29 0	E3	XTHL		(HL) ← Address $2^n E_i$
1	73	MOV	M, E	Store $2^n E_i$
2	23	INX	H	
3	72	MOV	M, D	
4	23	INX	H	Address E_i
5	1B	DCX	D	To round up only if
6	CD	CALL	SHFTN	fractional part > 1/2
7	A0			
8	82			
9	77	MOV	M, A	Store E_i
A	E1	POP	H	(L) ← V_i
B	67	MOV	H, A	(H) ← E_i
C	C1	POP	B	Restore B, C, D, E
D	D1	POP	D	
E	C9	RET		Exit
F	00	NOP		
8 2A 0	48	MOV	C, B	SHFTN (C) ← n
1	AF	XRA	A	Loop - clear carry
2	7A	MOV	A, D	
3	1F	RAR		Shift (DE) right
4	57	MOV	D, A	to divide by 2
5	7B	MOV	A, E	
6	1F	RAR		
7	5F	MOV	E, A	
8	0D	DCR	C	Loop n times
9	C2	JNZ	82A1	to divide by 2^n
A	A1			
B	82			
C	D0	RNC		Exit if LSB = 0
D	13	INX	D	Else round up
E	7B	MOV	A, E	(A) ← less
F	C9	RET		significant byte
8 0				
1				
2				
3				
4				
5				
6				
7				
8				

Figure 6-4e

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

TEMPERATURE LOOKUP AND DISPLAY

ADDR		CODE				
8 2B	0	E5	PUSH	H		
	1	C5	PUSH	B		
	2	F5	PUSH	PSW		
	3	21	LXI	H,	8310	
	4	10				
	5	83				
	6	46	MOV	B,	A	Copy starting value into B, E, D
	7	23	INX	H		
	8	5E	MOV	E,	M	
	9	23	INX	H		
	A	56	MOV	D,	M	
82B	B	23	INX	H		Loop - address count
	C	4E	MOV	C,	M	(C) ← repetitions
	D	23	INX	H		Address slope
82B	E	F5	PUSH	PSW		Loop - save A/D
	F	7E	MOV	A,	M	Add low byte of slope to low byte of temperature
8 2C	0	80	ADD	B		
	1	27	DAA			
	2	47	MOV	B,	A	
	3	23	INX	H		Address high byte of slope and add the second byte of temperature
	4	7E	MOV	A,	M	
	5	8B	ADC	E		
	6	27	DAA			
	7	5F	MOV	E,	A	
	8	2B	DCX	H		Address low byte
	9	3E	MVI	A,	00	Add carry to high byte of temperature
	A	00				
	B	8A	ADC	C		
	C	27	DAA			
	D	57	MOV	D,	A	
	E	F1	POP	PSW		(A) ← A/D input
	F	3C	INR	A		Increment input
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					Figure 6-4f

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		Instruction	Comment
8 2D	0	C	A	JZ 82DB	Exit when A/D
	1	D	B		input is incremented
	2	8	2		to zero
	3	0	D	DCR C	Decrement repetitions
	4	C	2	JNZ 82BE	Loop to add slope
	5	B	E		until all repetitions
	6	8	2		of this slope done
	7	2	3	INX H	Address high byte
	8	C	3	JMP 82BB	Loop to address
	9	B	B		next slope
	A	8	2		
82D	B	E	B	XCHG	Exit and display
	C	C	D	CALL DWORD	Display temperature
	D	D	1		
	E	0	2		
	F	F	1	POP PSW	
8 2E	0	C	D	CALL DBYTE	Display voltage
	1	9	5		(C)← voltage
	2	0	2		
	3	E	B	XCHG	(DE)← temperature
	4	2	E	MVI L, FA	Address whole
	5	F	A		degrees in display
	6	7	E	MOV A, M	
	7	F	6	ORI 80	Insert decimal point
	8	8	0		
	9	7	7	MOV M, A	
	A	7	9	MOV A, C	(A)← voltage
	B	C	1	POP B	
	C	E	1	POP H	
	D	C	9	RET	
	E				
	F				
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				Figure 6-4g

TABLE OF REPETITIONS AND SLOPES

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A D D R			CODE		
8	31	0	9	9	Starting temperature
		1	6	2	-023.701°C
		2	9	7	as 100's complement
		3	4	0	FF - C0
		4	0	5	0.405 °C/20mv
		5	0	4	
		6	1	0	BF - B0
		7	7	3	0.373°C/20mv
		8	0	3	
		9	1	0	AF - A0
		A	4	1	0.341 °C/20mv
		B	0	3	
		C	2	0	9F - 80
		D	2	4	0.324 °C/20mv
		E	0	3	
		F	1	0	7F - 70
8	32	0	3	5	0.335 °C/20 mv
		1	0	3	
		2	1	0	6F - 60
		3	5	5	0.355
		4	0	3	
		5	1	0	5F - 50
		6	9	4	0.394
		7	0	3	
		8	1	0	4F - 40
		9	5	8	0.458
		A	0	4	
		B	0	8	3F - 38
		C	3	2	0.532
		D	0	5	
		E	0	8	37 - 30
		F	0	6	0.606
8	33	0	0	6	
		1			
		2			
		3			
		4			
		5			
		6			
		7			
		8			Figure 6-4h

TABLE OF REPETITIONS AND SLOPES (continued)

A D D	R	CODE	
8	0		
833	1	04	2F - 2C
	2	81	0.681
	3	06	
	4	04	2B - 28
	5	43	0.743
	6	07	
	7	04	27 - 24
	8	20	0.820
	9	08	
	A	04	23 - 20
	B	18	0.918
	C	09	
	D	04	1F - 1C
	E	46	1.046
	F	10	
8 34	0	04	1B - 18
	1	14	1.214
	2	12	
	3	04	17 - 14
	4	52	1.452
	5	14	
	6	04	13 - 10
	7	03	1.803
	8	18	
	9	02	0F - 0E
	A	90	2.190
	B	21	
	C	02	0D - 0C
	D	24	2.524
	E	25	
	F	02	0B - 0A
8 35	0	89	3.089
	1	30	
	2		
	3		
	4		
	5		
	6		
	7		
	8		Figure 6-4i

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

This page intentionally left blank.

CODING SHEET

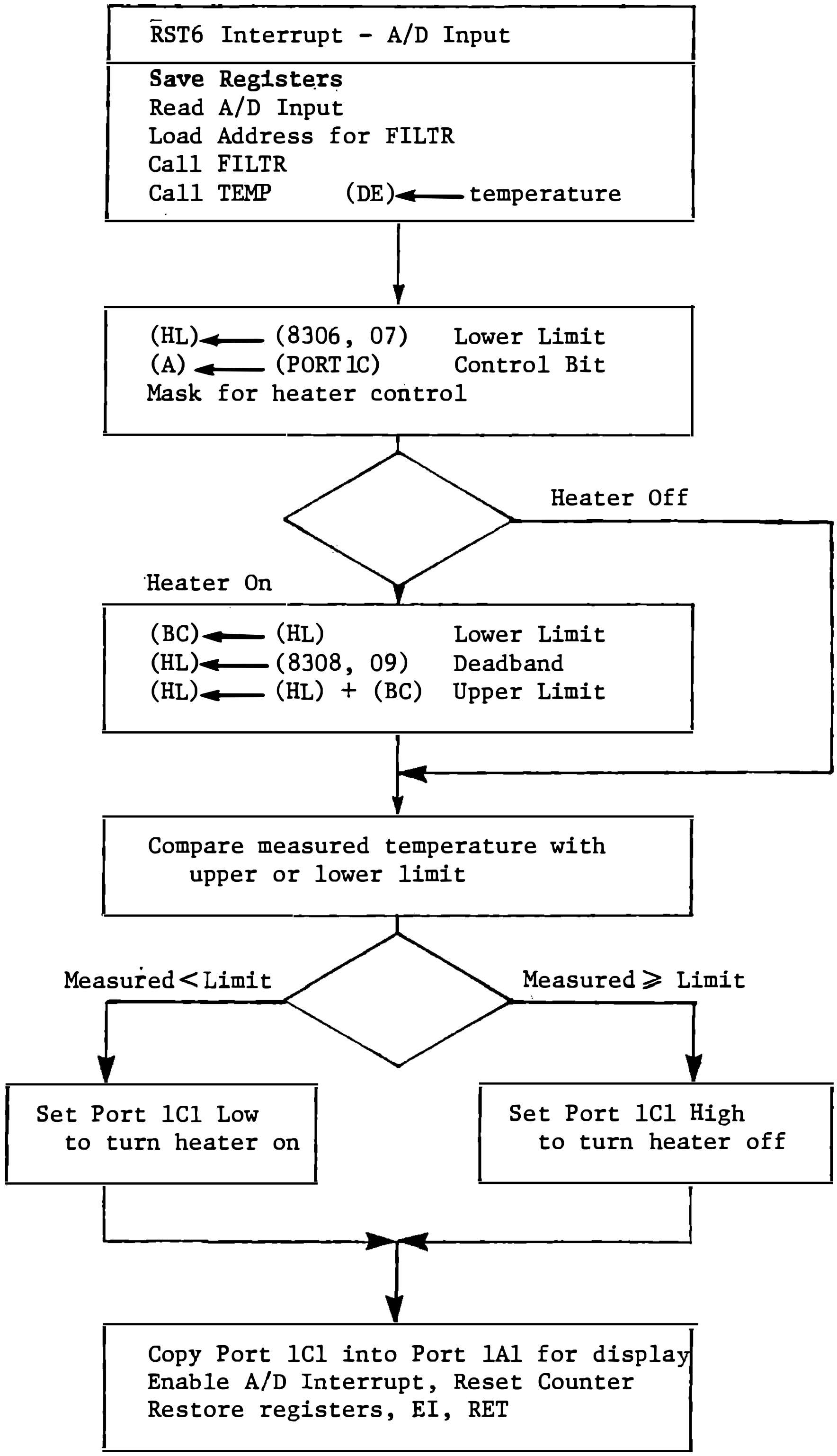
MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

TABLE OF REPETITIONS AND SLOPES (continued)

ADD	R	CODE		
8	0			
	1			
835	2	0	1	09
	3	7	3	3.573
	4	3	5	
	5	0	1	08
	6	6	5	4.065
	7	4	0	
	8	0	1	07
	9	9	7	4.697
	A	4	6	
	B	0	1	08
	C	3	7	5.537
	D	5	5	
	E	0	1	07
	F	0	0	6.700
836	0	6	7	
	1	F	F	06 125.974°
	2	0	0	upper limit for
	3	0	0	thermistor
	4			
	5			
	6			
	7			
	8			
	9			
	A			
	B			
	C			
	D			
	E			
	F			
8	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			

Figure 6-4j



Thermostat with Deadband - RST 6

Figure 6-5

6.1.2 On-Off Control With Deadband

EXERCISE

Modify the program of the preceeding Section (6.1.1) to provide a deadband. You can do this either by entering upper and lower temperature limits or by entering a value for the deadband which is always added to the lower temperature limit to obtain the upper limit. (The latter approach is used in the given solution.)

The main program is identical to that of figure 6-4a except that instead of storing data entered through ENTWD it calls a subroutine (STORE, at 82FO) which stores a deadband value at 8308,09 if the STEP key was used, or a lower temperature limit at 8306, 07 if the RUN key was used. NEXT stores the data entered as a lower temperature limit and also enters a default deadband of 2 degrees.

Figures 6-5 and 6-6 show interrupt service for the deadband thermostat. The present state of the control is tested to determine whether power is on or off. If it is off, the temperature is compared with the lower limit. If the power is on, the deadband is added to the lower limit before the comparison is made. Now power is turned on if the temperature is less than the selected limit, or off if the temperature is greater.

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR	CODE	Instruction	Comments
8200	3E	MVI A,82	Initialization
1	82		same as
2	D3	OUT CNT1	Figure 6-4a
3	07		
4	3E	MVI A,92	
5	92		
6	D3	OUT CNT2	
7	0F		
8	3E	MVI A,94	
9	94		
A	D3	OUT TIMCT	
B	17		
C	3E	MVI A,20	
D	20		
E	D3	OUT TIM2	
F	16		
8210	F3	DI	Data entry same
1	3E	MVI A,03	as Figure 6-4a
2	03		
3	D3	OUT PORT1C	
4	06		
5	D3	OUT PORT1A	
6	04		
7	CD	CALL ENTWD	
8	46		
9	03		
A	CD	CALL STORE	Store temperature
B	F0		limit at 8306,07
C	82		and deadband at 08,09
D	F7	RST 6	Enable A/D interrupt
E	DB	IN PORT0A	Test keyboard
F	00		
8220	3C	INR A	
1	CA	JZ 821E	Loop until
2	1E		key pressed
3	82		
4	C3	JMP 8210	Go to DI and
5	10		turn power off
6	82		and accept data
7			
8			

Figure 6-6a

THERMOSTAT - RST 6 INTERRUPT SERVICE

ADDR		CODE		Mnemonic	Operand	Comment
823	0	F	5	PUSH	PSW	
	1	E	5	PUSH	H	
	2	D	5	PUSH	D	
	3	C	5	PUSH	B	
	4	2	1	LXI	H, 8300	Address for FILTR
	5	0	0			
	6	8	3			
	7	D	B	IN	PORT1B	Read A/D
	8	0	5			
	9	C	D	CALL	FILTR	(A)← filtered
	A	7	0			A/D voltage
	B	8	2			
	C	C	D	CALL	TEMP	Calculate temperature
	D	B	0			Display
	E	8	2			(DE)← temperature
	F	2	A	LHLD	8306	(HL)← low limit
824	0	0	6			for temperature
	1	8	3			
	2	D	B	IN	PORT1C	(A)← controls
	3	0	6			
	4	E	6	ANI	02	Test for heater
	5	0	2			on/off
	6	C	2	JNZ	8250	Jump if off
	7	5	0			(Port 1C1 high)
	8	8	2			
	9	4	D	MOV	C, L	Heater on
	A	4	4	MOV	B, H	(BC)← temp request
	B	2	A	LHLD	8308	(HL)← deadband
	C	0	8			
	D	8	3			
	E	0	9	DAD	B	(HL)← upper limit
	F	0	0	NOP		
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					Figure 6-6b

MICROCOMPUTER TRAINING SYSTEM CODING SHEET — INTEGRATED COMPUTER SYSTEMS

ADDR	CODE	INSTRUCTION	COMMENT
8 25 0	7D	MOV A, L	Compare temperature
1	93	SUB E	with limit
2	7C	MOV A, H	Sets CY if
3	9A	SBB D	temperature high
4	3E	MVI A, 01	(A) ← 03 if high
5	01		← 02 if low
6	17	RAL	
7	D3	OUT CNT1	Set Port 1C1 low
8	07		(power on) if
9	DB	IN PORT1C	temperature low
A	06		
B	D3	OUT PORT1A	Display power control
C	04		
D	3E	MVI A, 07	Reenable and
E	07		reset A/D
F	D3	OUT CNT2	
8 26 0	0F		
1	C1	POP B	
2	D1	POP D	
3	E1	POP H	
4	F1	POP PSW	
5	FB	EI	
6	C9	RET	
7			
8		SUBROUTINES REQUIRED	
9		FILTR 8270-AF	
A		TEMP 82B0-EF	
B		DATA REQUIRED	
C		TEMP TABLE 8310-836F	
D			
E		8300 04	n
F		8301 00	Preload for
8 0		8302 00	FILTR
1		8303 00	
2			
3		8306, 07	Temperature limit
4		8308, 09	Deadband
5			Stored by STORE
6			(82F0-82FF)
7			
8			Figure 6-6c

SUBROUTINE STORE

ADDR	CODE		INSTRUCTION	COMMENT
8 2F 0	F	E	CPI RUN	Test command
1	1	4		
2	D	A	JC 82FC	If STEP jump to
3	F	C		store deadband
4	8	2		
5	2	2	SHLD 8306	Else store lower
6	0	6		temperature
7	8	3		
8	C	8	RZ	Exit if RUN
9	2	1	LXI H, 0020	If NEXT enter
A	2	0		default deadband
B	0	0		of 2.0°C
C	2	2	SHLD 8308	STEP (or NEXT)
D	0	8		store deadband
E	8	3		
F	C	9	RET	
8 0				
1				
2				
3			ENTER WITH	
4			(HL) = TEMP	ERATURE °C
5			(LOW	ER LIMIT)
6			OR D	EADBAND °C
7			(A) = COMM	AND
8			COMMANDS A	RE:
9			RUN - ST	ORE TEMP ONLY
A			STEP - ST	ORE DEADBAND
B			NEXT - ST	ORE TEMPERATURE
C			AN	D SET DEFAULT
D			DE	ADBAND=2.0°C
E				
F				
3 0				
1				
2				
3				
4				
5				
6				
7				
8				Figure 6-6d

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

6.1.3 Thermostat with Alarm Limits

OPTIONAL EXERCISE:

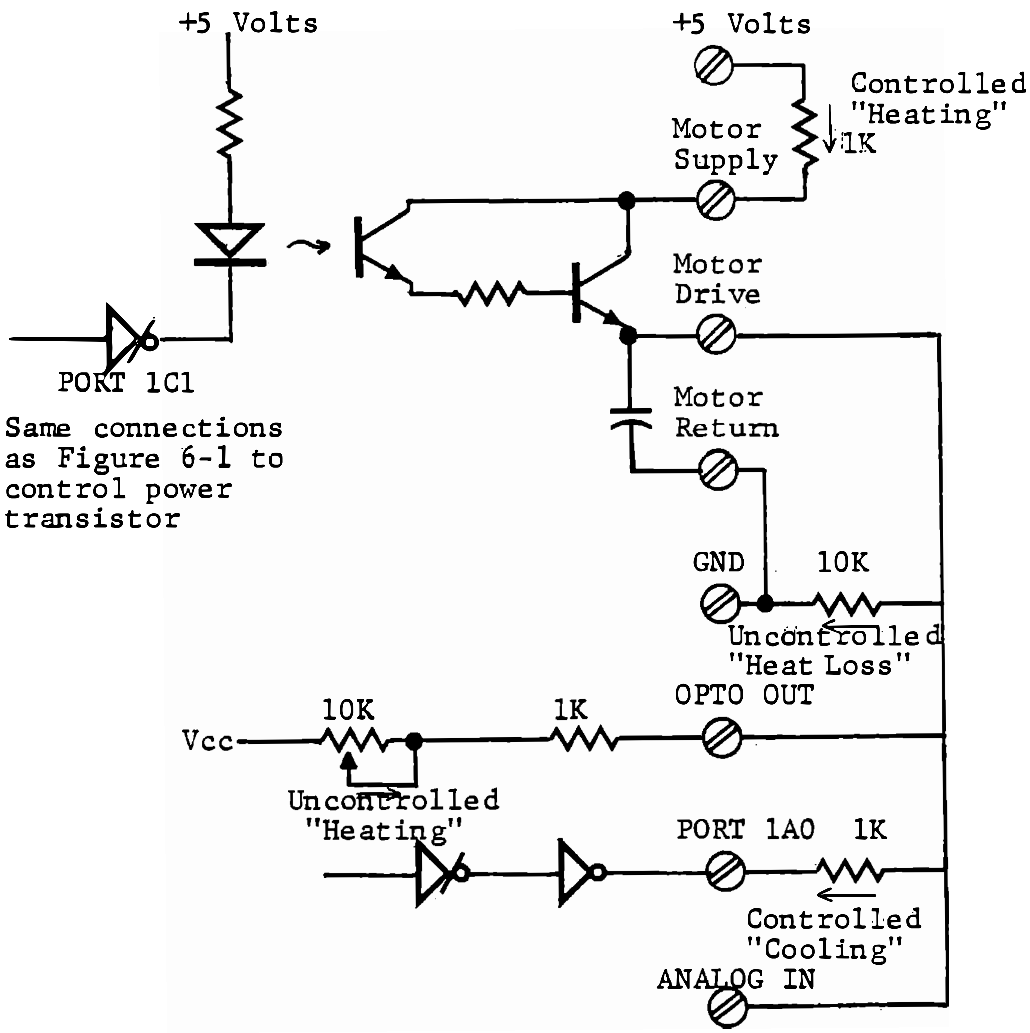
A very common requirement in process control systems is to test a temperature for certain limits and give an alarm if the limits are exceeded. This is likely to be used in conjunction with a temperature control, either thermostatic or proportional, so there may be as many as five limits:

Highest	- Danger to equipment or personnel. Probably indicates an equipment failure.
Next Highest	- Process is out of control. May result in degraded product.
Normal High	- Heater should be turned off to maintain normal process temperature.
Normal Low	- Heater should be turned on to maintain normal process temperature.
Lowest	- Process is out of control. May result in degraded product.

In a batch process, of course, the initial and final temperatures are likely to be lower than the lowest control temperature, so the alarm corresponding to that temperature should not be given until after the normal low temperature has been reached, nor after the process has been turned off.

There may also be time limits imposed on the process. In Section 4.2.9 an optional exercise was suggested in which a heater was controlled according to a schedule of times, with an upper limit for off-time and a lower limit for on-time. We will impose such limits here. In addition, we will place an upper limit for on-time, because if the heater stays on too long it may indicate a failure in the temperature sensor such that the highest temperature limit could be reached without being detected.

Develop your own flow charts and program for this problem. Select limits that can be reached by the heater you are using, and force the alarm conditions to occur by connecting MOTOR CONTROL - to ground so that the heater will stay on.



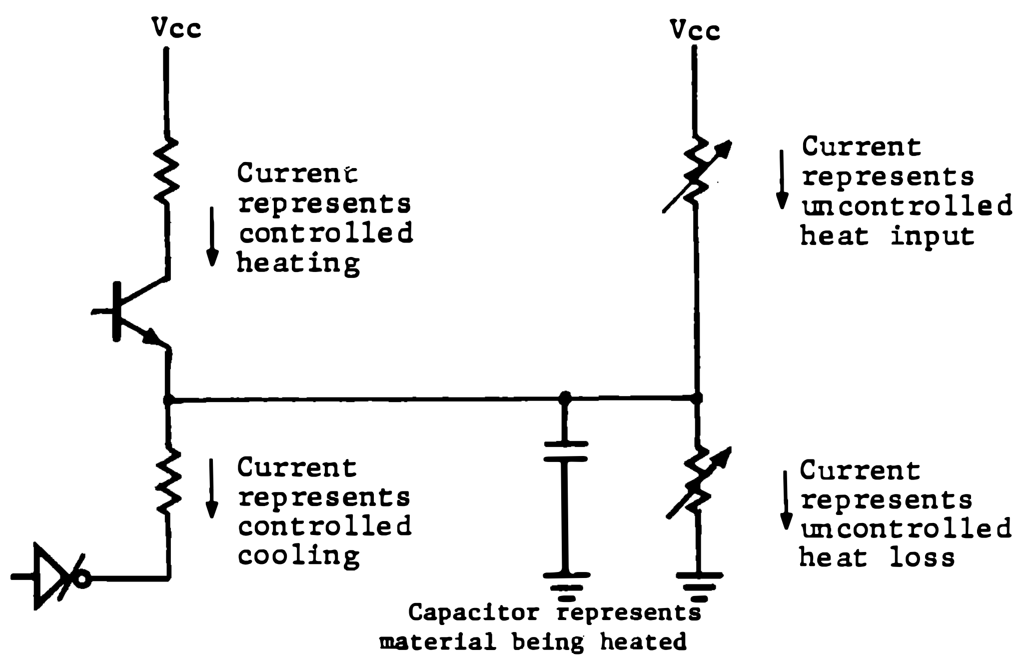
Circuit Connections for Simulation

Figure 6-7

6.1.4 Two-Way Control

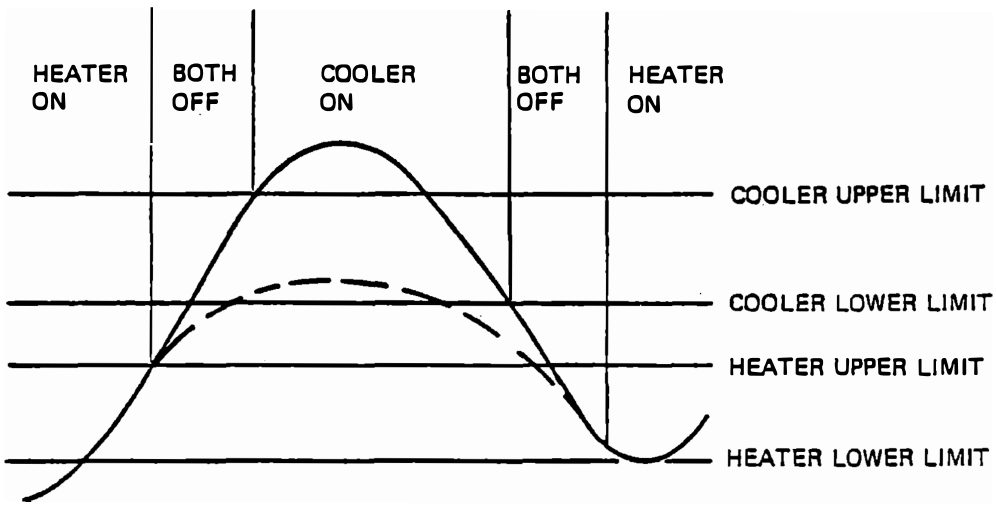
OPTIONAL EXERCISE:

In the simple thermostat problem it was assumed that only heating would be needed to maintain a required temperature; incidental heat loss would provide for cooling. In many temperature control problems that is not the case - both heating and cooling are required. The building with both furnace and air conditioner is the obvious example, but chemical processes also may require both. We cannot readily provide a realistic exercise with a heater and cooler, but the functions are easily simulated by charging and discharging a capacitor. Figure 6-7 shows connections to the interface board, and Figure 6-8 shows a circuit diagram for such a simulation.

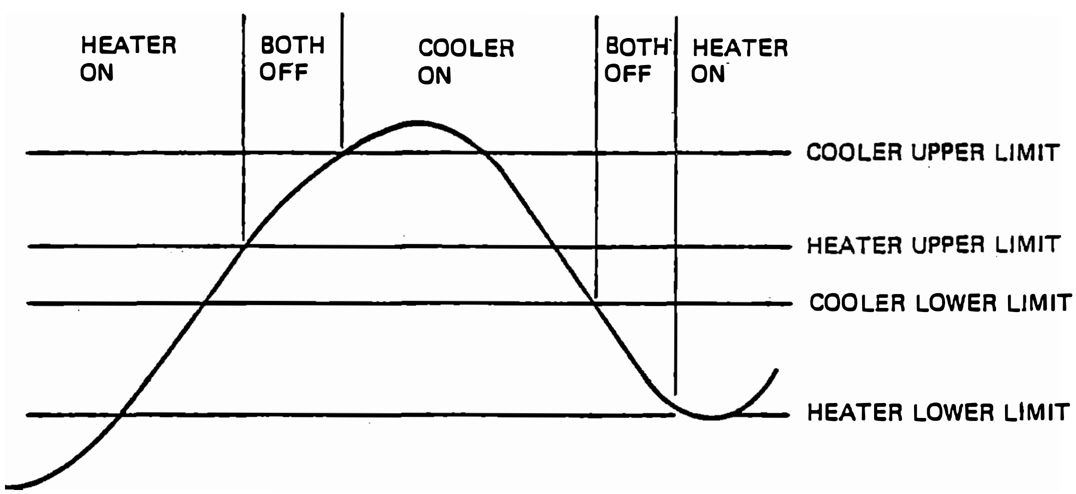


Heating and Cooling Simulation

Figure 6-8



Normal Inner Limits



Inverted Inner Limits

Heating and Cooling Limits

Figure 6-9

We use the SENSE pot to simulate a variable heat input that cannot be controlled by the program and a 10K resistor to simulate heat loss. (An external 10K pot can be used instead, but is not necessary). The power transistor and Port 1A0, each with a series 1K resistor, represent the controlled heating and cooling.

Figure 6-9 depicts the control process. In Figure 6-9a a normal situation is shown. As the process temperature rises with the heater on, it crosses the heater lower limit and reaches the heater upper limit, where the heater is turned off. If either thermal inertia or uncontrolled heating causes the temperature to reach the cooler upper limit the cooler is turned on to return the process toward the desired temperature, and it runs until the temperature drops to the cooler lower limit. if the temperature does not reach the cooler pper limit the cooler remains off (dashed line). Obviously in the case of temperature control of a building (rather than a chemical process) there are many days when only the heater is turned on or only the cooler is turned on.

In process control however, it is often the case that the cooler involves greater thermal inertia than the heater. Heating can usually begin very quickly after a control signal and also stop very quickly. Cooling is likely to involve significant delay, since cold water must be pumped through pipes that are full of water heated by the process, and when the cooler is turned off the remaining cold water in the pipes will still absorb heat. In such a case the heater upper limit may be above the cooler lower limit, as shown in Figure 6-9b. In an extreme case, both cooler limits might be between the two heater limits. This implies that sometimes both the heater and cooler would be in operation at the same time. The conclusion is that independent upper and lower limits should be provided for both the heater and the cooler. (These may of course be expressed as lower limit and deadband).

Clearly the control algorithm is essentially the same as for the case of a heater (or cooler) only, but must be processed twice for each temperature measurement. Once again we leave program development to the student.

6.2 PROPORTIONAL Vs. INTEGRAL CONTROL

On - off control is unsuitable for many purposes. Imagine steering a car with a three position switch allowing only left turn, straight ahead, or right turn. You might make your way along a sufficiently broad highway, but the turn radius suitable for high speed driving would made parking very difficult. Proportional control is a method of applying a varying control force depending on the magnitude of the adjustment required.

In steering a car along a straight road you may be able to take your hands off the steering wheel for several seconds, and continue in a straight line. Soon, however, the car will drift off the track and a gentle touch on the wheel will be needed to return to the straight line. You have observed an "error signal" and applied a "control force" to correct the error. When the error becomes zero you again relax the control force.

If a bump in the road caused the car to swing sharply you would apply a more vigorous control force through the steering wheel. This is the essence of proportional control: a larger error results in a stronger restoring force.

When you reach an intersection and want to turn, you have changed the "setpoint". Suddenly a large error signal appears, because now the intended direction (i.e. setpoint) is pointed 90 degrees away from your car's current direction. You apply a <u>large</u> control force to correct for this <u>large</u> error signal, until once again the error signal reaches zero. Thus the control force is in direct relation or

<u>proportional</u> to the error signal. A proportional control system can be described by:

Control Force = Gain (Desired Value - Measured Value).

The difference between the desired value and the measured value is called the error signal. It may be temperature, distance, angle, speed, voltage, or any of a host of other continuous variables. The control force might be electric current or gas flow to a heater; voltage, current or frequency to an electric motor; hydraulic pressure, etcetera. Note that gain is usually not a dimensionless ratio in this abstract form. For instance if a measurement in degrees is to give a control force in pounds, gain would have the dimensions "pounds per degree". In many cases with computer control both the measured value and the control force may appear as voltage analogs of the real variables.

In some processes it is not enough to provide only a correcting force. It may be necessary to provide some driving force all the time. For instance, we might operate a heater at some steady current, but increase or decrease that current in response to a temperature measurement. Then the control force would be:

(a) $F = G(E) + S$

where F = control force

G = gain

E = error signal as above

S = steady state force

Now if the system is well understood and conditions are constant, corrective changes are made in the control force only when some disturbance causes an error signal.

If you were driving a camper or truck on a windy highway you would not take your hands off the steering wheel. Some force is needed all the time to keep the truck going in a straight line. This force must be increased or decreased momentarily to compensate for gusts or bumps. It is not a constant force, however, but must be adjusted if the wind force changes. Systems of this kind, that require a continuous control force to be adjusted for both momentary and long term changes, are more common than those where the control force is usually zero or constant.

The first kind of control system, with no steady state force, is called "proportional control" or "pure proportional control". It is well suited to processes that are subject to temporary disturbances or changes in setpoint, but will otherwise do what is intended by themselves. A simple autopilot in an airplane or boat detects any error between the compass and the setpoint and applies a control force to reduce the error to zero. Until a wind gust or a wave disturbs it, the vessel will go in a straight line.

CLOSED LOOP CONTROL

When some control force is required all the time, as on our windswept highway, pure proportional control is not satisfactory because it will deliver a control force only when there is an error signal - as our camper goes off the road. To provide the continuously adjustable control, we would use "integral control". This system is named from its control equation, which is described below. Integral control can provide a steady state control force that will maintain a zero error signal when there are no disturbances. It will correct for disturbances such as the wind gusts, and it will adjust the steady state force in response to changing conditions.

Pure integral control also has a weakness. It is inclined to overcorrect for momentary disturbances because it cannot immediately distinguish between (for instance) a gust of wind and a change in the wind force. When both momentary and long term changes in the conditions will occur it is best to use a combination of proportional and integal control.

In the remainder of this chapter we will develop a "proportional plus integral" control system, working up by steps from an open loop system of pulse width modulation, to a pure integral control system, and finally to the combined system. Much of the description, the program, and the experiments will be devoted to observing the behavior of the system. First we will develop the control equation for an integral control system to see why it is so named.

In pure integral control we adjust the control force whenever an error signal occurs, and maintain that new control force until another error signal is measured.

$$\text{(b)} \qquad F = GE_1 + GE_2 + GE_3 \text{ ----}$$

As long as repeated measurements indicate a positive error (desired value greater than measured value) the control force will be increased. Eventually it will be enough to make the error negative, and the force will be reduced. After a time, if the control system is stable, the force will reach just the right value to maintain a zero error under the existing conditions. The sum of a series of measurements of a variable is the integral of that variable, so if equation (b) is expressed as:

$$F_i = GE_i + F_{i-1}$$

or

$$\text{(c)} \qquad F_i = G \int E$$

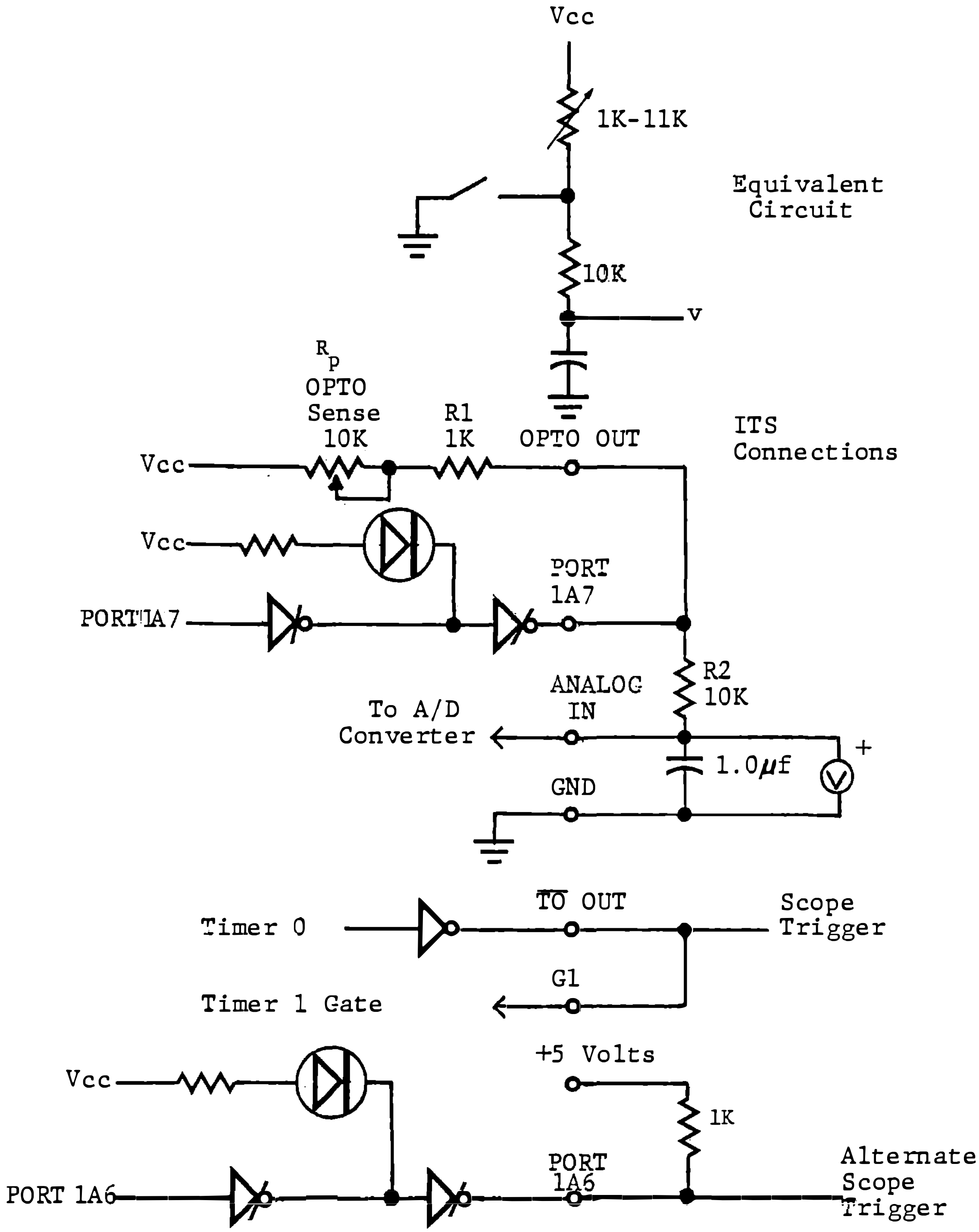
the derivation of the name "integral control" is apparent.

In the following experiments we require both analog output for the Control Force and analog input to determine the error signal. Since the interface board uses the D/A converter for A/D conversion we will use pulse width modulation to generate a voltage on a capacitor for the digital to analog output and measure that voltage with the automatic A/D input. Clearly, in any of these exercises the output voltage or the pulses could be driving some other load such as a heater or a motor, and the voltage input could be obtained from a thermistor for heat, a linear potentiometer for position, or some other sensor with a voltage output. It is important here that control is exercised so as to force a measurable end result to be equal to a desired value.

In the first exercise we will develop some of the program modules necessary for the Pulse Width Modulation Control System: Initialization, Data Entry via Keyboard, Digital Data Filter, Digital Voltmeter Display, Interrupt Service for PWM and the Main Loop. However, we will not yet develop the modules responsible for Error Signal generation until section 6.2.4. This will allow the opportunity to experiment with an Open Loop PWM Control System since the system cannot perform error detection and correction. We can thus demonstrate how the system output is dependent on external factors (in this case, the setting of the OPTO SENSE pot). In the second exercise (section 6.2.3) we will observe the system's response time characteristics via data logging, and an oscilloscope if available.

In the third exercise (section 6.2.4) we will develop the program modules necessary to close the loop: Error Signal Calculation and Display, Control Force Calculation and Modification. We will also enable additional command keys to select either Open Loop or Closed Loop control. We can thus demonstrate how Closed Loop Control eliminates the system output's dependence on external factors.

Then, in section 6.2.6, we will experiment with various aspects of Closed Loop Control. Access to an inexpensive oscilloscope will be helpful.



Connections for PWM Experiment

Figure 6-10

6.2.1 Voltage Control Circuit

The circuit of Figure 6-10 is similar to that used in Section 4.2 for pulse width modulation, except that a capacitor is included to average the PWM signal. It is charged through the OPTO SENSE pot to introduce an external variable not controllable by the program. If this were to drive a load, an external amplifier would be needed, but we will not be concerned with that here. The capacitor voltage is measured both by the A/D converter and the voltmeter.

When Port 1A7 is high (output turned off) the capacitor is charged through the SENSE pot (Rp) the internal resistor R1, and the external resistor R2. It charges toward the Vcc (5 volt) supply, with a time constant of RsC,

where $$Rs = Rp + R1 + R2.$$

When Port 1A7 is set low, the capacitor discharges through R2, with a time constant (R2)C. With the resistances shown the charging time constant ranges from 11 to 21 milliseconds, according to the pot setting; the discharge time constant is ten milliseconds. If we operate pulse width modulation with a cycle time of a few hundred microseconds the capacitor voltage will change only slightly during each cycle.

Using a linear approximation, the voltage increases during charging by:

(a) $$\Delta v_c = \frac{t_c}{R_s C}(V_c - v)$$

where t_c = charging time

$R_s C$ = charging time constant

V_c = supply voltage (5 volts)

v = capacitor voltage

When Port 1A7 is low the capacitor discharges by:

(b) $$\Delta v_d = \frac{t_d}{R_2 C}(v - V_g)$$

where t_d = discharging time

$R_2 C$ = discharging time constant

V_g = voltage drop across the open collector buffer (0.2 volts)

Provided the pulse sequence is maintained for a time greater than several time constants, a steady state will be reached where the charging and discharging are equal in each cycle. Then:

(c) $$\frac{t_c}{R_s C}(V_c - v) = \frac{t_d}{R_2 C}(v - V_g)$$

This equation can be solved for the steady state average voltage:

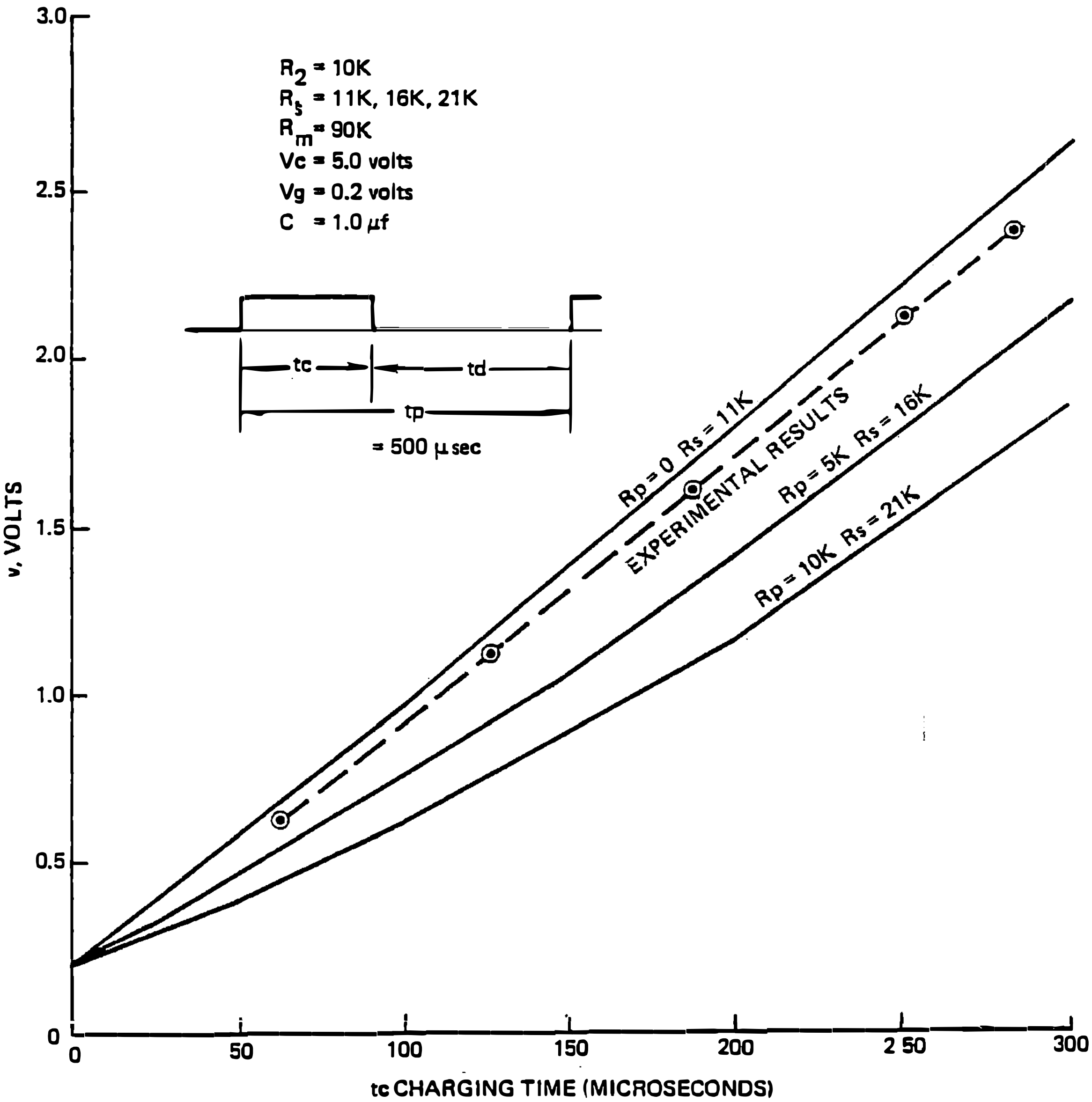
(d) $$v = \frac{\dfrac{t_c}{R_s C} V_c + \dfrac{t_d}{R_2 C} V_g}{\dfrac{t_c}{R_s C} + \dfrac{t_d}{R_2 C}}$$

The value of the capacitor factors out of equation (d), showing that the steady state average voltage is independent of the capacitor. The equations above neglect the voltmeter, which drains a little current to ground. Accounting for the voltmeter resistance (Rm) leads to equation (e):

$$\text{(e)} \quad v = \frac{\dfrac{t_c V_c}{R_s} \qquad \dfrac{t_d V_g}{R_2}}{t_c \; \dfrac{1}{R_s} + \dfrac{1}{R_m} + t_d \; \dfrac{1}{R_2} + \dfrac{1}{R_m}}$$

This equation is plotted in Figure 6-11 for a fixed total period of 500 microseconds and varying charging time, for three different values of potentiometer resistance. Note that with zero resistance in the potentiometer the voltage is nearly linear with charging time. The voltage is only moderately sensitive to the pot setting because of the fairly large value (10K) of R2, but the pot does cause a substantial departure from linearity. Figure 6-11 also shows experimental results generated with the program to be developed.

$$v = \frac{\frac{tc}{R_sC} Vc + \frac{td}{R_2C} Vg}{tc\left(\frac{1}{R_sC} + \frac{1}{R_mC}\right) + td\left(\frac{1}{R_2C} + \frac{1}{R_mC}\right)}$$



PWM Voltage - Fixed Period

Figure 6-11

With constant total period tp = tc + td we can solve equation (e) for the ratio of charging time to total period required for any desired voltage.

$$\text{(f)}\qquad \frac{t_c}{t_p} = \frac{v - \dfrac{R_m}{R_2 + R_m} V_g}{\left(\dfrac{R_m}{R_2 R_m}\right)\left[V_c \dfrac{R_2}{R_s} - V_g + \left(1 - \dfrac{R_2}{R_s}\right) v\right]}$$

If the pot is set to zero resistance we have the following values:

$$\frac{R_m}{R_2 + R_m} = \frac{90K}{10K + 90K} = 0.90$$

$$\frac{R_2}{R_s} = \frac{10K}{10K + 1K} = 0.91$$

$$V_c = 5.0$$

$$Vg = 0.2$$

Then:

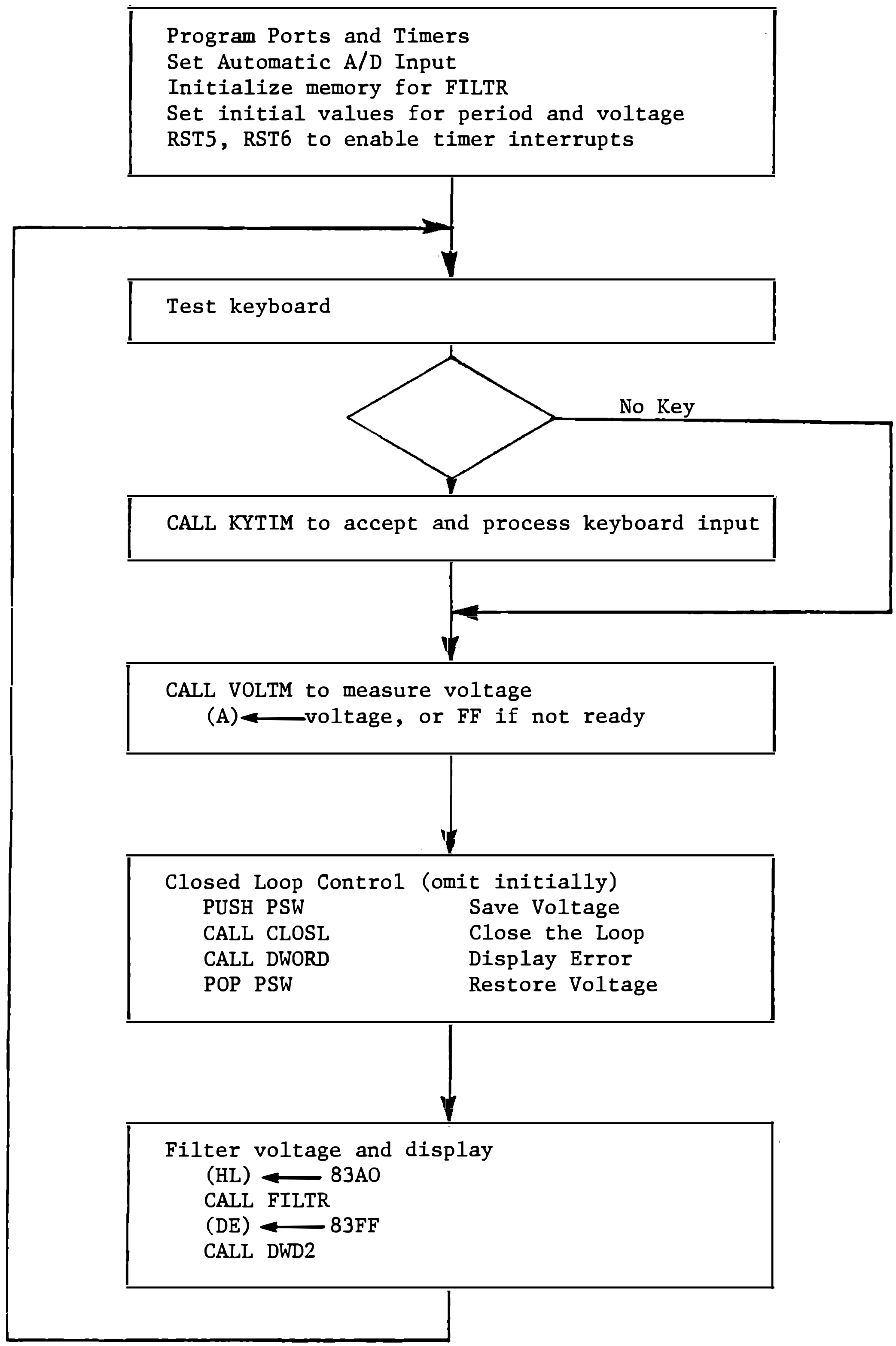
$$\text{(g)}\qquad \frac{t_c}{t_p} = \frac{v - 0.18}{3.91 + 0.08v}$$

Since the .08v term in the denominator is quite small the relationship is nearly linear when the pot is set to zero, as shown in Figure 6-11. We will use this linear relationship in our pulse width modulation scheme.

This page intentionally left blank.

Timer 0 will be loaded with a count for the total period; at each interrupt from this counter Port 1A7 will be set high to start charging. Timer 1 will be loaded with a count for charging time, derived from the desired voltage. Timer 1 will be started when charging starts, and its interrupt will set Port 1A7 low to stop charging.

We will enter a desired voltage in hexadecimal with the least significant bit representing 10 millivolts, just as the A/D converter represents a voltage. Subtract 12 (representing 0.18 volts). This value will be used to determine the charging time in microseconds, so double it to account for two system clocks per microsecond, and load the result to Timer 1. The total period, loaded to Timer 0, should correspond to 3.91 volts or 391 microseconds. This is given by a hexadecimal value of 30E (= decimal 782). This value must be adjusted to compensate for resistor and voltage tolerances and the error caused by neglecting .08 v. The program provides for keyboard input of the total period so that the adjustment can be made while the program is running.



PWM Voltage Control - Main Loop

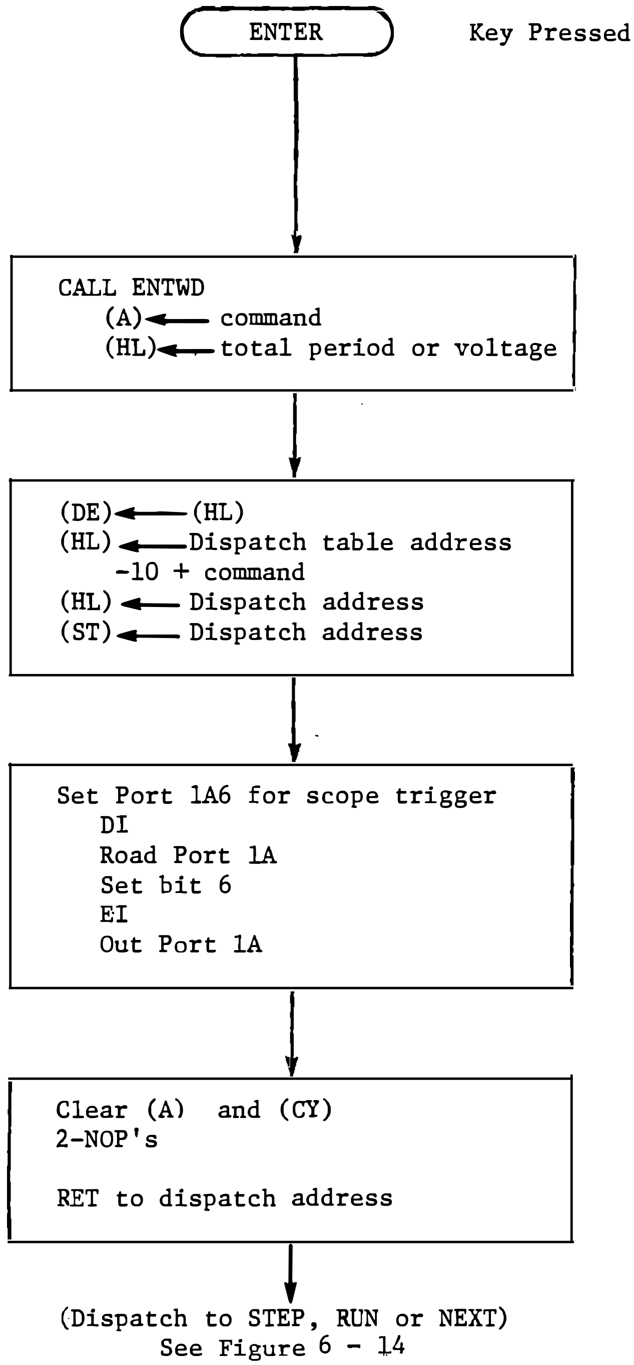
Figure 6-12

6.2.2 Voltage Control by PWM

EXERCISE:

In the program of Figure 6-12 we repetitively measure and display the voltage at the analog input, and test the keyboard for data entry. The voltage is determined by pulse width modulation under interrupt control; this is discussed in Section 6.2.2.4. The main loop in Figure 6-12 includes a call to a closed loop control subroutine that will be developed in Section 6.2.4. For developing the open loop program you can enter eight NOP instructions. The following sections discuss the three subroutines KYTIM, VOLTM, and FILTR.

As always, it is recommended that you study the problem and develop your own solution for each program module. Follow the flow charts given here closely, because a number of revisions will be made as we develop the closed loop system. With various modifications and additions we will use this program through the remainder of Chapter 6.



PWM Voltage - Subroutine KYTIM

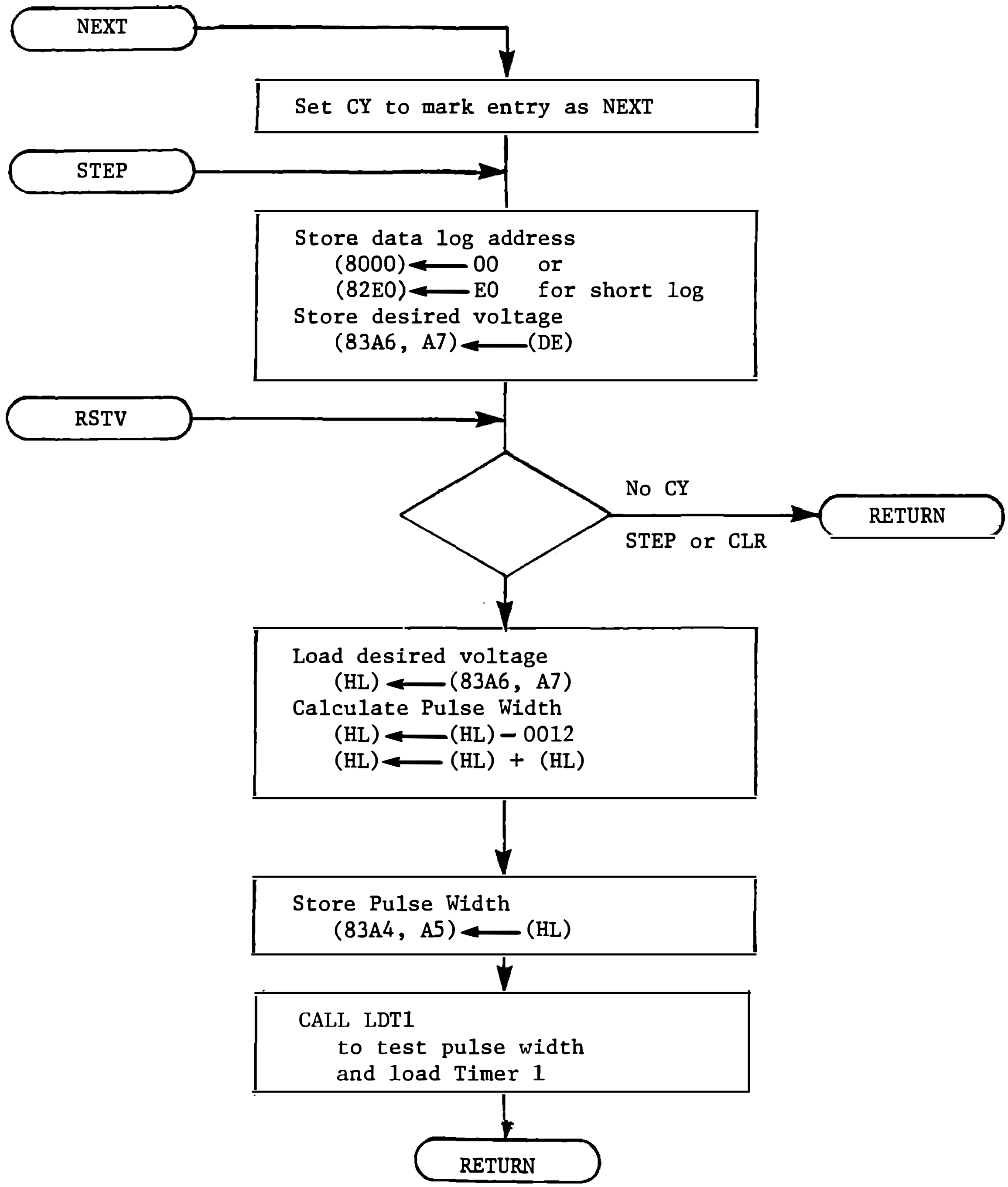
Figure 6-13

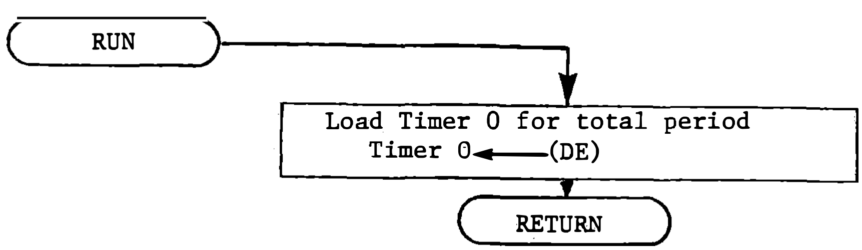
6.2.2.1 Data Entry Subroutine KYTIM

Subroutine KYTIM (Figure 6-13) is called for data entry when the main loop detects a key depression. KYTIM calls ENTWD (0346) to accept up to four hex keys (returned in HL) and a command (returned in A).

A dispatch table is used for distinguishing among the keys. Although it is not necessary to treat all keys differently at present, we will add some functions later. One of the main purposes of this exercise is to observe the effects of different control algorithms. The program includes a data logging facility, and also provides for oscilliscope observations. Immediately after looking up the dispatch address and before jumping to the process, Port 1A6 is switched high to give a scope trigger for use in observing the response time after a new voltage is keyed in. This output is available at a tie block, but must be pulled up through a resistor since it is an open collector buffer. Using any undefined key will generate the scope trigger without changing the pulse width or total period; the ADDR key will be reserved for this function.

Since both KYTIM and interrupt service alter data at Port 1A, interrupts should be disabled while KYTIM is manipulating the port. To maintain good control, however, the time during which interrupts are disabled should be as short as possible.





PWM KYTIM - Set Pulse Widths

Figure 6-14

During debugging you may want to place a breakpoint at the RET instruction that dispatches to the command processing module. Recall that the monitor interrupt cannot occur until an instruction has been excuted with two or more machine cycles after EI. To overcome this limitation, place the EI before OUT PORT1A, which requires three machine cycles. No interrupt can occur until the OUT instruction has been executed; then either of the hardware interupts or the monitor interrupt can occur.

The key processing modules are described below and shown in Figure 6-14.

If the command is NEXT or STEP the input data represents a desired voltage. Before storing and processing this value, memory location 8000 is cleared to initiate a new data log. The desired voltage is then stored at (83A6, A7). For the data logging function it is important that these steps occur in the sequence indicated.

The STEP key calls for the same processing as NEXT up to this point, but the following steps are omitted by a conditional return if carry is clear, indicating that the command was STEP. The STEP key is useless in the open loop system but is needed for closed loop control. The RSTV ("restore voltage") entry also reaches this conditional return. In a later modification of the program the BRK and CLR keys will jump to this entry after performing their other functions, and the conditional return will be executed for CLR but not for BRK.

Now the required pulse width is calculated. This is done in response to NEXT, and it will be done in response to BRK. Because of this alternate entry 'the desired voltage is loaded into (HL) from the location (83A6, A7) where NEXT or STEP has stored it. Subtract 0012 to allow for the zero offset voltage Vg (0.18 volt) and double the result to give two clocks (one microsecond) for each ten millivolts. Then the calculated pulse width is stored at (83A4, A5). Finally, subroutine LDT1 is called to load Timer 1 with the new pulse width.

6 2.2.2 Subroutine LDT1

Timer 1 is to be loaded with a calculated pulse width both in response to the NEXT key and later under closed loop control. It is entered with a pulse width as two bytes in (HL).

Before actually loading Timer 1 we check that a legitimate pulse width has been entered. The system will go out of control if a very long time is entered to Timer 1. This could occur in the closed loop system if a negative width were calculated, or in the open loop system if you request a voltage less than 12. The timer treats 0000 as a maximum delay, so we also test for this value. One way of making the test would be:

```
PU SHH       Save original value
DCX H        Force 0000 to FFFF
DAD H        High bit to carry
POP H        Restore original value
RC           Exit if zero or negative
```

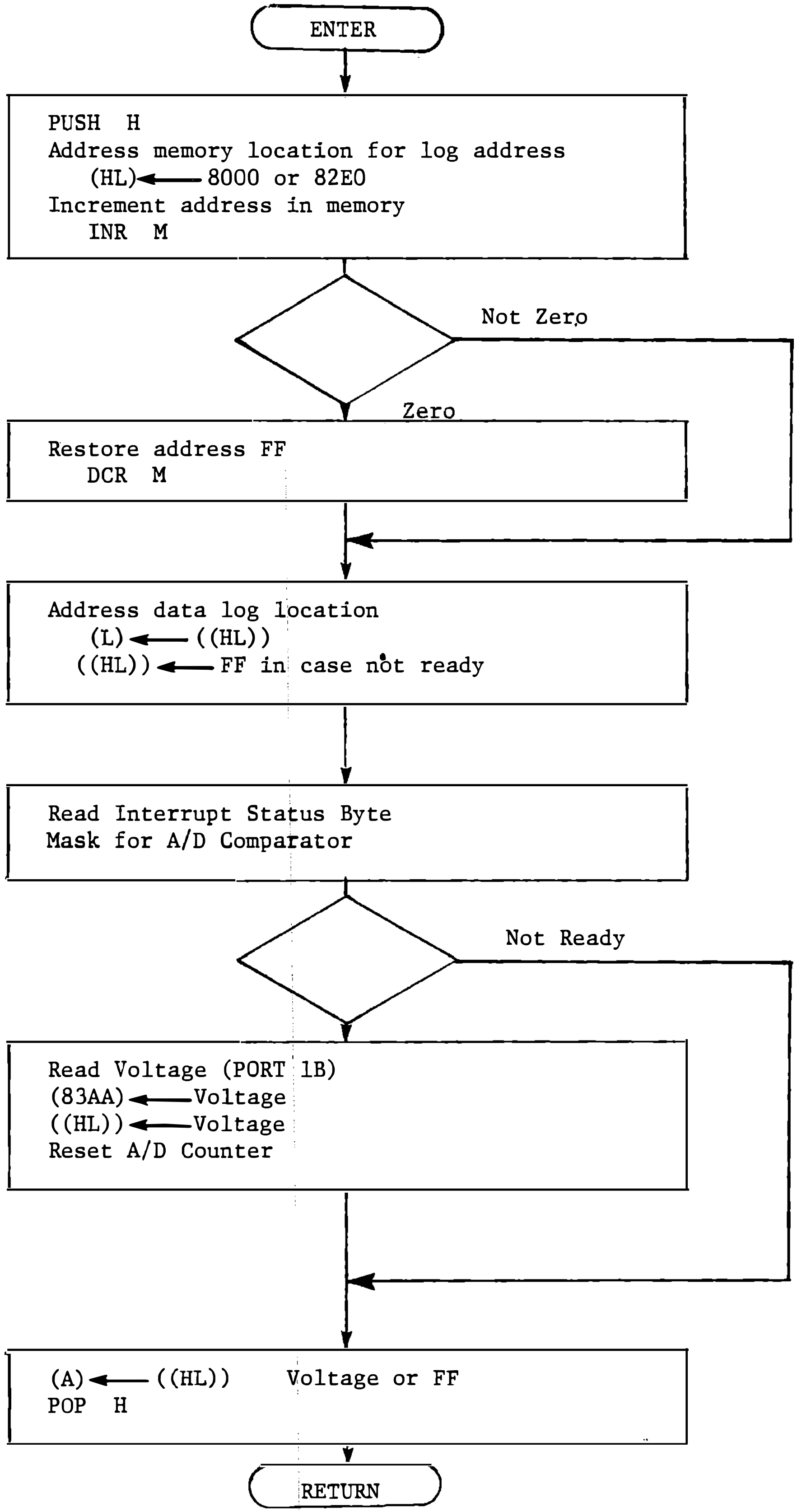
The above method has the virtue of changing only the carry flag. Another method is:

```
DCX H        Force 0000 to FFFF
MOV A,H      Set all flags according
ORA A            to content of H.
INX H        Restore original value
RM           Exit if zero or negative.
```

This method, which the author has used, is faster and avoids using any stack area.

Provided that the test indicates a legitimate pulse width, Timer 1 is loaded with the data. This sets the charging pulse width. The RUN key copies the input data from ENTWD to Timer 0, to set the total period.

This page intentionally left blank.



Logging Voltmeter

Figure 6-15

6.2.2.3 Voltmeter Subroutine VOLTM

The automatic A/D converter is used to measure the capacitor voltage. To permit observing response times without additional instruments, the voltmeter subroutine will log the voltage each time it is called. The following initialization steps are necessary:

Program Port 1B for input.

Set Port 1C1 high to enable counting.

Program and load Timer 2 to divide the system clock by 8.

Store an initial address for the data log.

Although in this program VOLTM is called from the main loop, and returns the measured voltage in register A, it is written to permit its use as an interrupt service routine. It stores the measured voltage in a fixed memory location and also logs the voltage at an address obtained from memory. Figure 6-15 shows the subroutine.

Memory page 80xx is allocated to the data log. Address 8000 stores the low byte of the last address used in the log . This location is loaded with 00 each time a new log is to be started. VOLTM increments the content of location 8000 to count the number of times it has been called, and uses the new value as the low byte of the log address. We are interested in the behavior of the system for a relatively brief period ater a change in the system setpoint (the voltage request), and do not want to destroy those data with later measurements. Therefore, the logging address is incremented only up to FF, and thereafter remains fixed.

CLOSED LOOP CONTROL

Since VOLTM is called by the main program, not by an A/D interrupt, it is possible that a voltage conversion will not be ready. If the input voltage is greater than 2.55 volts it cannot be measured by the A/D converter because the comparator will always see the input greater than the D/A output. The comparator signal appears in bit 3 of the interrupt status byte. This is tested by IN PORT2B, ANI 08; if the bit is low the subroutine returns a value of FF with the zero flag set. If the comparator signal is high the voltage is read by IN PORT1B, and will be returned with the zero flag not set. Before return, however, the A/D counter must be reset by disabling the A/D interrupt, to start a new conversion.

Note that this subroutine will also return FF and Zero set if the voltage is less than 2.55 volts but insufficient time for the conversion is allowed between calls. In the program being developed, however, enough time is taken by the main loop and subroutines to ensure a valid conversion if the voltage is less than 2.55 volts.

6.2.2.4 Using FILTR

The voltage generated by pulse width modulation is averaged, or filtered, by the capacitor. It fluctuates by about 50 millivolts, so the voltage measured will vary in the less significant bits. To obtain a valid eight bit measurement we will use subroutine FILTR to provide additional filtering for display of the output.

FILTR is called with the raw (unfiltered) voltage in (A) and a memory address (83A0) in (HL). That memory location must be loaded with 1,2,3 or 4 during initialization, and the following two bytes must be cleared initially.

FILTR returns the input value (the raw voltage) in L, and the filtered voltage in both A and H. For display, load DE with 83FF and call DWD2 (02D4). This will display the data at the right hand side of the display, leaving your last keyed in data displayed at the left.

FILTR and DWD2 take most of the time needed for the A/D conversion. If you should choose not to use them you must provide about a one millisecond delay by other means, such as a call to the monitor subroutine DELAY (0236).

6.2.2.5 Timer Operation

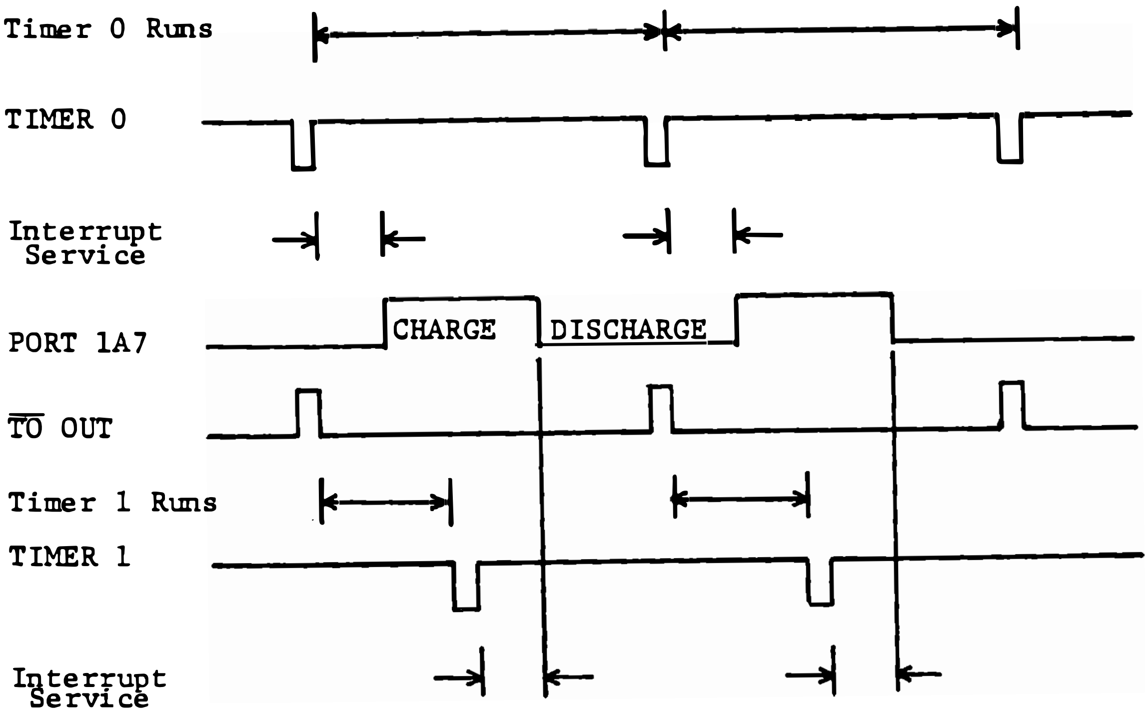
Timer 0 is used to define the total period. It operates in mode 2 so that RST5 interrupts occur at precisely repeated intervals according to the value keyed in with RUN, and this period is repeated until a new value is entered. Interrupt service for RST5 sets Port 1A7 high to start charging the capacitor. It also sets 1A5 low to clear the scope trigger.

Timer 1 controls the charging time. While it is counting, Port 1A7 will be high so that charging can occur. At its terminal count, RST6 interrupt occurs and Port 1A7 is set low to start discharging the capacitor.

Timer 1 is used in mode 5, "Hardware Triggered Strobe". This mode of the 8253 interval timer was briefly described in Chapter 3 but has not been used in any previous exercise. In this mode the timer can be preloaded with a count value, and when a rising edge signal occurs at its gate input it starts counting. The timer output is normally high; it goes low for one clock period at the zero count and rises again to create an interrupt one clock time later. The timer then waits for another rising edge at its gate input, obtained from Timer 0 output.

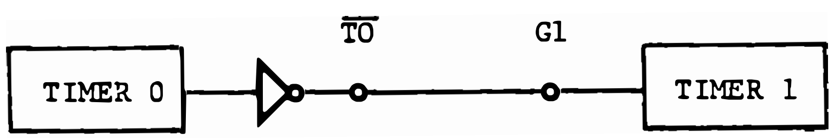
Figure 6-16 shows the timing relationship of Timer 0, Timer 1, and Port 1A7. Figure 6-17 shows the interrupt service routines. Note that both service routines should take the same length of time from the interrupt to the switching of Port 1A7.

Both mode 2 and mode 5 of the interval timer allow the timer to be laoded at any time. If the timer is counting, the present period is completed before the new time value is loaded. Therefore we can load these timers from the KYTIM subroutine without regard for their present states.



Timer 0 in Mode 2

Timer 1 in Mode 5, triggered by Timer 0



PWM Timer Operation

Figure 6-16

This page intentionally left blank.

RST 5 Interrupt - Timer 0

Set Port 1A7 high to start charging
Set Port 1A6 low to clear scope trigger
Reenable Timer 0 Interrupt

Exit

RST6 Interrupt - Timer 1

Set port 1A7 low to start discharging

Reenable Timer 1 Interrupt

Exit

Note **Ports 1A0 - 1A5 are not used in the system, so these may be set high or low as convenient.**

PWM Interrupt Service

Figure 6-17

6.2.2.6 PWM Memory Allocation

You should develop a detailed flow diagram for the main program and write your own programs for MAIN, KYTIM, VOLTM, and the interrupt service routines. Subroutine FILTR was developed in Section 5.5. Remember to provide its initialization and to load (HL) with the required memory address. The following memory assignments are used in the given solution:

8200 - 8227	Main - Initialize
8228 - 824F	Interrupt Service
8250 - 826F	Subroutine VOLTM
8270 - 82AF	Subroutine FILTR
82B0 - 82CF	Finish Initialization
82D0 - 82FF	Main Loop
8100 - 815F	Subroutine KYTIM
8160 - 817F	Subroutine LDT1
8180 - 81FF	Subroutine CLOSL and its local subroutines

Data memory assignments are:

83A0	value of N for FILTR (must be initialized to 1,2,3, or 4)
83A1,A2,A3	Used by FILTR
83A4,A5	Pulse width
83A6,A7	Desired voltage
83A8	Measured voltage
8000	Data Log Address
8001 - 80FF	Data Log

6.2.2.7 Debugging

Check that you have provided all of the proper initialization by comparing your program with the solution given in Figure 6-18. Then step through all of the initialization procedure, including calls to special entries of KYTIM and the RST6 and RST5 programmed calls to interrupt service routines. (This is an added advantage to using those calls for enabling the interrupts, since it allows the monitor to operate through the service routine).

Since FILTR was developed in an earlier exercise it should need no debugging, except for checking that it has been loaded correctly.

To check the voltmeter subroutine (VOLTM) you should omit the RST5 and RST6 in the initialization procedure, so that interrupts will not be enabled. Connect a 10K resistor in parallel with the capacitor (from ANALOG IN to ground) to obtain a voltage within the A/D range. Enter a breakpoint at the start of the voltmeter subroutine, and press RUN. Step through the voltmeter section to test the program flow, also observing the A register when the interrupt status byte is read and when the A/D input is read.

KYTIM should be checked with RST5 and RST6 still omitted. Enter an RST4 before the RET that jumps to the processing module. Run the program in STEP mode. Press a command key, and after the RST4 command is executed step through the KYTIM process for that command. When all commands have been tested, remove the RST4, restore the RST5 and RST6 instructions, and run the program in AUTO mode.

6.2.2.8 Program Operation

Start the program with an initial value of 0400 for period and C8 for voltage (2.00 volts). Turn the OPTO SENSE pot fully to the left for no resistance (highest voltage) and observe the average voltage with the voltmeter and on the display. The A/D input value varies in the less significant bits, because it senses the voltage at random points in the charge, discharge cycle. You can observe any single measurement by pressing an undefined key (e.g., ADDR). While the key is held down the measurements are stopped. Do this repeatedly and observe the range of voltage.

The voltage measured will be less than the requested 2.00 volts because the total period (500 microseconds) is too long. Gradually reduce the total period by keying in values less than 0400, followed by RUN. You should be able to obtain an accurate output of 2.00 volts, or C8 in the hexadecimal display. (If the display and voltmeter do not agree, adjust the ANALOG IN pot to make the A/D measured voltage agree with the voltmeter). The total period needed will generally be less than the nominal value of 30E, principally because the supply voltage Vc at the terminal blocks will be less than 5.0 volts. Enter different voltage requests and record the resulting output.

TOTAL PERIOD

VOLTAGE REQUEST		RESULT	
DECIMAL	HEX	VOLTMETER	A/D
2.40	F0	________	____
2.00	C8	________	____
1.50	96	________	____
1.00	64	________	____

Find the lowest voltage request that reduces the output voltage. There is a lower limit to the charging pulse width, set by the time taken by Timer 0 interrupt service. Lower voltages can only be obtained by extending the total period. Now request 2.00 volts again (C8) and alter the OPTO SENSE pot setting to reduce the output voltage to 1.50 volts (observed as 96 hex). Reduce the total period (entering values with the RUN key) to raise the output to 2.00 volts. With this setting again record the results for different voltage requests, and find the lowest value that can be achieved.

TOTAL PERIOD

VOLTAGE REQUEST		RESULT	
DECIMAL	HEX	VOLTMETER	A/D
2.40	F0	________	____
2.00	C8	________	____
1.50	96	________	____
1.00	64	________	____

The departure from linearity should be obvious. Any request lower than C8 will produce too low an output, and any request greater than C8 will produce too high an output. When we close the loop in Section 6.2.3 we will overcome this sensitivity to external conditions.

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

PWM VOLTAGE CONTROL - INITIALIZE

ADDR	CODE	Instruction	Operand	Comment
8200	3E	MVI	A,82	Program 8255#1
1	82			Port 1B in for A/D
2	D3	OUT	CNT1	
3	07			
4	3E	MVI	A,92	Program 8255 #2
5	92			
6	D3	OUT	CNT2	
7	0F			
8	3E	MVI	A,34	Program Timer 0
9	34			Both bytes
A	D3	OUT	TIMCT	Mode 2
B	17			Binary
C	3E	MVI	A,7A	Program Timer 1
D	7A			Both bytes
E	D3	OUT	TIMCT	Mode 3
F	17			Binary
8210	3E	MVI	A,94	Program Timer 2
1	94			Low byte
2	D3	OUT	TIMCT	Mode 2
3	17			Binary
4	3E	MVI	A,08	Load Timer 2
5	08			for ÷8
6	D3	OUT	TTM2	
7	16			
8	21	LXI	H,83A0	Initialize memory
9	A0			for FILTR
A	83			
B	36	MVI	M,04	n = 4
C	04			
D	AF	XRA	A	Clear 2^n E_i
E	23	INX	H	Address 83A1
F	77	MOV	M,A	
8220	23	INX	H	Address 83A2
1	77	MOV	M,A	
2	3C	INR	A	Set Port 1C0
3	D3	OUT	PORT1C	for automatic
4	06			A/D input
5	C3	JMP	82B0	Jump past
6	B0			interrupt service
7	82			and FILTR
8				

Figure 6-18a

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

PWM VOLTAGE CONTROL

ADDR		CODE		MNEMONIC	OPERAND	COMMENT
8	0			INTERRUPT		SERVICE
	1					
	2					
	3					
	4					
	5					
	6					
	7					
822	8	F	5	PUSH	PSW	RST5 – Timer 0
	9	3	E	MVI	A, 80	Set Port 1A7 high
	A	8	0			to start charging
	B	D	3	OUT	PORT 1A	All other bits low
	C	0	4			
	D	C	3	JMP	8240	Jump past RST6
	E	4	0			
	F	8	2			
8 23	0	F	5	PUSH	PSW	RST6 – Timer 1
	1	3	E	MVI	A, 00	Set Port 1A7 low
	2	0	0			to start discharge
	3	D	3	OUT	PORT 1A	(all other bits also)
	4	0	4			Note identical timing
	5	3	E	MVI	A, 03	as in RST5
	6	0	3			
	7	D	3	OUT	CNT 2	Reenable and clear
	8	0	F			Timer 1 interrupt
	9	F	1	POP	PSW	Exit
	A	F	B	EI		
	B	C	9	RET		
	C	0	0	NOP		
	D	0	0	NOP		
	E	0	0	NOP		
	F	0	0	NOP		
8 24	0	3	E	MVI	A, 01	Reenable and clear
	1	0	1			Timer 0 interrupt
	2	D	3	OUT	CNT 2	
	3	0	F			
	4	F	1	POP	PSW	Exit
	5	F	B	EI		
	6	C	9	RET		
	7					
	8					Figure 6-18b

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

PWM - SUBROUTINE VOLTM WITH LOG

ADDR	CODE		Instruction	Operand	Comments
8250	E5		PUSH	H	Save (HL)
1	21		LXI	H, 8000	Address storage
2	00	*			location for
3	80	*			data log address
4	34		INR	M	Increment address
5	C2		JNZ	8259	but not beyond FF
6	59				
7	82				
8	35		DCR	M	Restore FF
8259	6E		MOV	L, M	(HL)← data log address
A	36		MVI	M, FF	Store FF in case
B	FF				A/D not ready
C	DB		IN	PORT 2B	Read interrupt
D	0D				status byte
E	E6		ANI	08	Mask for
F	08				A/D comparator
8260	CA		JZ	826D	Jump if
1	6D				A/D not ready
2	82				
3	DB		IN	PORT 1B	Read A/D voltage
4	05				
5	32		STA	83AA	Store voltage
6	AA				in fixed location
7	83				and log in
8	77		MOV	M, A	variable location
9	3E		MVI	A, 06	Reset A/D counter
A	06				Leave interrupt
B	D3		OUT	CNT 2	disabled
C	0F				
826D	7E		MOV	A, M	(A)← voltage
E	E1		POP	H	Restore (HL)
F	C9		RET		
830					
1		*	(8000)	CON	TAINS
2			DATA	LOG	ADDRESS
3			RETURN	(A)	= VOLTAGE
4			ZERO	FLAG	NOT SET
5			IF	A/D NOT	READY
6			(A) =	FF	ZERO FLAG SET
7					
8					Figure 6-18c

SUBROUTINE FILTR

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR	CODE			
8270	D5	PUSH D		Save (DE)
1	C5	PUSH B		Save (BC)
2	46	MOV B, M		$(B) \leftarrow n$
3	48	MOV C, B		$(C) \leftarrow n$
4	23	INX H		Address $2^n E_{i-1}$
5	E5	PUSH H		Save address
6	5E	MOV E, M		
7	23	INX H		$(DE) \leftarrow 2^n E_{i-1}$
8	56	MOV D, M		
9	6B	MOV L, E		$(HL) \leftarrow 2^n E_{i-1}$
A	62	MOV H, D		
827B	29	DAD H		
C	0D	DCR C		
D	C2	JNZ 827B		$(HL) \leftarrow 2^{2n} E_{i-1}$
E	7B			
F	82			
8280	4F	MOV C, A		$(C) \leftarrow V_i$
1	7D	MOV A, L		
2	93	SUB E		$(DE) \leftarrow (HL) - (DE)$
3	5F	MOV E, A		$= 2^n(2^n - 1) E_{i-1}$
4	7C	MOV A, H		
5	9A	SBB D		
6	57	MOV D, A		
7	69	MOV L, C		
8	26	MVI H, 00		$(HL) \leftarrow V_i$
9	00			
A	CD	CALL SHFTN		Divide by 2^n
B	A0			$(DE) \leftarrow (2^n - 1) E_{i-1}$
C	82			
D	EB	XCHG		
E	19	DAD D		$(DE) \leftarrow V_i + (2^n - 1) E_{i-1}$
F	EB	XCHG		$= 2^n E_i$
3 0		CONTINUED		AT 8290
1				
2		ENTER (A)	=	V_i (new value)
3		((HL))	=	n (filter constant)
4		((HL) + 1)	}	$2^n E_{i-1}$
5		((HL) + 2)		
6		RETURN		
7		((HL) + 3) =		(A) = (H) = E_i
8				(L) = V_i

Figure 6-18d

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

ADDR	CODE		
8 29 0	E3	XTHL	(HL) ← address $2^n E_i$
1	73	MOV M, E	
2	23	INX H	Store $2^n E_i$
3	72	MOV M, D	
4	23	INX H	Address E_i
5	1B	DCX D	To round up only if
6	CD	CALL SHFTN	fractional part > 1/2
7	A0		
8	82		
9	77	MOV M, A	Store E_i
A	E1	POP H	(L) ← V_i
B	67	MOV H, A	(H) ← E_i
C	C1	POP B	Restore B, C, D, E
D	D1	POP D	
E	C9	RET	Exit
F	00	NOP	
8 2A 0	48	MOV C, B	SHFTN (C) ← n
1	AF	XRA A	Loop - clear carry
2	7A	MOV A, D	
3	1F	RAR	Shift (DE) right
4	57	MOV D, A	to divide by 2
5	7B	MOV A, E	
6	1F	RAR	
7	5F	MOV E, A	
8	0D	DCR C	Loop n times
9	C2	JNZ 82A1	to divide by 2^n
A	A1		
B	82		
C	D0	RNC	Exit if LSB = 0
D	13	INX D	Else round up
E	7B	MOV A, E	(A) ← less
F	C9	RET	significant byte
8 0			
1			
2			
3			
4			
5			
6			
7			
8			Figure 6-18e

PWM VOLTAGE - SET INITIAL VALUES

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR	CODE		Comment
8 2B 0	11	LXI D, 0400	Set total period
1	00		to 500 u sec
2	04		
3	CD	CALL RUN	Run key function of
4	38		KYTIM & loads Timer 0
5	81		for total period
6	11	LXI D, 00C8	Set voltage
7	C8		to 2.00 volts
8	00		
9	CD	CALL NEXT	Next key function of
A	20		KYTIM & loads the
B	81		width, loads Timer 1
C	EF	RST 5	
D	F7	RST 6	
E	C3	JMP 82D0	
F	D0		
8 2C 0	82		
1			
2			
3			
4			
5			
6			
7			
8			
9			
A			
B			
C			
D			
E			
F			
8 0			
1			
2			
3			
4			
5			
6			
7			
8			Figure 6-18f

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

PWM VOLTAGE CONTROL - MAIN LOOP

ADDR	CODE	Instruction	Comments
8 2D 0	DB	IN PORTA	Test keyboard
1	00		(FF if no key)
2	3C	INR A	
3	C4	CNZ KYTIM	Call for keyboard
4	00		entry and process
5	81		
6	CD	CALL VOLTM	Measure voltage
7	50		(A)← A/D voltage
8	82		(FF if not ready)
9	00	NOP	
A	00		Save 8 bytes
B	00		for closed loop
C	00		control.
D	00		
E	00		
F	00		
8 2E 0	00		
82E 1	21	LXI H, 83A0	Memory address
2	A0		for FILTR
3	83		
4	CD	CALL FILTR	
5	70		(L)← raw voltage
6	82		(H)← filtered voltage
7	11	LXI D, 83FF	Display raw voltage
8	FF		(at right)
9	83		and filtered voltage
A	CD	CALL DWD2	
B	D4		
C	02		
D	C3	JMP 82D0	Loop
E	D0		
F	82		
8 0			
1			
2			
3			
4			
5			
6			
7			
8			

Figure 6-18g

PWM - SUBROUTINE KYTIM

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR	CODE	INSTRUCTION	COMMENT
8100	00	NOP	for DI during debug
1	CD	CALL ENTWD	(A)← command
2	46		(HL)← total period
3	03		or voltage
4	EB	XCHG	(DE)← data
5	21	LXI H, 8108	Address dispatch
6	08		table -10H
7	81		
8	85	ADD L	Add command
9	6F	MOV L, A	
A	6E	MOV L, M	(ST)← dispatch address
B	E5	PUSH H	
C	F3	DI	
D	DB	IN PORT 1A	
E	04		Set Port 1A6 high
F	F6	ORI 40	for scope trigger
8110	40		without changing
1	FB	EI	Port 1A7.
2	D3	OUT PORT 1A	
3	04		
4	AF	XRA A	Clear A and CY
5	00	NOP	
6	00	NOP	During debug RST4
7	C9	RET	or BKPT here
8118	17	MEM	Undefined
9	17	REG	Undefined
A	17	ADDR	Trigger scope
B	21	STEP	Store desired voltage
C	38	RUN	Set total period
D	20	NEXT	Store voltage, set width
E	17	BRK	Undefined
F	17	CLR	Undefined
8 0			
1			
2			
3			
4			
5			
6			
7			
8			Figure 6-18h

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR	CODE		
8 12 0	37	STC	NEXT (mark with CY)
1	21	LXI H, 8000	STEP (CY clear)
2	00		Load data logging
3	80		address with its
4	75	MOV M, L	own location
5	EB	XCHG	(HL) ← input data
6	22	SHLD 83A6	Store desired voltage
7	A6		
8	83		
812 9	D0	RNC	Exit if STEP
A	11	LXI D, FFEE	To subtract 12H
B	EE		(0.18 volts) from
C	FF		desired voltage
D	19	DAD D	(HL) ← v - vs
E	29	DAD H	(HL) ← pulse width
F	00	NOP	
8 13 0	22	SHLD 83A4	Store calculated
1	A4		pulse width
2	83		
3	CD	CALL LDT1	Load Timer 1
4	60		with calculated
5	81		pulse width
6	C9	RET	
7	00	NOP	
813 8	7B	MOV A, E	RUN
9	D3	OUT TIM0	Load Timer 0
A	14		for total period
B	7A	MOV A, D	
C	D3	OUT TIM0	
D	14		
E	C9	RET	
F	00	NOP	
8 0			
1			
2			
3			
4			
5			
6			
7			
8			

Figure 6-18i

LDT1 LOAD TIMER 1

ADDR		CODE			MNEMONIC	OPERAND	COMMENTS
8 16	0	2B			DCX	H	LDT1
	1	7C			MOV	A, H	Test pulse width
	2	B7			ORA	A	for zero or
	3	23			INX	H	negative
	4	F8			RMINUS		Exit if ≤ 0
	5	7D			MOV	A, L	Load Timer 1
	6	D3			OUT	TIM1	with changing
	7	15					pulse width
	8	7C			MOV	A, H	
	9	D3			OUT	TIM1	
	A	15					
	B	C9			RET		
	C						
	D						
	E						
	F						
8	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						
	9						
	A						
	B						
	C						
	D						
	E						
	F						
8	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						

CODING SHEET
MICROCOMPUTER TRAINING SYSTEM
INTEGRATED COMPUTER SYSTEMS

Figure 6-18j

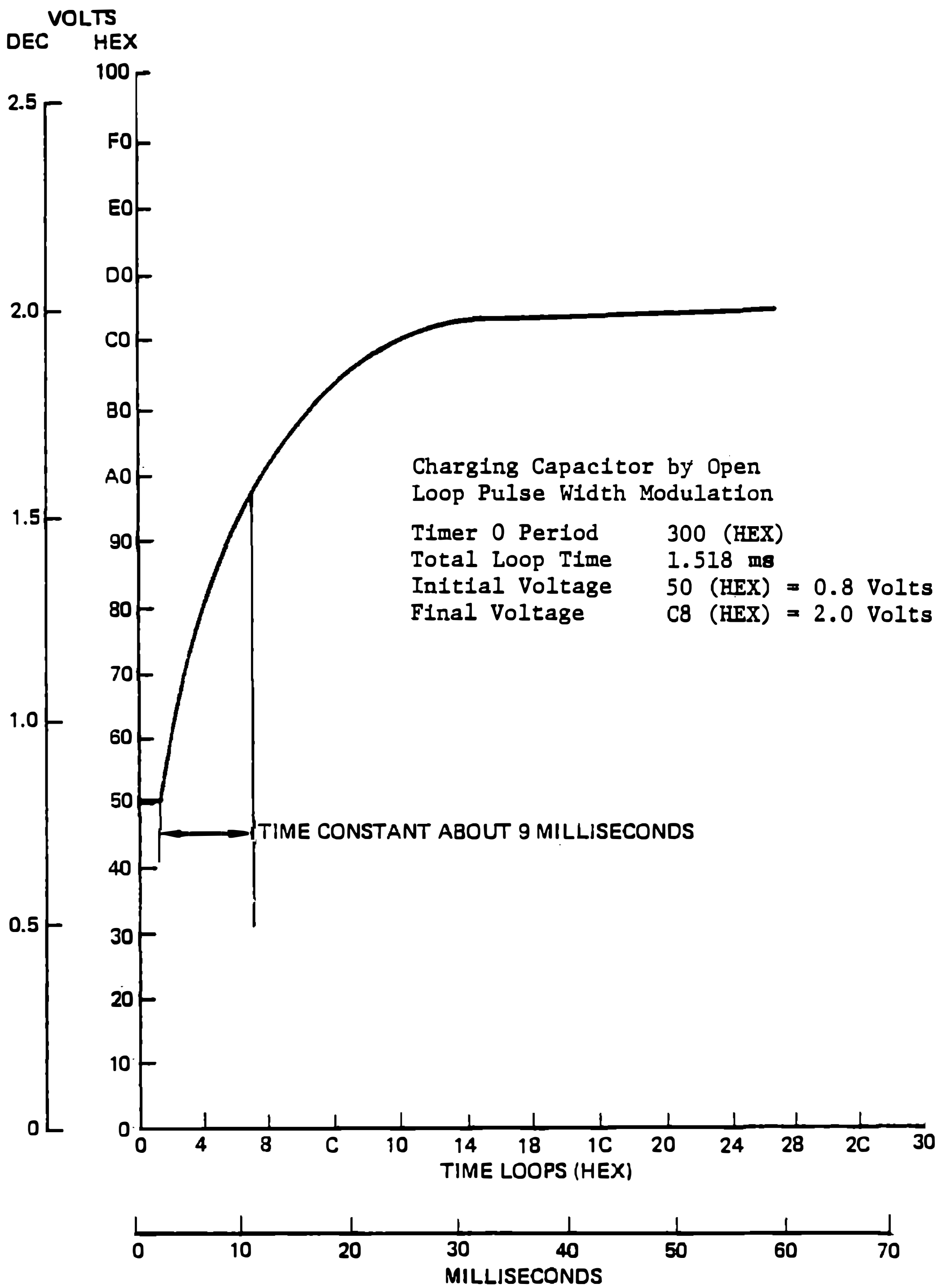
This page inentionally left blank

6.2.3 Observing Response Time

If an oscilloscope is available, observe the response of the capacitor voltage to a new input. Trigger the oscilloscope from Port 1A6, which is set high when a key is pressed and set low when the next interrupt occurs. If you do not have an oscilloscope you can observe the response by reviewing the data log created by VOLTM. Recall that whenever a new voltage request is made through KYTIM we start a new log, storing the voltage each time it is measured at the next available location in memory area 8001 - 80FF.

To use the log, set the desired total period (with RUN), enter a voltage (with NEXT) and then enter another voltage (with NEXT). Now press RST and review the data stored at 8000 - 80FF by entering ADDR 8000 and NEXT. Figure 6-19 is a plot of such data with a total period of 300 (hex), initial voltage 50 (hex) and a final voltage of C8. The curve shows the exponential charging of the capacitor toward 2.0 volts with an effective time constant of about 6 loop times or about 9 milliseconds.

A voltage has been recorded for each repetition of the main loop. These are not exactly equal intervals of time, principally because the number of interrupts during the main loop varies. With a Timer 0 period of 300 (hex) there will usually be four interrupts from each timer during the main loop, and occasionally one more, giving a total loop time of 1.49 to 1.59 milliseconds, and an average of 1.518. This value was used for the millisecond time scale in Figure 6-19.



PWM - Open Loop Response

Figure 6-19

The number of interrupts and the total loop time can be calculated from:

$$n = \frac{t_\ell}{t_p - t_i}$$

$$t_t + nt_p$$

where n = number of interrupts per loop

t_ℓ = time for one pass through the main loop and subroutines with no interrupts

t_p = total period of charge/discharge cycle (loaded to Timer 0)

t_i = timer for processing interrupts (RST5 plus RST6)

For the author's solution the values are given below.

t_ℓ	=	2092.3	clocks
t_i	=	251.1	clocks
t_p	=	768	clocks
n	=	4.048	average interrupts/loop
t_t	=	3108.7	clocks per loop
	=	1.518	milliseconds per loop

6.2.4 Closing the Loop

EXERCISE:

With the program of the preceding section we can generate a PWM voltage whose value is predictable if the circuit conditions are known. Open loop control is satisfactory in such a case Changing the SENSE pot setting alters the resulting voltage and introduces an error, which can be corrected in either of two ways. We can change the mathematical model that relates pulse width to voltage, or we can simply adjust the pulse width to achieve the desired value. Note that in this system the total period (nominally 030E) represents the "mathematical model"; a correction to this value can approximate the intended relationship of two counts of pulse width equal to one count of voltage, although it cannot remove the non-linearity. Alternately, we can adjust the pulse width to some value different than twice the desired voltage. Either of these methods can be applied by a computer program in response to an observed difference between the measured voltage and the desired voltage. We used the first method by manual entry of new periods in the preceding exercise. Now we will apply the second method automatically.

6.2.4.1 Error Signal Calculation

The error signal is the difference between the desired value and the measured value of the controlled variable. We obtain a measured value by reading the A/D input. Subroutine KYTIM has stored the desired value in memory at 83A6. (Only single byte values are meaningful now). The error signal is the desired value minus the measured value, which is positive when the measured value is too low, negative when it is too high. To adjust the driving force to correct the output we will add a positive error signal to the present charging pulse width, or subtract the magnitude of a negative error.

It turns out to be more convenient to subtract the desired voltage from the measured voltage, giving the complement of the error signal. Moreover, this seems to be a more meaningful value to display, being positive when the output is too high. Then we will take its twos complement for the correctly signed error signal to be added to the pulse width.

The error signal could range from -FF to +FF if the full voltage range of the A/D converter were available. Actually the range is somewhat less, but it is certainly greater than an eight bit value. The subtraction of measured voltage minus desired voltage gives a nine bit result in A and CY, with CY representing the sign. We will display only the eight bit value, since most of the time the sign will be obvious. For the calculation, however, we will convert it to its two byte twos complement.

This can be done by:

```
CMA           complement magnitude
MOV C,A       (C) <--- magnitude byte
CMC           complement sign
SBB A         (A) <--- 00 or FF
MOV B,A       (B) <--- sign byte
INX B         increment for two's complement
```

Now the properly signed error signal can be added to the pulse width (loaded into HL) by DAD B, giving a new pulse width.

When no error exists the pulse width will be constant; a positive error will increase the width and therefore the voltage; a negative error will decrease the width. Since the error signal is added into the steady state force, we have *integral* control. In Section 6.2.7 we will discuss the relationship between this simple control system and the integral control equation. First, however, we will develop subroutine CLOSL to perform the calculation and control the pulse width, and we will observe the results. CLOSL is to be located at 8180 - 81BF, and will be called after the return from VOLTM with (A) = measured voltage. You should now complete the main loop according to Figure 6-12. Note that CLOSL is to return data in (HL) for display by DWORD. FILTR needs the voltage returned by VOLTM, so the main loop saves that value by PUSH PSW before the call to CLOSL and recovers it before calling FILTR.

CLOSL is specified in Section 6.2.4.2 and shown in Figure 6-20. CLOSL itself calculates the error signal and loads the old pulse width. It calls another subroutine, INTEG (located at 81C0), to calculate the new pulse width. Then CLOSL calls LDT1 to load Timer 1, provided that the pulse width is positive and greater than zero.

INTEG calculates the new pulse width, in this version, simply by adding the error signal to the old width. It tests for a negative result and stores the pulse width (which is the integral of errors) if it is positive. (A zero integral is permitted, although zero is forbidden to be loaded to the timer).

The specification for INTEG in Section 6.2.4.3, Figure 6-21, states requirements that are to be met by both versions of INTEG that will be developed.

This page intentionally left blank

6.2.4.2 Subroutine CLOSL Specification (Call at 8180)

Function Calculate the error signal and set a new pulse width
Return negative error signal and desired voltage
ready for display by DWORD.

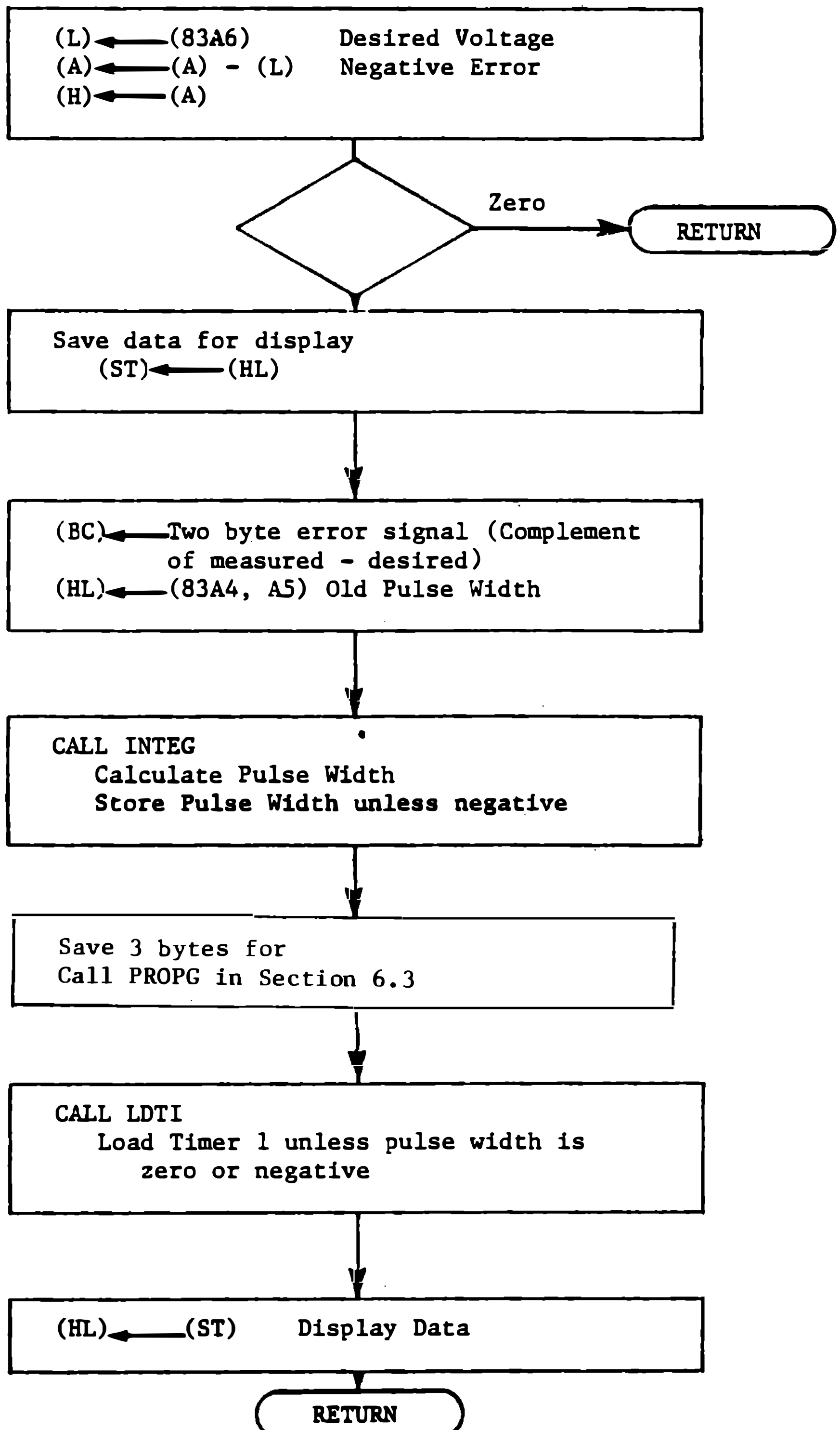
Enter (A) = Measured Voltage
(83A6) = Desired Voltage
(83A4,A5) = Old Pulse Width

All registers are used

Calls INTEG for pulse width calculation (at address 81C0)

LDT1 to load Timer 1 (at address 8160)

CLOSL Enter with (A) = Voltage



PWM Subroutine CLOSL

Figure 6-20

6.2.4.3 Subroutine INTEG Specification (call at 81C0)

Function Calculate new pulse width.
If result is positive (greater than zero)
store result.

Enter (BC) = Error Signal
(HL) = Old Pulse Width

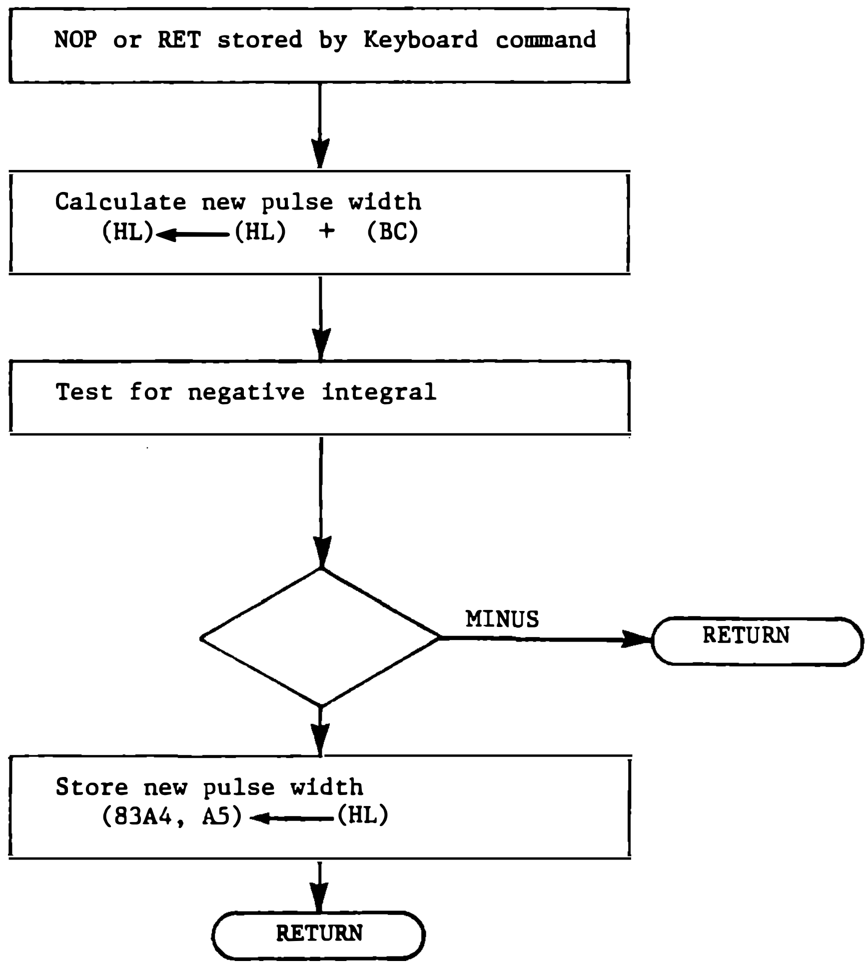
Return (HL) = New Pulse Width
(83A4,A5) = New Pulse Width

Alternate Returns

If new pulse width is less than, or equal to zero, return without storing result.

Provision is made for insertion by keyboard control of a NOP or RET instruction at the entry to INTEG. The RET will disable closed loop control.

INTEG Called by CLOSL
(BC) = Error Signal
(HL) = Old Pulse Width



PWM Subroutine INTEG

Figure 6-21

6.2.4.4 Additional Command Keys

We will now define four additional command keys to be processed by KYTIM:

REG - Force output low temporarily for observing response. (Dispatch to 8150).

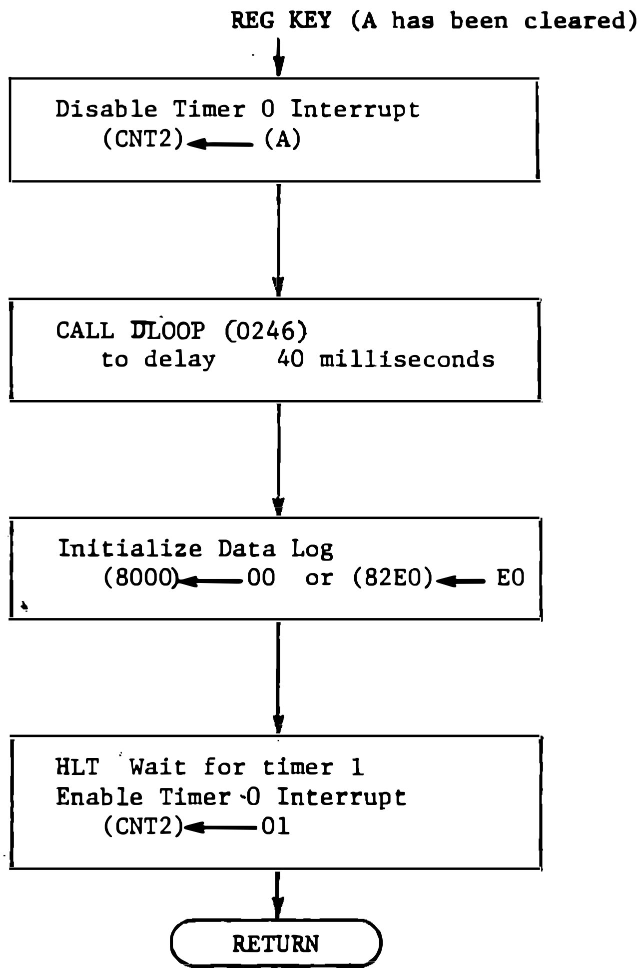
MEM - Store data to be used by a subsequent version of CLOSL. Store the data returned by ENTWD into memory locations 83A8,A9. (Dispatch to 8140).

BRK - Set open loop operation. (Dispatch to 8145).

CLR - Set closed loop operation. (Dispatch to 8147).

The change between open and closed loop operation will be accomplished by modifying subroutine INTEG. In response to CLR, store a NOP instruction at 81CO to allow INTEG to complete its functions. In response to BRK, store a RET instruction at 81CO to cause an immediate exit from INTEG.

(All of the addresses above refer to the author's solution. Your program may require different addresses).



REG Module of KYTIM

Figure 6-22

We will be interested in observing the response of the closed loop control system to a disturbance. This can be done by oscilloscope observation or by logging data. For convenience in using the oscilloscope the REG key will force a low output for long enough to discharge the capacitor. This is done by disabling the timer 0 interrupt and calling a monitor subroutine to generate the delay. At return from the delay start a new data log (as in NEXT and STEP), wait for a Timer 1 interrupt, and then re-enable and clear the Timer 0 interrupt. (See Figure 6-22).

Monitor subroutine DLOOP (located at 0246) is the delay function in GETKY. It repeatedly scans the keyboard, and returns after 20 milliseconds provided no key has been pressed. (This delay time is extended to about 40 milliseconds here by a different initial value).

Note that Timer 0, operating in mode 2, continues to run and reload itself even though its interrupt is disabled. Its output repeatedly triggers Timer 1 as in normal operation. The first interrupt form Timer 1 sets Port 1A7 low; it is not set high again until Timer 0 is enabled after the delay. A HLT instruction before enabling Timer 0 causes the first charge/discharge cycle after the delay to have its normal timing.

The main loop and subroutines KYTIM, CLOSL, and INTEG are given in Figure 6-23. Locations 8200 through 82CF are unchanged. Changes in the Main Loop and the KYTIM dispatch table are marked with an asterisk. The additions to KYTIM and the new subroutines CLOSL and INTEG are located at 8140.

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

ADDR	CODE		Instruction	Operand	Comment
8 2D 0	DB		IN	PORT0A	Test keyboard
1	00				(FF if no key)
2	3C		INR	A	
3	C4		CNZ	KYTIM	Call for keyboard
4	00				entry process
5	81				
6	CD		CALL	VOLTM	Measure voltage
7	50				(A)← A/D voltage
8	82				(FF if not ready)
9	F5	*	PUSH	PSW	Save input
A	CD	*	CALL	CLOSL	Calculate error,
B	80	*			adjust pulse width
C	81	*			
D	CD	*	CALL	DWORD	Display error
E	D1	*			and voltage request
F	02	*			
8 2E 0	F1	*	POP	PSW	(A)← input voltage
1	21		LXI	H, 83A0	Memory address
2	A0				for FILTR
3	83				
4	CD		CALL	FILTR	
5	70				(L)← raw voltage
6	82				(H)← filtered voltage
7	11		LXI	D, 83FF	
8	FF				Display raw
9	83				voltage at right
A	CD		CALL	DWD2	and filtered
B	D4				voltage
C	02				
D	C3		JMP	82D0	Loop
E	D0				
F	82				
8 0					
1		*	CHANGED	FROM	
2			FIGURE	6-18g	
3					
4					
5					
6					
7					
8					Figure 6-23a

PWM - SUBROUTINE KYTIM

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

ADDR	CODE			
8 10 0	00	NOP		
1	CD	CALL	ENTWD	(A) ← command
2	46			(HL) ← total period
3	03			or voltage
4	EB	XCHG		(DE) ← data
5	21	LXI	H, 8108	Address dispatch
6	08			table −10H
7	81			
8	85	ADD	L	Add command
9	6F	MOV	L, A	
A	6E	MOV	L, M	(ST) ← dispatch address
B	E5	PUSH	H	
C	F3	DI		
D	DB	IN	PORT 1A	
E	04			set Port 1A6 high
F	F6	ORI	40	for scope trigger
8 11 0	40			without changing
1	FB	EI		Port 1A7
2	D3	OUT	PORT 1A	
3	04			
4	AF	XRA	A	Clear A and CY
5	00	NOP		
6	00	NOP		
7	C9	RET		To dispatch address
8	40	MEM		
9	50	REG		
A	17	ADDR		Trigger scope
B	21	STEP		Store desired voltage
C	38	RUN		Set total period
D	20	NEXT		Store voltage, set width
E	45	BRK		
F	47	CLR		
8 0				
1				
2				
3				
4				
5				
6				
7				
8				Figure 6-23b

KYTIM - NEXT, STEP, RUN

ADDR	R	CODE	Instruction	Comments
8 12	0	37	STC	NEXT (mark with CY)
812	1	21	LXI H, 8000	STEP (CY clear)
	2	00		Address data log
	3	80		and load with its
	4	75	MOV M, L	own address
	5	EB	XCHG	
	6	22	SHLD 83A6	
	7	A6		
	8	83		
	9	D0	RNC	Exit if STEP
	A	11	LXI D, FFEE	to subtract 12H
	B	EE		(0.18 volts) from
	C	FF		desired voltage
	D	19	DAD D	(HL)← v - Vo
	E	29	DAD H	(HL)← pulse width
	F	00	NOP	
8 13	0	22	SHLD 83A4	Store calculated
	1	A4		pulse width
	2	83		as integral
	3	CD	CALL LTD1	Load Timer 1
	4	60		with calculated
	5	81		pulse width
	6	C9	RET	
	7	00	NOP	
813	8	7B	MOV A, E	RUN
	9	D3	OUT TIM0	Load Timer 0
	A	14		for total period
	B	7A	MOV A, D	
	C	D3	OUT TIM0	
	D	14		
	E	C9	RET	
	F	00	NOP	
3	0			
	1		IDENTICAL	TO
	2		FIGURE 6-18i	
	3			
	4			
	5			
	6			
	7			
	8			

CODING SHEET
MICROCOMPUTER TRAINING SYSTEM
INTEGRATED COMPUTER SYSTEMS

Figure 6-23c

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

KYTIM - MEM, BRK, CLR, REG

ADDR	CODE											
8 14 0	E	B	X	C	H	G						MEM
1	2	2	S	H	L	D		8	3	A	8	Store gains for CLOSL
2	A	8										
3	8	3										
4	C	9	R	E	T							
814 5	3	E	M	V	I		A	,	C	9		BRK - Open loop
6	C	9										Enter RET in INTEG
814 7	3	2	S	T	A			8	1	C	0	CLR - close loop
8	C	0										Enter NOP in INTEG
9	8	1										
A	2	1	L	X	I		H	,	8	0	0 0	Start a new log.
B	0	0										
C	8	0										
D	7	5	M	O	V		M	,	L			
E	C	9	R	E	T							
F	0	0	N	O	P							
8 15 0	D	3	O	U	T		C	N	T	2		REG - Force output low
1	0	F										Disable Timer 0 interrupt
2	C	D	C	A	L	L		D	L	O	O P	Monitor function
3	4	6										Delays ~40 milliseconds
4	0	2										
5	2	1	L	X	I		H	,	8	0	0 0	Start a new log
6	0	0										
7	8	0										
8	7	5	M	O	V		M	,	L			
9	7	6	H	L	T							Wait for Timer 1
A	3	E	M	V	I		A	,	0	1		Enable and clear
B	0	1										Timer 0 interrupt
C	D	3	O	U	T		C	N	T	2		
D	0	F										
E	C	9	R	E	T							
F	0	0	N	O	P							
8 0												
1												
2												
3												
4												
5												
6												
7												
8												Figure 6-23d

LDT1 LOAD TIMER 1

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR		CODE			
8 16	0	2B	DCX	H	LDT1
	1	7C	MOV	A, H	Test pulse width
	2	B7	ORA	A	for zero or negative
	3	23	INX	H	
	4	F8	RMINUS		Exit if ≤ 0
	5	7D	MOV	A, L	Load Timer 1
	6	D3	OUT	TIM1	with charging
	7	15			pulse width
	8	7C	MOV	A, H	
	9	D3	OUT	TIM1	
	A	15			
	B	C9	RET		
	C				
	D				
	E				
	F				
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				
	9				
	A				
	B				
	C				
	D				
	E				
	F				
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				

Figure 6-23e

INTEGRATED COMPUTER SYSTEMS — MICROCOMPUTER TRAINING SYSTEM — CODING SHEET

PWM - SUBROUTINE CLOSL - VERSION 1

ADDR		CODE	INSTRUCTION	COMMENTS
8 18	0	2A	LHLD 83A6	CLOSL
	1	A6		(L)← desired voltage
	2	83		At entry (A)=measured
	3	95	SUB L	(A)← -Error Signal
	4	67	MOV H, A	(H)← -Error Signal
	5	C8	RZ	Exit if error =0
	6	E5	PUSH H	Save display data
	7	2F	CMA	
	8	4F	MOV C, A	
	9	3F	CMC	(BC)← +Error
	A	9F	SBB A	as two bytes
	B	47	MOV B, A	
	C	03	INX B	
	D	2A	LHLD 83A4	(HL)← Old pulse width
	E	A4		
	F	83		
8 19	0	CD	CALL INTEG	Calculate and store
	1	C0		new pulse width
	2	81		
819	3	00	NOP	Save 3 bytes for
	4	00	NOP	proportional control
	5	00	NOP	
	6	CD	CALL LDT1	Load Timer 1 with
	7	60		new pulse width
	8	81		
	9	E1	POP H	Recover display data
	A	C9	RET	
	B			
	C			
	D			
	E			
	F			
8	0		ENTER WITH	
	1		(A) = MEASURED	
	2		RETURN WITH	
	3		(L) = DESIRED VOLTAGE	
	4		(H) = NEGATIVE ERROR	
	5		INTEGRAL AND TIMER 1	
	6		UPDATED	
	7			
	8			Figure 6-23f

INTEGRATED COMPUTER SYSTEMS — MICROCOMPUTER TRAINING SYSTEM — CODING SHEET

ADDR	CODE			COMMENT
8 1C 0	00		NOP/RET	Set by keyboard entry
1	09		DAD B	(HL)← new integral
2	7C		MOV A, H	Test for negative
3	B7		ORA A	integral
4	F8		RMINUS	Exit if negative
5	22		SHLD 83A4	Store new integral
6	A4			
7	83			
8	C9		RET	
9				
A				
B				
C			ENTER WITH	
D			(BC) = + ERROR	
E			(HL) = OLD	INTEGRAL
F				
8 0			RETURN	
1			(BC) PRESERVED	
2			(DE) PRESERVED	
3			(HL) NEW	INTEGRAL
4				
5			NEW INTEGRAL STORED	
6			UNLES NEGATIVE	
7				
8				
9				
A				
B				
C				
D				
E				
F				
8 0				
1				
2				
3				
4				
5				
6				
7				
8				Figure 6-23g

6.2.5 Closed Loop Operation

With closed loop control the program will force the output voltage to equal the requested voltage. You can enter a voltage with the NEXT key, which calculates a new pulse width, or with STEP, which does not. In either case the program will adjust the duty cycle to generate the requested voltage. The voltage will now be independent of the total period and the OPTO SENSE pot setting. It will fluctuate over a range of about six counts (60 millivolts), from C5 to CB. The fluctuation is displayed in the measured voltage (at the right) and the error signal (at the left). These will be changing so rapidly as to be unreadable. Since the measurement and display are handled in the main loop, pressing a key will stop the measurements and allow you to read the voltage. By doing this repeatedly you can observe the range of measurements. The ADDR key will do this without changing any stored data or controls. The fluctuation of the voltage is an inherent and undesirable effect of closed loop control Fortunately it can be reduced by several means that we will investigate later.

Perform the experiments described in the following sections. Results of these experiments can be observed with your voltmeter. The experiments of Section 6.2.6 require either an oscilloscope or the laborious task of plotting data from the log made by the VOLTM subroutine.

6.2.5.1 Seeking Desired Voltage

Enter the following data and commands:

CLR	Set closed loop control
400,RUN	Set 0.5 millisecond period
C8,STEP	Set 2.0 volts

Observe the filtered voltage in the second pair of digits from the right, and observe the voltmeter reading. They should agree closely. Adjust the ANALOG IN pot, and observe that this now changes the actual output as observed on the voltmeter, because the closed loop control forces its measured input voltage to equal the requested voltage. Now enter the following voltages and observe the resulting output.

Total Period 0.5 ms (400 hex)

Voltage Request			Result	
Decimal	Hex		Voltmeter	A/D
2.40	F0	STEP	________	____
2.00	C8	STEP	________	____
1.50	96	STEP	________	____
1.00	64	STEP	________	____

6.2.5.2 External Resistance Variation

Enter the following data and commands:

CLR	Set closed loop control
400,RUN	Set 0.5 millisecond period
C8,STEP	Set 2.0 volts

Adjust the OPTO SENSE pot over its full range from left to right and back to the left again. There should be no appreciable change in the voltmeter reading.

BRK	Set open loop control

Adjust the SENSE pot to reduce the voltage to 1.50 volts (96 hex).

CLR	Set closed loop control

Request the several voltages again and observe the results.

Total Period 0.5 ms (400 hex)

Voltage Request			Result	
Decimal	Hex		Voltmeter	A/D
2.40	F0	STEP	________	____
2.00	C8	STEP	________	____
1.50	96	STEP	________	____
1.00	64	STEP	________	____

The results should be essentially the same as before.

6.2.5.3 Total Period Variation

Return the SENSE pot to the full left position. Enter:

CLR	Set closed loop control
400,RUN	Set 0.5 millisecond period
C8,STEP	Set 2.0 volts

Now enter different total periods and observe the results.

Period			Voltage	
Milliseconds	Hex		Voltmeter	A/D
0.375	0300	RUN	________	_____
0.50	0400	RUN	________	_____
0.75	0600	RUN	________	_____
1.00	0800	RUN	________	_____
2.00	1000	RUN	________	_____
32.00	0000	RUN	________	_____

With the final value (32 milliseconds) a greater difference between the voltmeter and the A/D converter may occur because of the wide variation in the voltage within a charge/discharge cycle. With the two millisecond period the capacitor charges and discharges by 300 millivolts in each cycle. At 32 milliseconds the voltage is actually swinging from 0.6 to 3.6 volts, but the average is held to 2.1 volts by closed loop control.

After the 32 millisecond test enter 400,RUN. The voltage will rise very nearly to 5.0 volts, because the charging time will have been set to about 15 milliseconds, much longer than the newly entered total period. Since the A/D converter cannot measure the off-scale voltage it returns FF. CLOSL calculates an error signal of C8 - FF = -37 and repeatedly reduces the pulse width by this amount until the voltage returns to a value within range, and thereafter adjusts the pulse width appropriately.

6.2.5.4 Response to Disturbance

Enter the following data and commands:

CLR	Set closed loop control
400,RUN	Set 0.5 millisecond period
C8,STEP	Set 2.0 volts
REG	Force voltage low

The output voltage will momentarily drop to about 0.2 volt and rise again to the requested voltage. We will observe this response in detail in section 6.2.6. The voltmeter will show the drop, but it will not be fast enough to go down more than a few tenths of a volt before closed loop control resumes and restores the desired voltage.

6.2.5.5 Open Loop Operation

The open loop control program is able to maintain a voltage very well, but is unable to compensate for changes in external conditions, as we observed earlier. Once an appropriate pulse width has been set by the closed loop system we can open the loop and the voltage will remain essentially constant. Enter these data and commands:

CLR	Set close loop control
400,RUN	Set 0.5 millisecond period
C8,STEP	Set 2.0 volts
REG	Force voltage low
BRK	Set open loop control
REG	Force voltage low

Observe that the open loop system returns to the requested voltage once an appropriate pulse width has been set by closed loop control. Now with open loop operation, adjust the SENSE pot to obtain 1.50 volts. Now press:

CLR	Set closed loop control
BRK	Set open loop control
REG	Force output low

CLOSED LOOP CONTROL

Again closed loop control has set the pulse width and open loop control can restore the voltage after a disturbance. This is sometimes an acceptable mode of operation for a control system where external variables change slowly. Closed loop control can be invoked periodically, or when conditions are known to have changed. After the necessary adjustments have been made to the control force an open loop system can maintain the operation.

Students who are not especially interested in closed loop control may want to skip the remainder of Chapter 6. It is concerned with methods of improving the performance of a closed loop control system.

6.2.6 Closed Loop Response

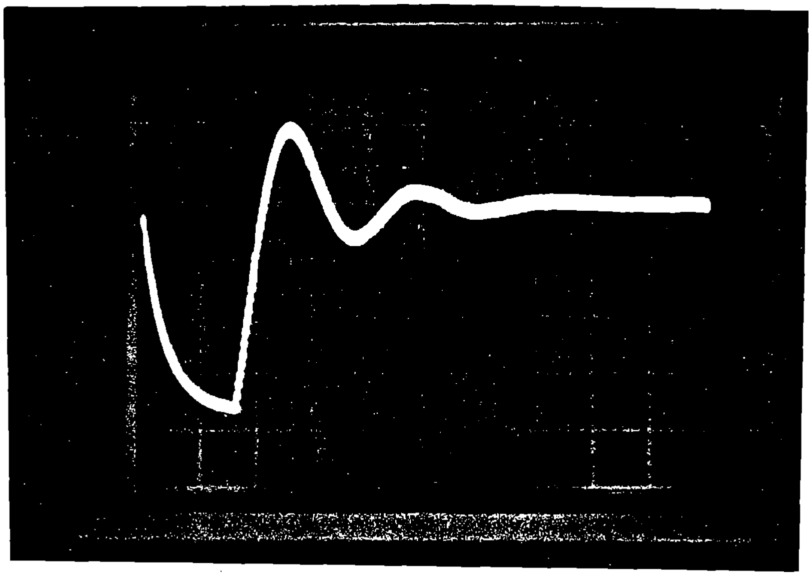
Observing the response to a disturbance under various conditions demonstrates very important features of closed loop control systems. This can be done most conveniently with an oscilloscope, or data can be logged and plotted. Section 6.2.6.1 describes the use of the oscilloscope and shows open and closed loop waveforms. Section 6.2.6.2 presents closed loop results obtained by the data log, and discusses the waveforms. The effect of changing the total period is shown in 6.2.6.3.

6.2.6.1 Oscilloscope Observation

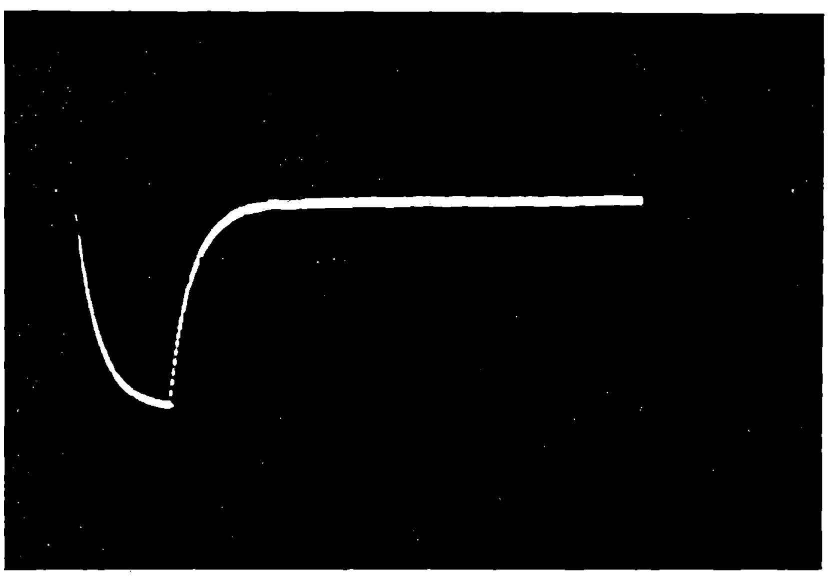
The oscilloscope is to be triggered by Port 1A6, which is set when a command key has been pressed and released. The capacitor voltage is to be observed.

Use a time scale of 20 milliseconds per division and a voltage scale of 0.5 volts/division. Enter the following:

CLR	Set closed loop control
400,RUN	Set 0.5 millisecond period
C8,STEP	Set 2.0 volts



CLOSED LOOP RESPONSE



OPEN LOOP RESPONSE

SCALES: 0.5 VOLT/DIV 20 MS/DIV

Open and Closed Loop Waveforms

Figure 6-24

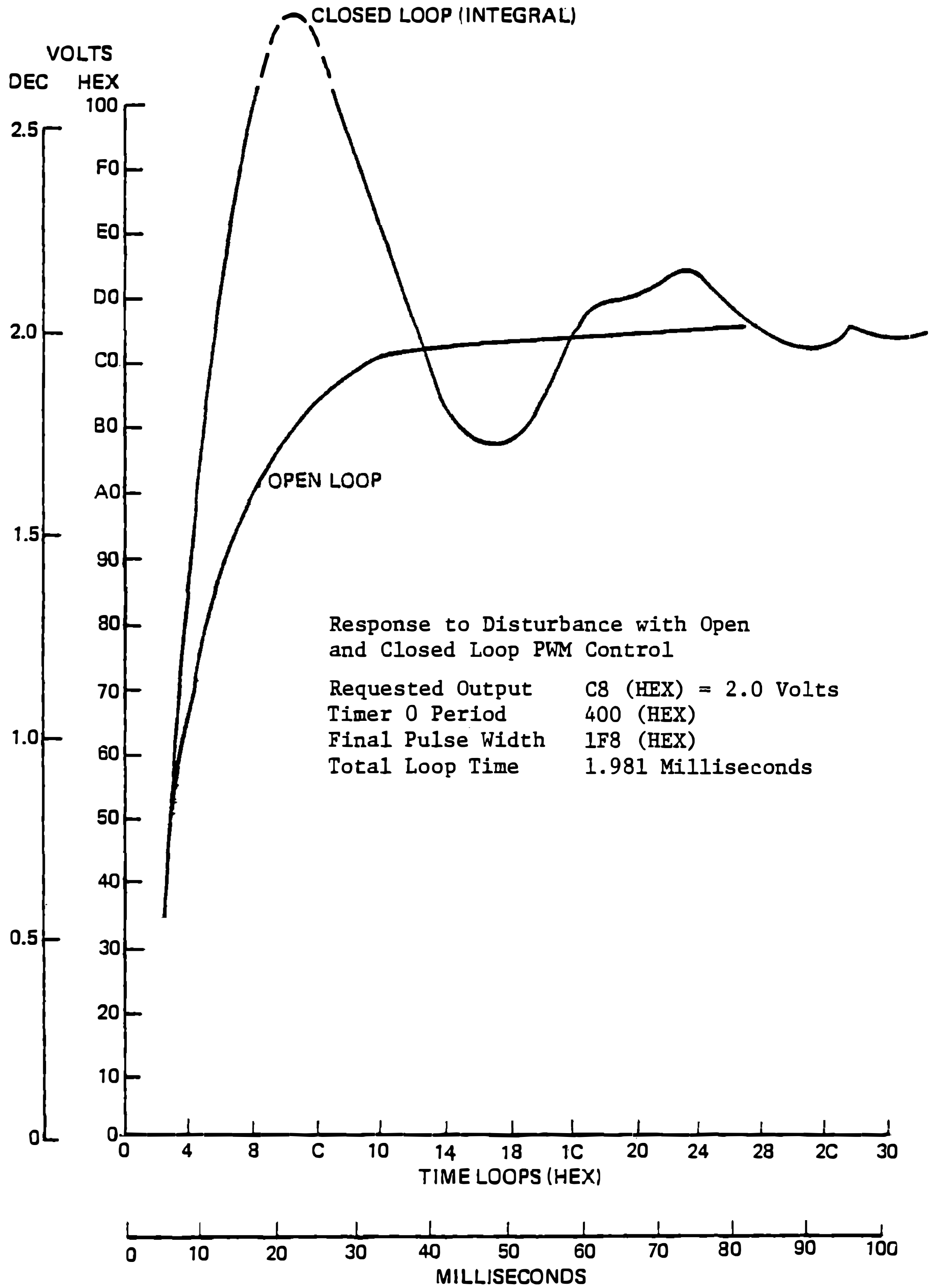
Test the scope triggering by repeatedly pressing ADDR. A single sweep should occur each time you release the key. This may take some adjustment of the oscilloscope trigger controls.

When you press and release REG the output will be forced low for about 40 milliseconds and then closed loop control will be resumed. The oscilloscope should display a waveform similar to the upper photograph in Figure 6-24. Now:

BRK	Set open loop control
REG	Force output low

A waveform similar to the lower photograph in Figure 6-24 should be seen.

On repeated operations of REG with open loop control the waveform should be very consistent. It merely shows the charging of the capacitor in response to a constant duty cycle PWM voltage. In closed loop control there will be substantial variations in the waveform, principally because of the random relationship between the time that the A/D conversion is completed and the time that CLOSL acts on the result. The reasons for the waveshape are discussed in the next section.

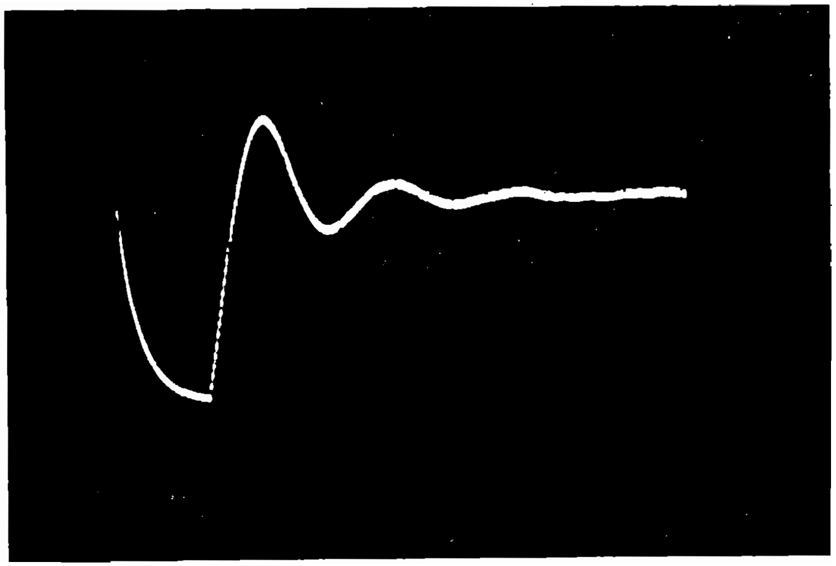


Closed Loop Response Waveform

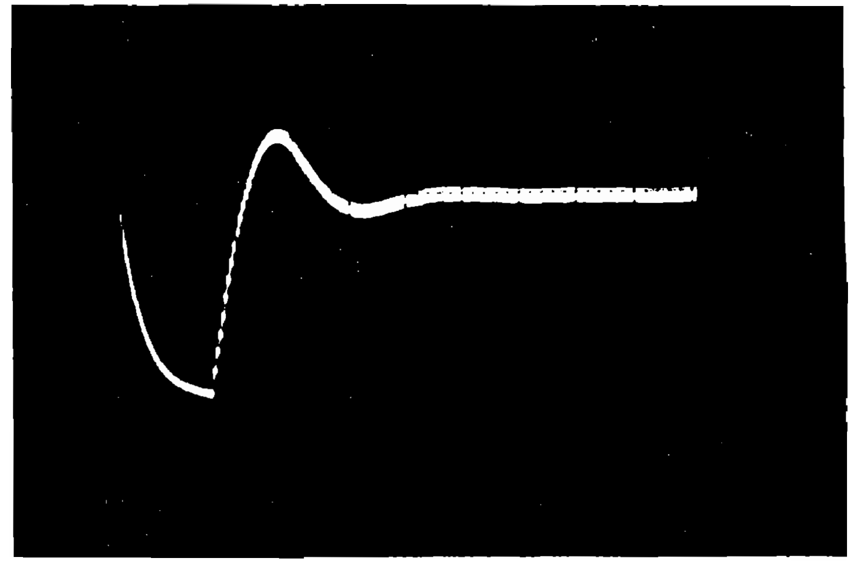
Figure 6-25

6.2.6.2 Closed Loop Response Waveform

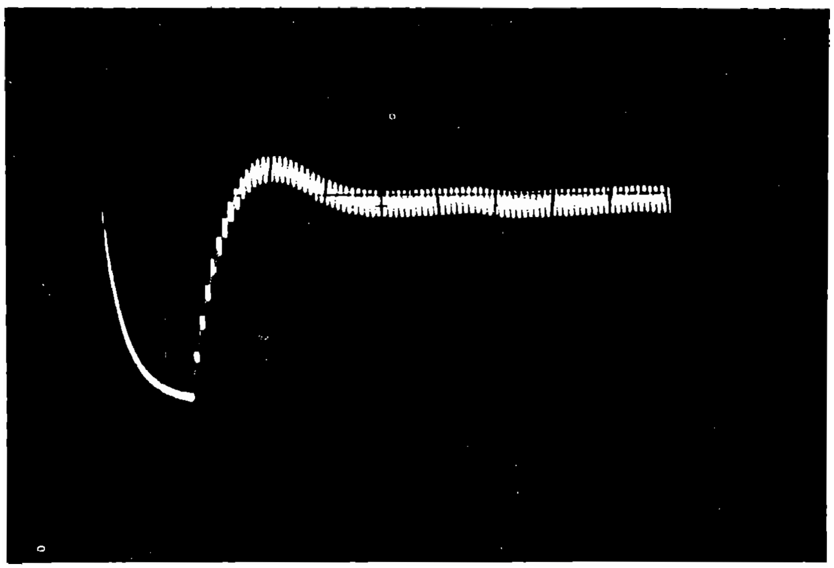
Waveforms observed as the closed loop control system restores the desired voltage after a disturbance are shown in Figures 6-24 and 6-25. The large overshoot as the desired voltage as approached, and the continuing oscillation above and below the desired voltage are inherent and undesirable results of Integral Control. When the low output voltage is measured the computer calculates a large error signal and adjusts the pulse width accordingly. At the next measurement a smaller but still substantial error is observed and a further adjustment is made to the pulse width. This continues until the actual voltage has reached and passed the desired voltage. At this point the pulse width is set too wide for the desired voltage. A negative error is detected and the pulse width is reduced. The capacitor is still charging however, so the voltage continues to rise. Moreover, the error detected is small and the adjustment made is not yet sufficient to bring the voltage back down to the desired value. The result is that the voltage rises substantially above the desired value before it starts down. This is called "overshoot". A number of measurements and adjustments are made before the voltage again reaches the desired value. By now the closed loop control system has reduced the pulse width too far, and undershoot occurs. The process continues indefinitely, reaching a steady state of oscillation above and below the desired value. The amount of overshoot and undershoot, the amplitude of the oscillation, and the time before a steady state is reached will be seen to depend on the total period of the pulse width modulation.



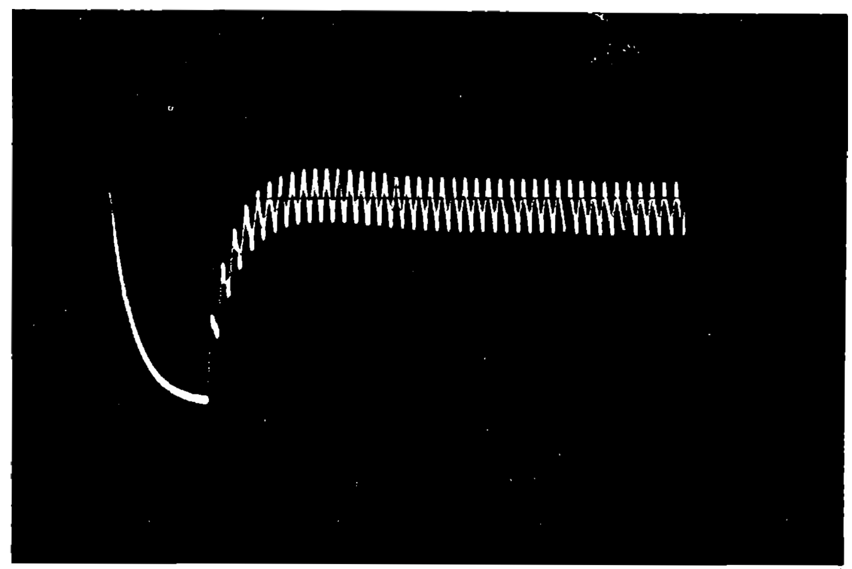
TOTAL PERIOD 400 (hex) 0.5 MS



TOTAL PERIOD 800 (hex) 1.0 MS



TOTAL PERIOD 1000 (hex) 2.0 MS



TOTAL PERIOD 2000 (hex) 4.0 MS

SCALES: 0.5 VOLTS/DIV, 20 MS/DIV

EFFECT OF TOTAL PERIOD
FIGURE 6-26

6.2.6.3 Effect of Total Period

Figure 6-26 shows oscilloscope traces for four different values of total period. It is apparent that by using a long period we can greatly reduce the overshoot and oscillation, but at the cost of introducing a large voltage fluctuation in each charge/discharge cycle. With a total period of 2,000 (hex) the overshoot is small but the voltage varies by about 0.4 volt during each cycle. This is clearly not a desirable scheme. When we develop a more sophisticated closed loop control system in Section 6.3 we will overcome this problem.

6.2.6.4 Gain of Integral Control

We have exercised integral control by adding the error signal to the steady state pulse width. Let use examine the meaning of this procedure in terms of the integral control equation:

$$\text{(a)} \qquad F = G\int E$$

Since the Integral represents the sum of all past error signals plus the new measurement, we can say:

$$\text{(b)} \qquad F = GE + F'$$

Where F' is the previous value of the control force.

The relation tc/tp represents the control force, since we can scale these two values without changing the result, but changing either alone does affect the output. Therefore:

(c) $$F = \frac{t_c}{t_p}$$

(d) $$F' = \frac{t_{c'}}{t_p}$$ for constant total period

(e) $$\frac{t_c}{t_p} = GE + \frac{t_{c'}}{t_p}$$ from (b) and (d)

The procedure we have used added the error signal E to the old pulse width tc' to obtain a new tc.

(f) $$t_c = E + t_{c'}$$

Dividing by tp:

(g) $$\frac{t_c}{t_p} = \frac{E}{t_p} + \frac{t_{c'}}{t_p}$$

From inspection of equations (e) and (g) it is apparent that G = 1/tp. When we increased the total period to reduce the overshoot we were reducing the gain of the control system. It is much more effective to do this by arithmetic in the control calculation, and this will be done in Section 6.3.

6.2.6.5 Pure Proportional Control

At the beginning of Section 6.2 we presented the control equation for proportional control with a steady state force:

(a) F = GE+S

The program of Section 6.3 will introduce a proportional term separate from the integral term we have been using. To see the effect of proportional control alone, change the RM instruction in INTEG to RET. In the program of Figure 6-23g, this change is:

```
81C4    C9    RET    (was RM)
```

The pulse width will be calculated as before but the integral will never be stored. This gives proportional control with a steady state term set by the open loop system. Do this experiment:

CLR	Set closed loop (proportional) control
C8, STEP	Request 2.0 volts
03E8, RUN	Enter total period

Adjust the total period to obtain 2.0 volts.

REG	Force output low

Observe that proportional control restores the output voltage much as open loop control would. Now open the loop and restore the 500 microsecond total period.

```
        BRK

        0400, RUN
```

The voltage will be lower than requested, because of the excessive total period. Close the loop with proportional conrol.

```
        CLR                      Set proportional control
```

Proportional control will increase the output voltage, but will not reach the desired value. This is because the nominal pulse width calculated in response to NEXT remains as the steady state value. The pulse width is increased by the proportional control system, but only by the amount of the error measured at that instant with no cumulative correction. In the system we have here, pure proportional control is little more effective than open loop control. It is useful in systems where no steady state control force is needed, or in combination with integral control.

6.3 PROPORTIONAL PLUS INTEGRAL CONTROL

A control system that requires a steady state force to be adjusted for variable external conditions demands integral control. Random disturbances are better overcome by proportional control. Therefore a combination of both forms of control is very commonly used; it is called Proportional Plus Integral control. The control equation becomes:

$$F = G_p E + G_i \int E$$

We determined at the end of section 6.2 that $G_i = 1/tp$ in our present system, where the error signal is added into the integral and the result sets the charging pulse width.

If we then add the error term again before loading the timer but do not change the integral in response to this addition, we will have a proportional plus integral system with equal gains.

We recognized in Section 6.2.6.4 a need to reduce the integral gain to avoid large overshoot. We also observed that with pure proportional control a large error signal was needed to affect the output significantly. The proportional control would have been more effective with a higher gain, since then the correction applied would have been greater for any given error signal. It is very common in proportional plus integral control systems for the proportional gain to be much greater than the integral gain.

6.3.1 Applying Gain to Error Signal

In our new system we will provide for dividing the error signal by some value before adding it into the integral term, and multiplying it by some other value before adding it as the proportional term.

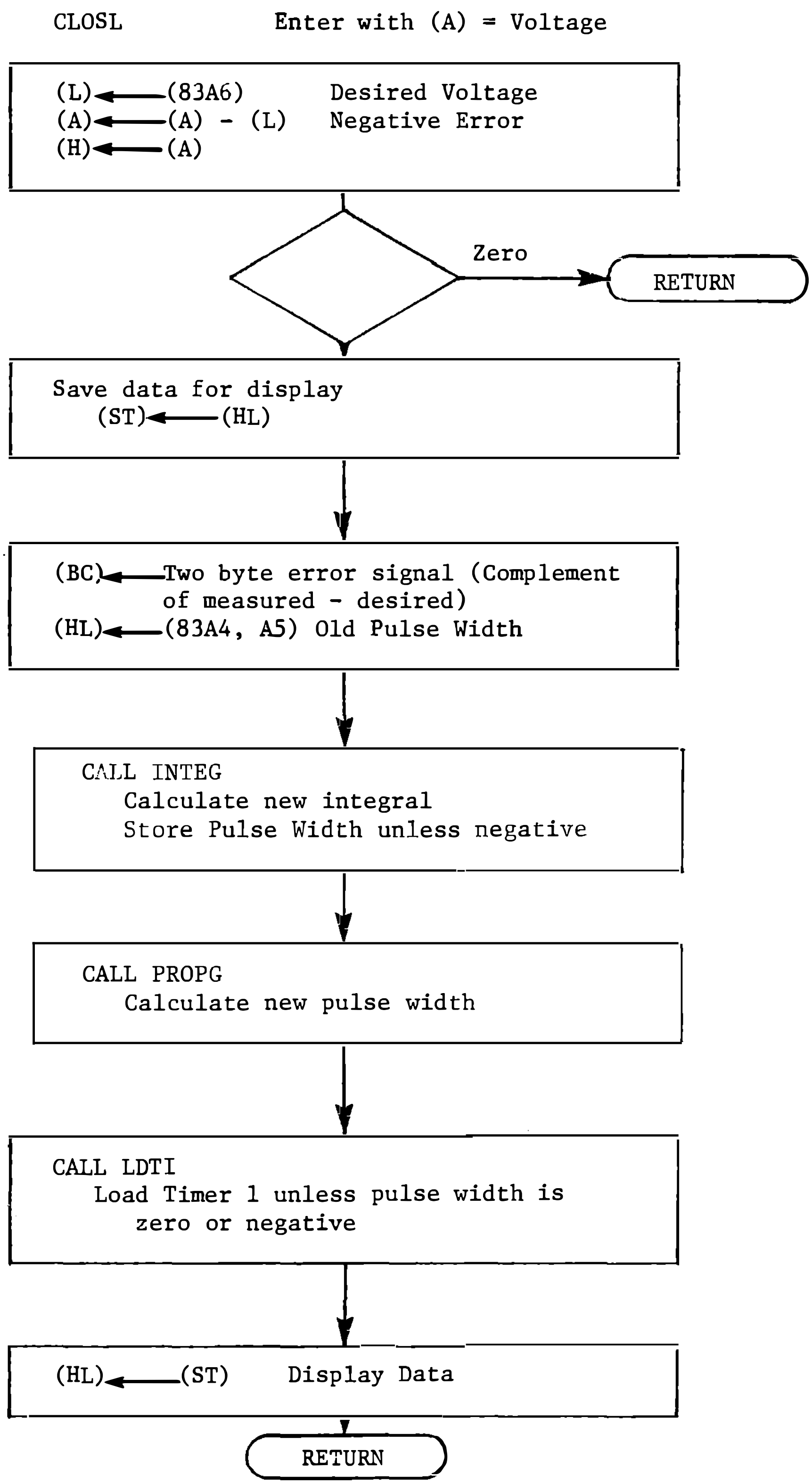
For ease of computation the integral gain divisor will be of the form $1/2^n$, so that the division is merely a shift of n bits to the right. Typical values for n will be 0 to 4, giving division by 1, 2, 4, 8 or 16. The multiplication for the proportional term will be done by repeated addition, so the number used (again typically 0 to 4) will now actually be the multiplier. Our control equation will now be

$$F = \frac{KE}{t_p} + \frac{1}{2^n\ t_p} \int E$$

For convenience in further discussion we will ignore the total period here and speak of K and $1/2^n$ as the proportional and integral gains. These are the data elements stored by the MEM key, at (83A8,A9). Both values are to be entered: K first, followed by n, followed by MEM. For instance, 203 MEM will set k=2 and n=3, giving a proportional gain of 2 and integral gain of 1/8. Note that the data entry procedure stores k at 83A9 and n at 83A8.

The calculation procedure is described briefly below, with detailed flow charts in Figure 6-27 and the program in Figure 6-28.

CLOSL:	Calculate error signal
INTEG:	Divide error by 2^n
	Add to old integral
	Store new integral unless negative
PROPG:	Multiply error by k
	Add to new integral
LDT1:	Load Timer 1 with new pulse width unless zero or negative



PWM Subroutine CLOSL

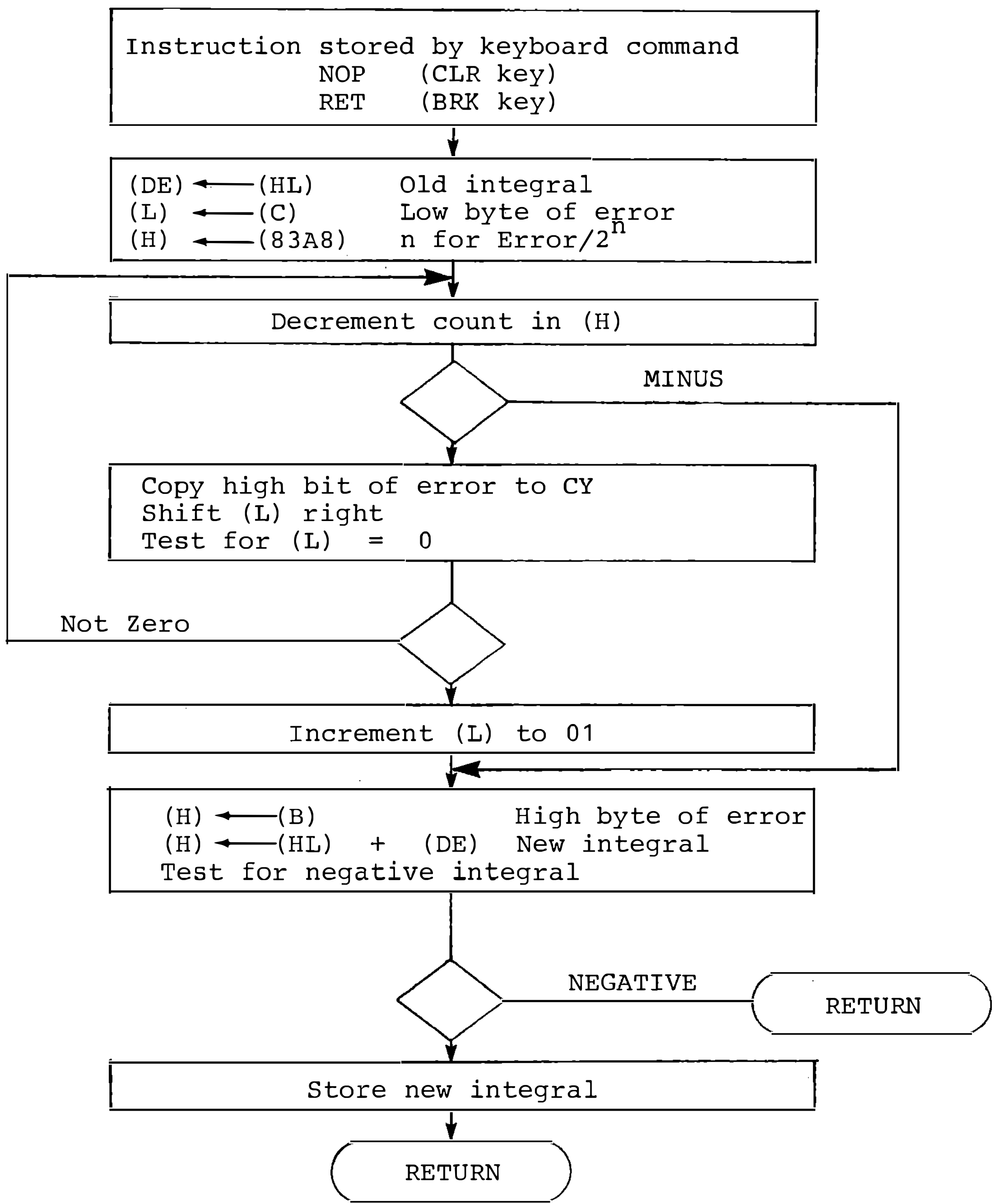
Figure 6-27a

6.3.2 Subroutine CLOSL Version 2

CLOSL is unchanged except that after calling INTEG to generate and store the new integral it now calls another subroutine PROPG to apply the proportional error term.

A revised version of INTEG divides the error by 2^n (shifts the error right by n bits) and adds this to the old integral. The new integral is stored, unless it is negative; it is returned in (HL) even if it is negative.

The new subroutine PROPG repeatedly adds the error into the integral, according to the value entered as K and stored at 83A9. Thus PROPG returns the sum of the integral and proportional terms.



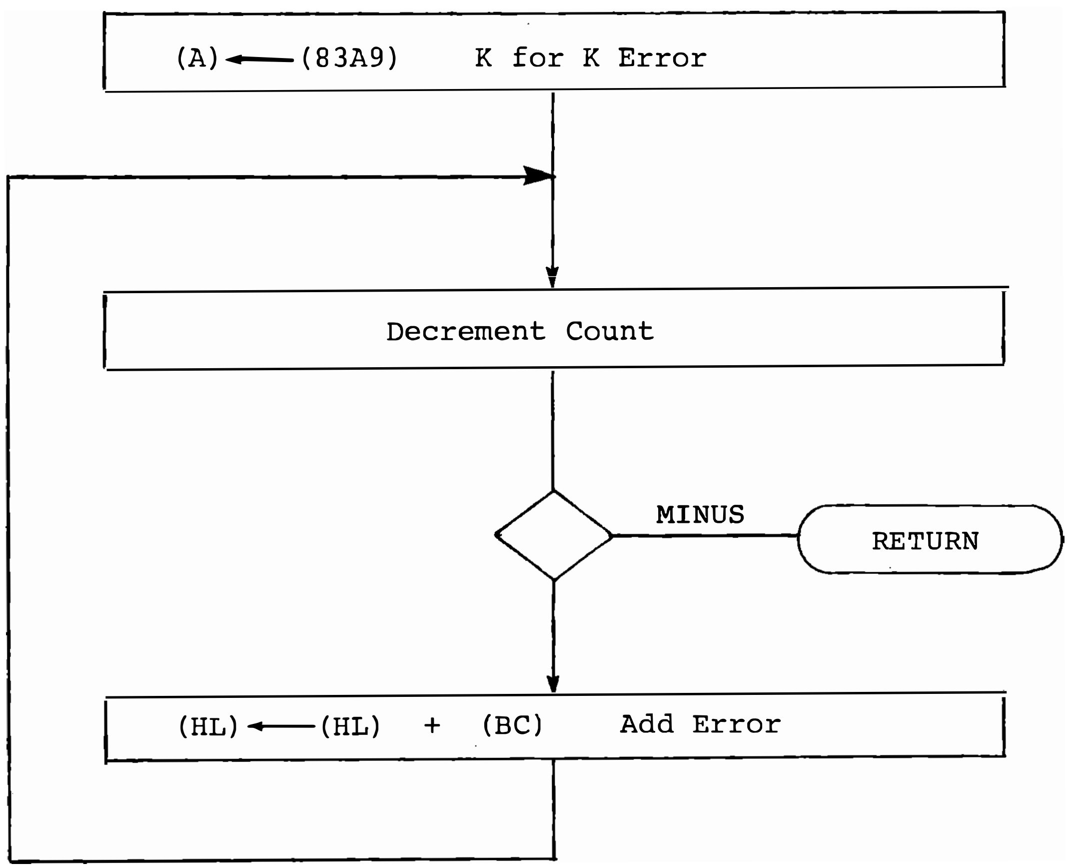
PWM - Subroutine INTEG Version 2

Figure 6-27b

6.3.2.1 Subroutine INTEG Version 2

Again provision is made for modifying the program by BRK and CLR, storing NOP or RET at the start of INTEG. CLR will set integral control (by entering NOP). BRK will disable integral control, leaving pure proportional control. If the proportional gain multiplier is set to zero the process is then identical to open loop operation. To divide the error signal by 2^n we shift the error right by n bits. Since the high byte of the error is always either 00 or FF we need not shift the high byte, but merely shift its low bit into the low byte as the low bit is shifted right. Note from Figure 6-27b that n is decremented before shifting, so that if n is zero initially no shifting is done and the gain is unity.

When n is greater than zero and the error signal is a small positive value, the result may be zero even if there was an error. It is desirable to increase the integral in this case, especially since an equally small negative error will reduce the integral. (FFFF shifted right remains FFFF). Therefore, we test the shifted error for zero, and if it is zero increment it to 01. Now a positive error will always increase the integral and a negative error will always decrease it, no matter how small the gain. A zero error does not affect the pulse width because of the Return If Zero in CLOSL after the error calculation. As before we must test for a negative integral in subroutine INTEG before storing the result. Zero is not forbidden here, and negative integrals would be acceptable but, as we will demonstrate, there is a possibility of losing control.



PWM - Subroutine PROPG

Figure 6-27c

6.3.2.2 Subroutine PROPG (Figure 6-27c)

The error signal is to be multiplied by K (stored at 83A9) and added to the integral, to obtain the new pulse width. Subroutine INTEG preserves the error in (BC) and returns the new integral in (HL). Since K will always be a small integer value it is efficient to simply add (BC) into (HL) while counting down. Once again the count should be decremented before the addition, so that a zero value for K will give a zero again.

6.3.3 Revised Program

The revised program with the new INTEG (version 2) and PROPG is given in Figure 6-28. For convenient reference, the main loop and subroutines KYTIM and LDT1 are repeated although no changes have been made. CLOSL now includes the call to PROPG after the call to INTEG. The new subroutines appear in Figures 6-28g and 6-28h.

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

PWM VOLTAGE CONTROL - MAIN LOOP

ADDR		CODE		INSTRUCTION	OPERAND	COMMENT
8 2D	0	D	B	IN	PORTOA	Test keyboard
	1	0	0			(FF if no key)
	2	3	C	INR	A	
	3	C	4	CNZ	KYTIM	Call for keyboard
	4	0	0			entry and process
	5	8	1			
	6	C	D	CALL	VOLTM	Measure voltage
	7	5	0			(A)← A/D voltage
	8	8	2			(FF if not ready)
	9	F	5	PUSH	PSW	Save input
	A	C	D	CALL	CLOSL	Calculate error
	B	8	0			Adjust pulse width
	C	8	1			
	D	C	D	CALL	DWORD	Display error
	E	D	1			and voltage request
	F	0	2			
8 2E	0	F	1	POP	PSW	(A)← input voltage
	1	2	1	LXI	H, 83A0	Memory address
	2	A	0			for FILTR
	3	8	3			
	4	C	D	CALL	FILTR	
	5	7	0			(L)← raw voltage
	6	8	2			(H)← filter voltage
	7	1	1	LXI	D, 83FF	
	8	F	F			
	9	8	3			
	A	C	D	CALL	DWD2	
	B	D	4			
	C	0	2			
	D	C	3	JMP	82D0	
	E	D	0			
	F	8	3			
3	0					
	1			IDENTICAL		TO
	2			FIGURE	6-23a	
	3					
	4					
	5					
	6					
	7					
	8					Figure 6-28a

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

PWM - SUBROUTINE KYTIM

ADDR		CODE				
8 10	0	00	NOP			
	1	CD	CALL	ENTWD	(A) ← Command
	2	46			(HL) ← total period
	3	03			or voltage
	4	EB	XCHG		(DE) ← data
	5	21	LXI	H, 8108	Address dispatch
	6	08			table -10H
	7	81			
	8	85	ADD	L	Add command
	9	6F	MOV	L, A	
	A	6E	MOV	L, M	(ST) ← dispatch address
	B	E5	PUSH H	H	
	C	F3	DI		
	D	DB	IN	PORT 1A	
	E	04			Set Port 1A6 high
	F	F6	ORI	40	for scope trigger
8 11	0	40			without changing
	1	FB	EI		Port 1A7
	2	D3	OUT	PORT 1A	
	3	04			
	4	AF	XRA	A	Clear A and CY
	5	00	NOP		
	6	00	NOP		
	7	C9	RET		To dispatch address
	8	40	MEM		
	9	50	REG		
	A	17	ADDR		Trigger scope
	B	21	STEP		Store desired voltage
	C	38	RUN		Set total period
	D	20	NEXT		Store voltage, set width
	E	45	BRK		
	F	47	CLR		
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				Figure 6-28b

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

KYTIM - NEXT, STEP, RUN

ADDR	CODE		Instruction	Operand	Comment
8120	3	7	STC		NEXT (mark with CY)
8121	2	1	LXI	H, 8000	STEP (CY clear)
2	0	0			Address data log
3	8	0			and load with its
4	7	5	MOV	M, L	own address
5	E	B	XCHG		
6	2	2	SHLD	83A6	
7	A	6			
8	8	3			
9	D	0	RNC		Exit if STEP
A	1	1	LXI	D, FFEE	To subtract 12H
B	E	E			(0.18 volts) from
C	F	F			desired voltage
D	1	9	DAD	D	(HL) ← $v - V_0$
E	2	9	DAD	H	(HL) ← pulse width
F	0	0	NOP		
8130	2	2	SHLD	83A4	Store calculated
1	A	4			pulse width
2	8	3			as integral
3	C	D	CALL	LDT1	Load Timer 1
4	6	0			with calculated
5	8	1			pulse width
6	C	9	RET		
7	0	0	NOP		
8138	7	B	MOV	A, E	RUN
9	D	3	OUT	TIM0	Load Timer 0
A	1	4			for total period
B	7	A	MOV	A, D	
C	D	3	OUT	TIM0	
D	1	4			
E	C	9	RET		
F	0	0	NOP		
8 0					
1			IDENTICAL		TO
2			FIGURE	6-18i	
3					
4					
5					
6					
7					
8					Figure 6-28c

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

KYTIM - MEM, BRK, CLR, REG

ADDR		CODE		MNEMONIC	OPERAND	COMMENTS
8 14	0	E	B	XCHG		MEM
	1	2	2	SHLD	83A8	Store gains for CLOSL
	2	A	8			
	3	8	3			
	4	C	9	RET		
814	5	3	E	MVI	A, C9	BRK - Open loop
	6	C	9			Enter RET in INTEG
814	7	3	2	STA	81C0	CLR - Close loop
	8	C	0			Enter NOP in INTEG
	9	8	1			
	A	2	1	LXI	H, 8000	Start a new log
	B	0	0			
	C	8	0			
	D	7	5	MOV	M, L	
	E	C	9	RET		
	F	0	0	NOP		
8 15	0	D	3	OUT	CNT2	REG - Force output low
	1	0	F			Disable Timer 0 interrupt
	2	C	D	CALL		Monitor function
	3	4				Delay ~ 2 milliseconds
	4	0	2			
	5	2	1	LXI	H, 8000	Start a new log
	6	0	0			
	7	8	0			
	8	7	5	MOV	M, L	
	9	7	6	HLT		Wait for Timer!
	A	3	E	MVI	A, 01	Enable and clear
	B	0	1			Timer 0 interrupt
	C	D	3	OUT	CNT2	
	D	0	F			
	E	C	9	RET		
	F	0	0	NOP		
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					Figure 6-28d

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

LDT1 LOAD TIMER 1

ADDR	CODE		LABEL	INSTRUCTION		COMMENTS
8 16 0	2	B		DCX	H	LDT1
1	7	C		MOV	A, H	Test pulse width
2	B	7		ORA	A	for zero or negative
3	2	3		INX	H	
4	F	8		RMINUS		Exit if ≤ 0
5	7	D		MOV	A, L	Load Timer 1
6	D	3		OUT	TIM1	with charging
7	1	5				pulse width
8	7	C		MOV	A, H	
9	D	3		OUT	TIM1	
A	1	5				
B	C	9		RET		
C						
D						
E						
F						
8 0						
1						
2						
3						
4						
5						
6						
7						
8						
9						
A						
B						
C						
D						
E						
F						
8 0						
1						
2						
3						
4						
5						
6						
7						
8						Figure 6-28e

PWM - SUBROUTINE CLOSL - VERSION 2

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR	CODE		INSTRUCTION			
8180	2A	LHLD	83A6	CLOSL
1	A6			(L) ← desired voltage
2	83			at entry (A) = measured
3	95	SUB	L	(A) ← −Error
4	67	MOV	H, A	(H) ← −Error
5	C8	RZ		Exit if error = 0
6	E5	PUSH	H	Save display data
7	2F	CMA		
8	4F	MOV	C, A	
9	3F	CMC		(BC) ← +Error
A	9F	SBB	A	as two bytes
B	47	MOV	B, A	
C	03	INX	B	
D	2A	LHLD	83A4	(HL) ← old integral
E	A4			
F	83			
8190	CD	CALL	INTEG	Calculate and
1	C0			store new integral
2	81			(HL) ← (HL) + (BC)/2ⁿ
3	CD	CALL	PROPG	Add proportional term
4	E0			(HL) ← (HL) + K*(BC)
5	81			
6	CD	CALL	LDT1	Load Timer 1 with
7	60			new pulse width
8	81			
9	E1	POP	H	Recover display data
A	C9	RET		
B				
C				
D				
E				
F				
8 0				
1				
2				
3				
4				
5				
6				
7				
8				

Figure 6-28f

ADDR		CODE	INSTRUCTION	COMMENT
8 1C	0	00	NOP/RET	Stored by keyboard entry
	1	EB	XCHG	(DE) ← old integral
	2	69	MOV L, C	(L) ← low byte error
	3	3A	LDA 83A8	(A) ← n for
	4	A8		Error/2^n
	5	83		
	6	67	MOV H, A	(H) ← n
8 1C	7	25	DCR H	Decrement count
	8	FA	JMIN 81D5	Exit loop after 0
	9	D5		
	A	81		
	B	78	MOV A, B	Set CY if error
	C	1F	RAR	negative
	D	7D	MOV A, L	Shift low byte
	E	1F	RAR	error right
	F	6F	MOV L, A	
8 1D	0	B7	ORA A	Test for error = 0
	1	C2	JNZ 81C7	Loop unless
	2	C7		shifted error = 0
	3	81		If shifted error = 0
	4	2C	INR L	make it 01
81D	5	60	MOV H, B	(H) ← high byte error
	6	19	DAD D	(HL) ← new integral
	7	7C	MOV A, H	Test for negative
	8	B7	ORA A	integral
	9	F8	RMINUS	Exit if negative
	A	22	SHLD 83A4	Store new integral
	B	A4		
	C	83		
	D	C9	RET	
	E	00	NOP	
	F	00	NOP	
8	0			
	1		ENTER WITH	
	2		(BC) = + ERROR (B=00 OR FF)	
	3		(HL) = OLD INTEGRAL	
	4		RETURN	
	5		(BC) PRESERVED	
	6		(DE) = OLD INTEGRAL	
	7		(HL) = NEW INTEGRAL	
	8			

Figure 6-28g

PWM - SUBROUTINE PROPG

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		Instruction	Comment
81E	0	3	A	LDA 83A9	(A) ← K
	1	A	9		for K error
	2	8	3		
81E	3	3	D	DCR A	Decrement K
	4	F	8	RMINUS	Exit after zero
	5	0	9	DAD B	Add error into
	6	C	3	JMP 81E3	pulse width
	7	E	3		from INTEG and
	8	8	1		loop until done
	9				
	A				
	B				
	C			ENTER WITH	
	D			(BC) = + ERROR	
	E			(HL) = PULSE WIDTH	
	F			UPDATED BY INTEG	
8	0				
	1			RETURN	
	2			(BC) PRESERVED	
	3			(DE) PRESERVED	
	4			(HL) = NEW PULSE WIDTH	
	5			= INTEG + K ERROR	
	6				
	7				
	8				
	9				
	A				
	B				
	C				
	D				
	E				
	F				
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				Figure 6-28h

6.3.4 Experiments with PI Control

The effect of proportional plus integral control is to achieve rapid response to a disturbance or to a change in the desired output value without the objectionable overshoot and oscillation associated with pure integral control. If an oscilloscope is available the observations are easily made; lacking an oscilloscope the data can be logged and plotted. Waveform photographs are presented here for several of the experiments.

6.3.4.1 Open Loop Control

To demonstrate that open loop control is still available in the system with version 2 of CLOSL and INTEG, enter the following data and commands:

400,RUN	Set 0.5 ms total period
MEM	Set zero proportional gain
BRK	Disable integral control
40,NEXT	Set minimum output
C8,NEXT	Request 2.0 volts

The voltage will rise to its initial value of about 1.5 volts.

Adjust the SENSE pot and observe that the voltage changes.

Restore the SENSE pot to the full left position.

6.3.4.2 Pure Proportional Control

Set pure proportional control by:

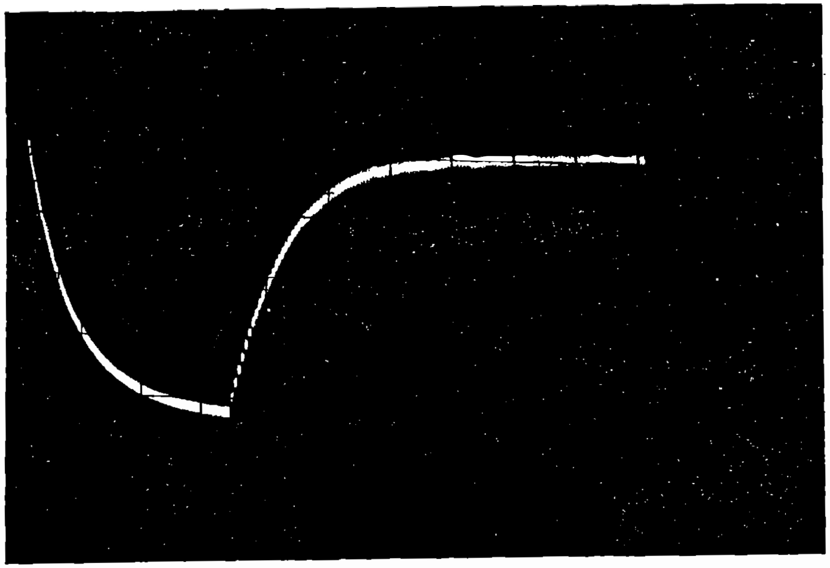
400,RUN	Set 0.5 ms total period
MEM	Set zero proportional gain
BRK	Disable integral control
64,NEXT	Request 1.0 volts
100,MEM	Set proportional gain = 1

The output voltage will rise somewhat as proportional control increases the charging time above the fixed value calculated by NEXT.

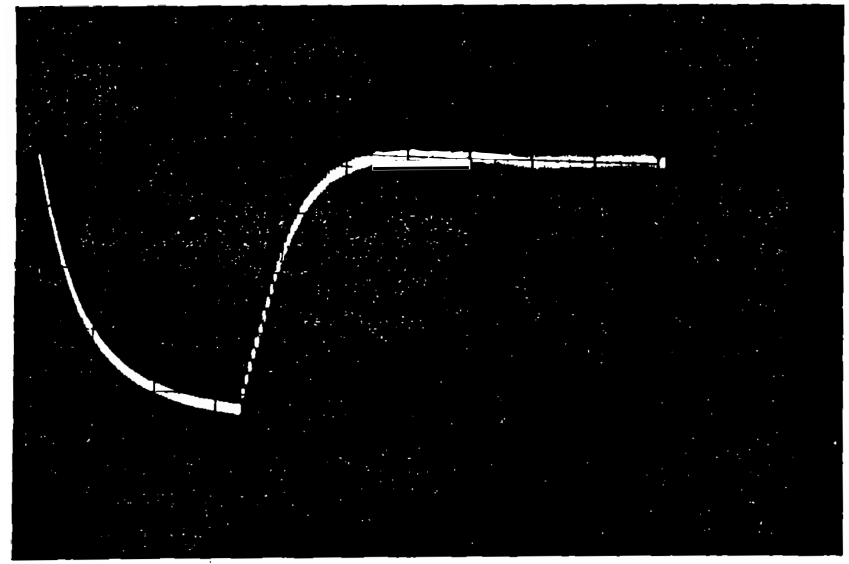
Observe the voltage generated with different proportional gains.

COMMAND	GAIN	OUTPUT VOLTAGE
MEM	0	________
100,MEM	1	________
200,MEM	2	________
300,MEM	3	________
400,MEM	4	________
600,MEM	6	________
800,MEM	8	________
1000,MEM	16	________

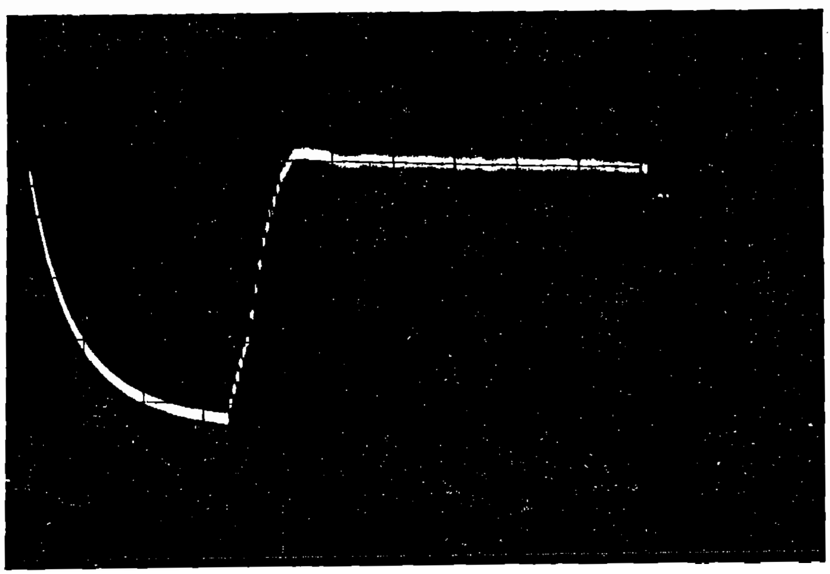
At a sufficiently high gain the multiplied error signal will lead to an unacceptable pulse width which will be rejected by LDT1.



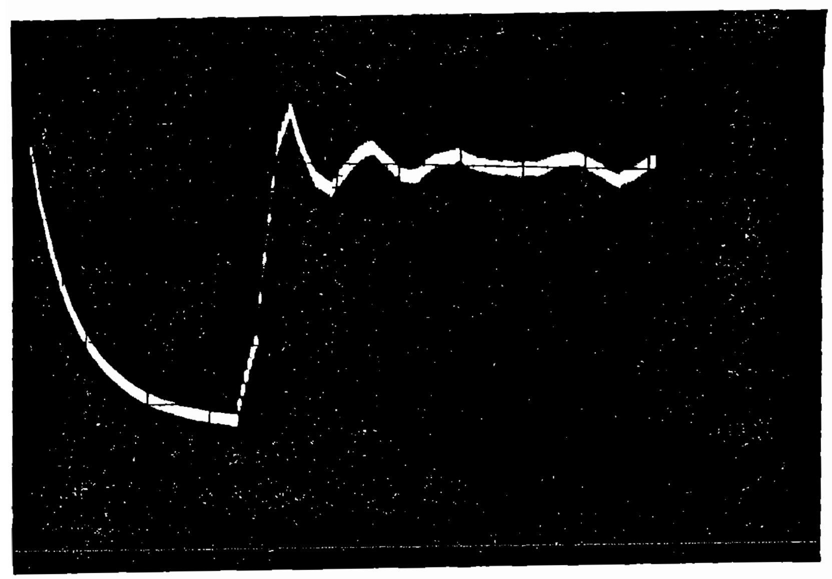
Proportional Gain = 0 (Open Loop



Proportional Gain = 1



Proportional Gain = 4



Proportional Gain = 8

RESPONSE WITH PROPORTIONAL CONTROL

FIGURE 6-29

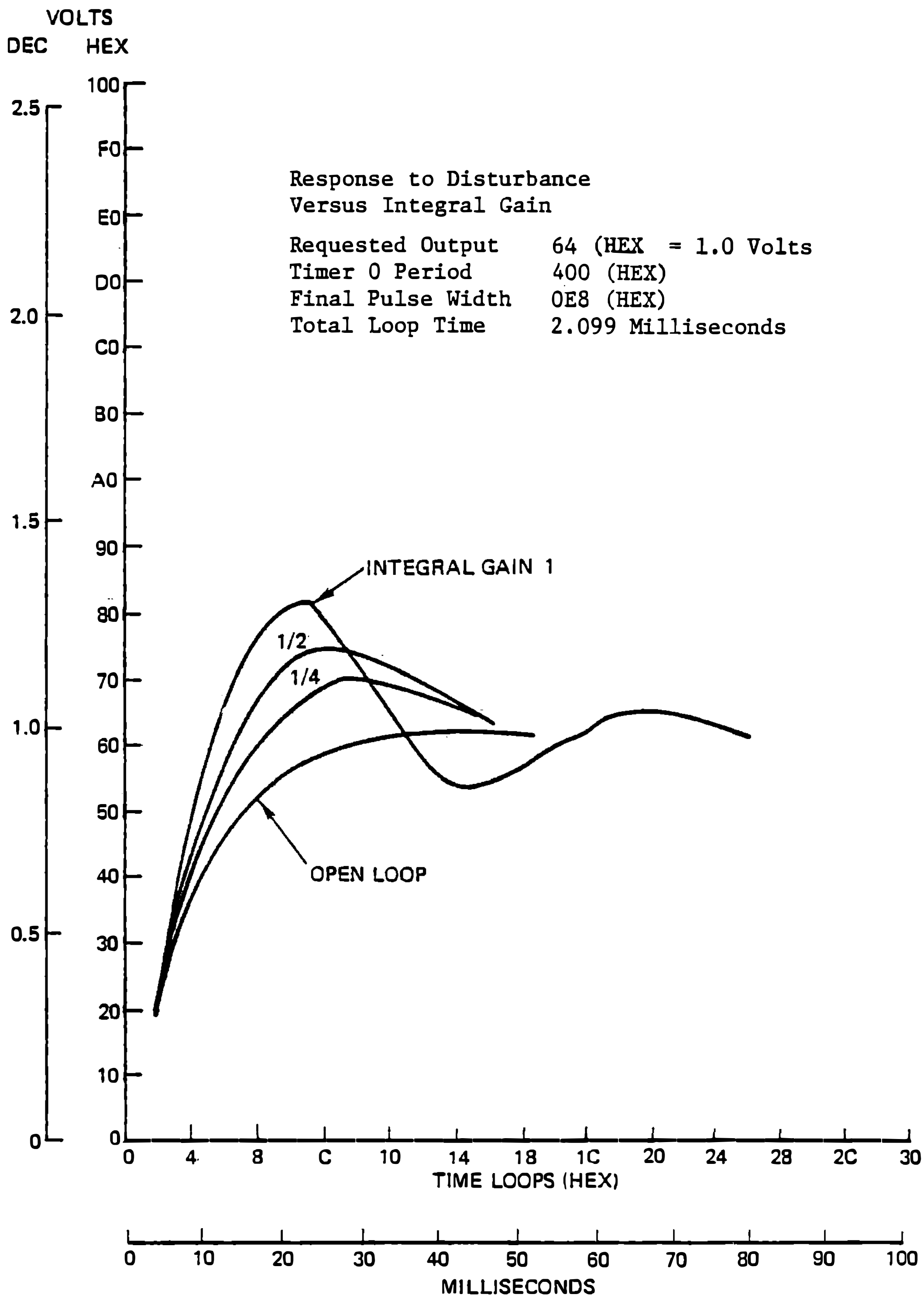
If an oscilloscope is available, do the following experiment. Set the oscilloscope to 0.2 volts/division, 5 milliseconds per division, to observe the rise time.

64,NEXT	Request 1.0 volt
MEM	Set integral gain = 1 and proportional gain = 0
CLR	Set closed loop to adjust the pulse width
BRK	Set open loop
REG	Force output low.

Observe the response to the disturbance.

100,MEM	Set proportional gain = 1
REG	Force output low

Repeat with successively higher proportional gain values to observe the response. Figure 6-29 shows several response waveforms. The speed of response increases as the proportional gain is increased. With high gain substantial overshoot and oscillation appear.



Response Versus Integral Gain

Figure 6-30

6.3.4.3 Pure Integral Control

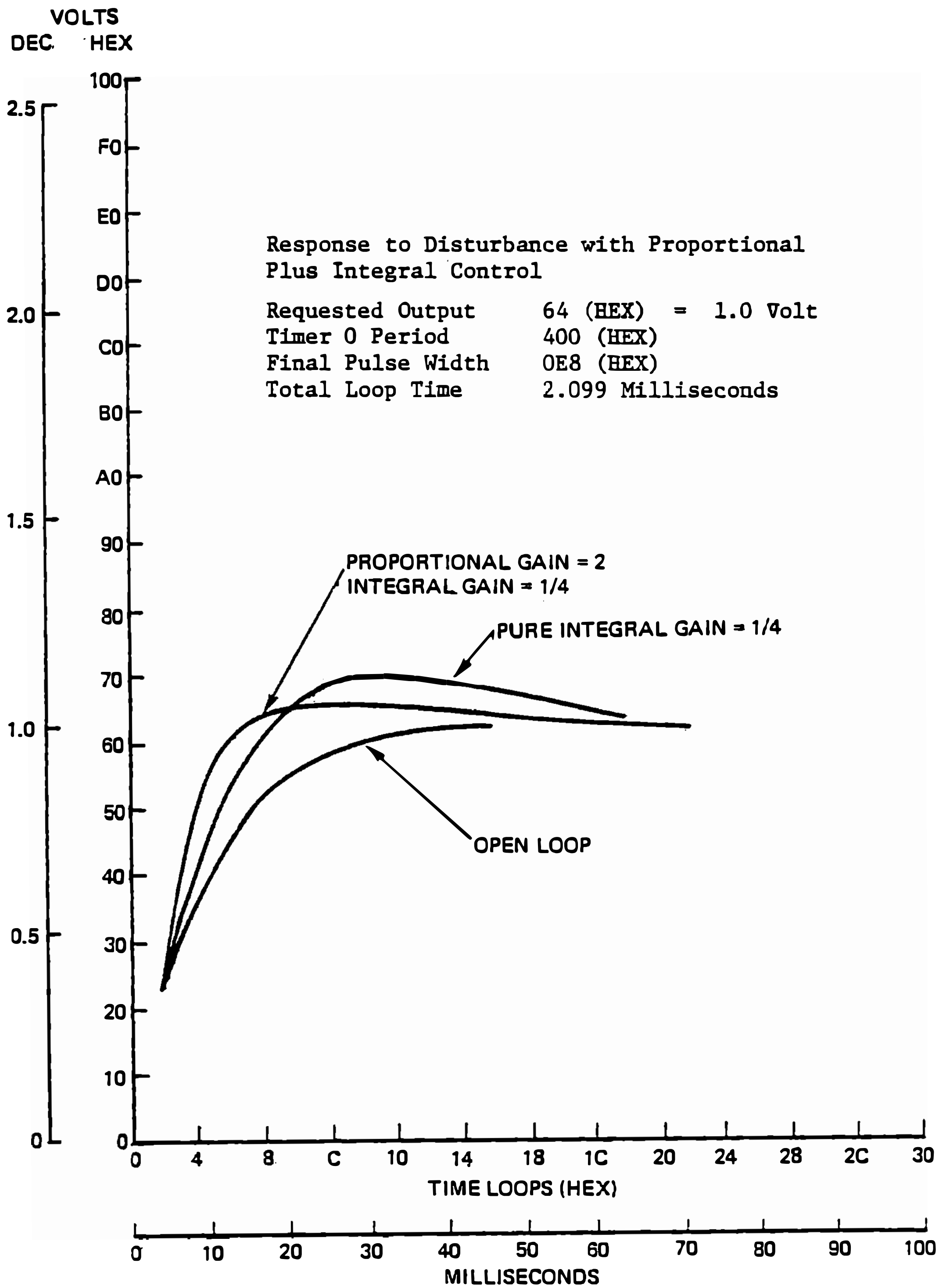
The effect of integral control on the response waveform has been observed previously. The following experiments shows the effect of reduced integral gain. If you have an oscilloscope make all of these observations. Otherwise, plot the response from the data log for gain = 1 and gain = 1/4. Enter these commands:

MEM	Set proportional gain = 0 and n = 0 for integral gain = 1
CLR	Enable integral control
64,NEXT	Request 1.0 volt
REG	Force output low

Observe or plot the response.

1,MEM	Set integral gain = 1/2
REG	Observe response
2,MEM	Set integral gain = 1/4
REG	Observe or plot response
3,MEM	Set integral gain = 1/8
REG	Observe response
BRK	Set open loop
REG	Observe response

Continue to decrease the gain and observe the response. Figure 6-30 shows responses for selected gains, and for open loop operation.



Proportional Plus Integral Response

Figure 6-31

6.3.4.4 Proportional Plus Integral Control

Directly after the preceding experiment observe or plot the response with proportional plus integral control.

202,MEM	Set proportional gain = 2 and integral gain = 1/4
REG	Observe or plot response

This demonstrates an effective control system. Less overshoot occurs than with pure integral control using the same gain because the higher gain proportional control promptly corrects the overshoot. A fast response to the disturbance is observed. Figure 6-31 compares the response obtained here with some of our earlier results.

6.3.4.5 Response to Voltage Request

The preceding observations of response time have all maintained a constant desired voltage, forcing the output low under open loop control. Observe the response to a change in desired voltage:

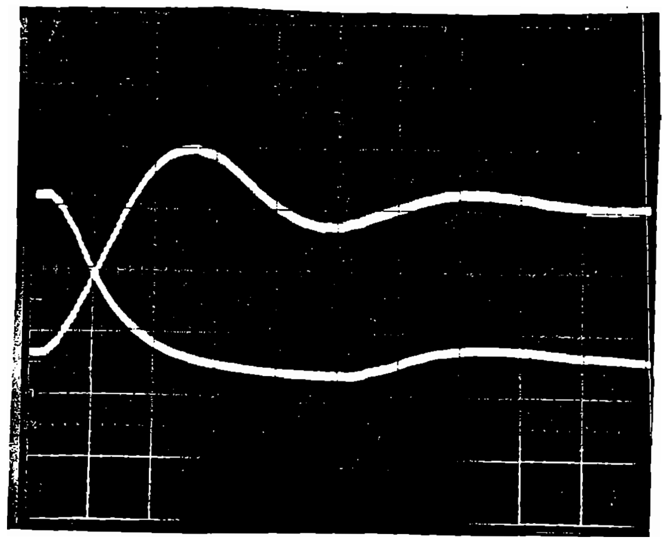
BRK,MEM	Set open loop operation
C8,NEXT	Request 2.0 volts
xxx,RUN	Set total period to obtain requested voltage
40,NEXT	Request minimum output and observe response
C8,NEXT	Request 2.0 volts and observe output

INCREASING PROPORTIONAL GAIN

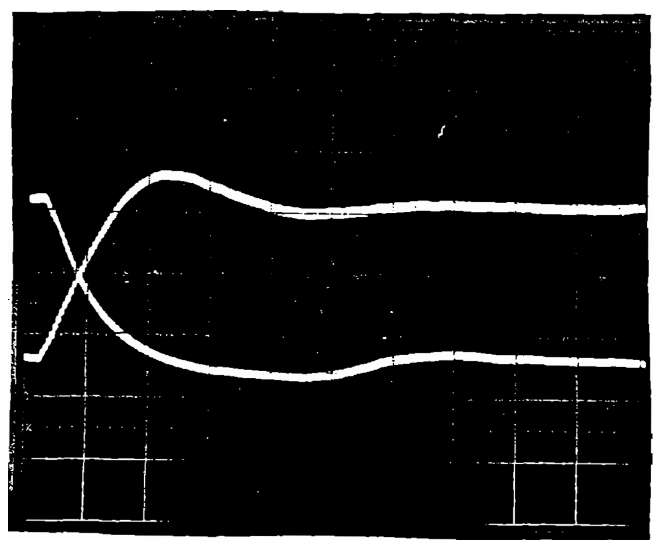
Proportional Gain = 0
Integral Gain = 1

Voltage was C8
Voltage requested 40

Voltage was 40
Voltage requested C8



Proportional Gain = 1
Integral Gain = 1



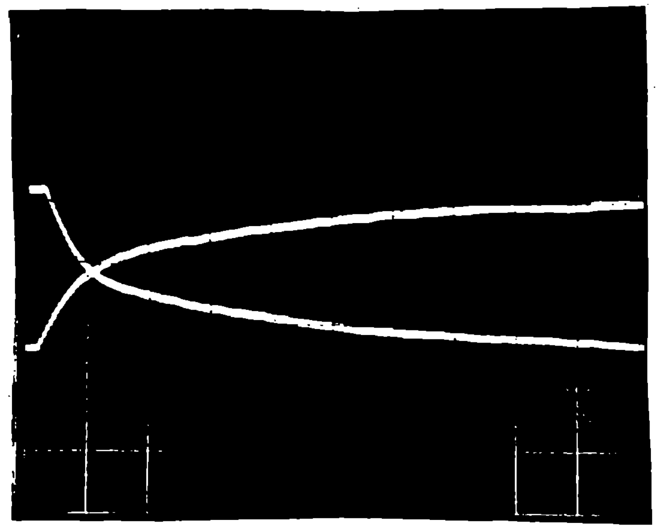
Response to Voltage Request

Figure 6-32a

INCREASING INTEGRAL GAIN

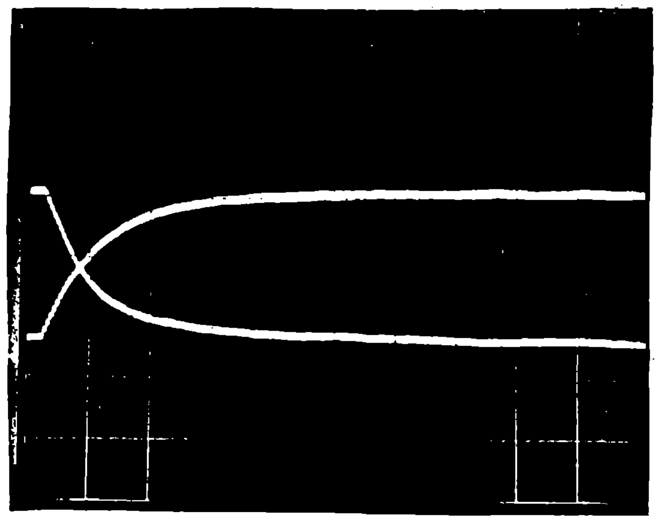
Proportional Gain = 2

Integral Gain = ¼



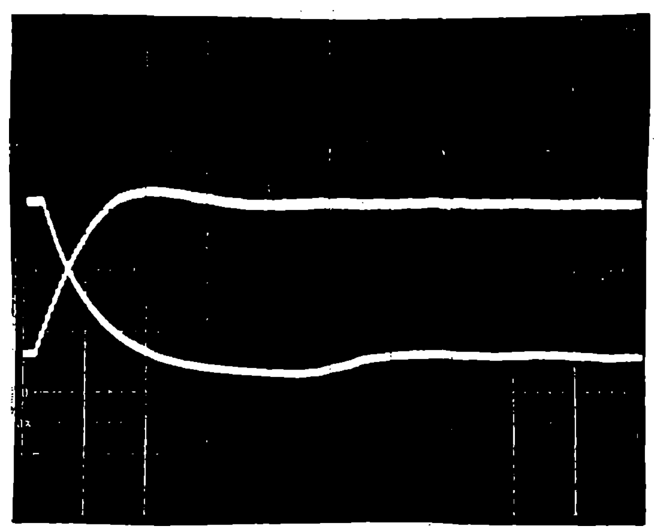
Proportional Gain = 2

Integral Gain = ½



Proportional Gain = 2

Integral Gain = 1



Response to Voltage Request (continued)

Figure 6-32b

Now set proportional plus integral control by:

202,MEM	Set proportional gain =2 and integral gain = 1/4
CLR	Enable integral control
50,STEP	Request minimum output and observe response
C8,STEP	Request 2.0 volts and observe response

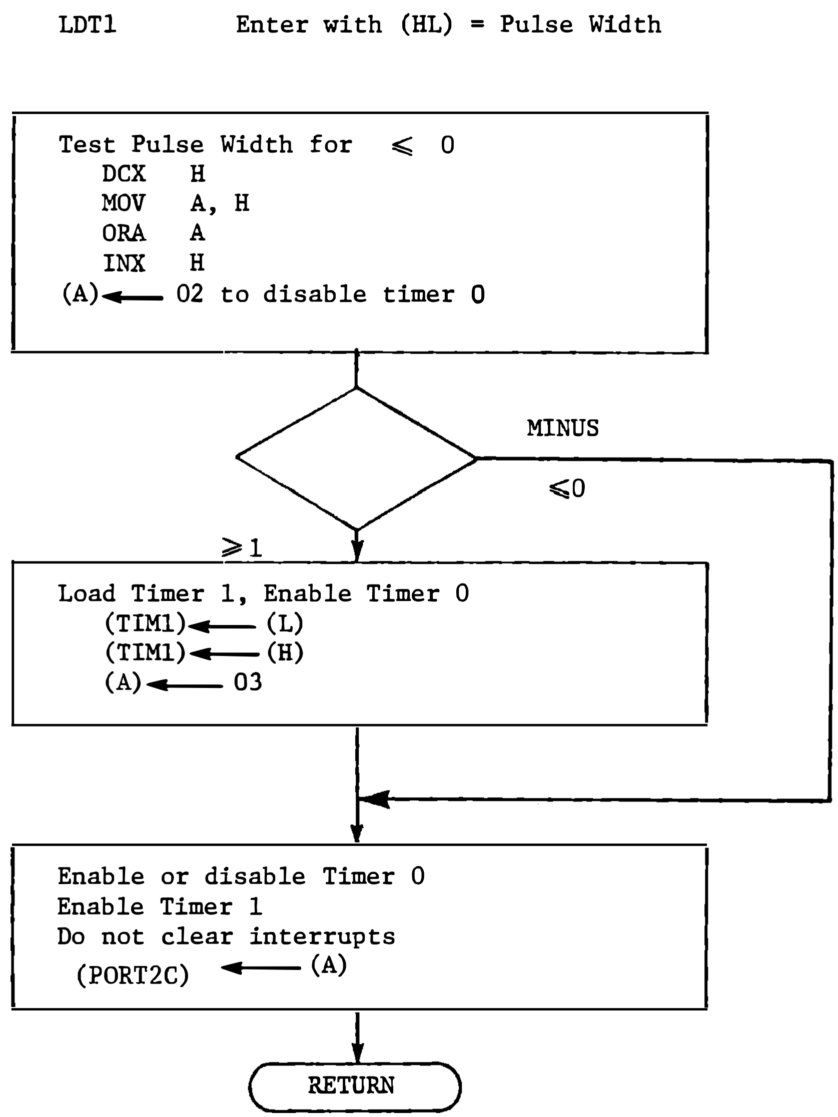
Results of this test are shown in Figure 6-32. Note that with open loop control the rise and fall are similar, but with closed loop control they are distinctly different.

6.3.5 Full Scale Control and Overflow

In subroutine LDT1 we have protected against loading Timer 1 with an unintended long time period when the calculated pulse width is zero or negative. The reaction when this occurs is to leave the preceding value of the pulse width unchanged. This has been acceptable but is not the best arrangement. It would be better to recognize the condition and disable charging altogether if the measured voltage is so much greater than the desired value that proportional control cannot operate correctly. This would allow the voltage to drop more quickly, and also remove the limitation imposed by the time taken in the interrupt service routines.

(Recall that Timer 0 interrupt starts charging the capacitor by setting Port 1A7 high, and no matter how small a value is loaded to Timer 1 the output will remain high until the Timer 0 interrupt is finished and the Timer 2 interrupt sets the output low. This takes 66 clock periods after OUT PORT1A in Timer 0 service, plus 50 clock periods for the Timer 1 interrupt and processing through its OUT PORT1A. This gives a minimum pulse of 57 microseconds and an output of 0.8 volts if the total period is 300 hex).

When we abandon proportional control and set a minimum (or maximum) control force we are employing "full scale control". We can do this here by changing subroutine LDT1.



LDT1 with Full Scale Control

Figure 6-33

6.3.5.1 LDT1 with Full Scale Control

Revise subroutine LDT1 according to Figure 6-33. With this program the calculated pulse width is tested as before for a negative or zero value. If the width is greater than zero it is loaded into Timer 1 and the interrupt from Timer 0 is enabled. If the width is zero or negative the timer is not loaded, and Timer 0 is disabled so that no charging occurs. Note that the enable or disable is not to clear the interrupt, so it is done by writing to Port 2C, not to CNT2. If the bit set function were used to enable the interrupt it would occasionally occur just as the timer generated an interrupt, thereby inhibiting a charging pulse. With the method of writing to Port 2C, normal operation will be totally unaffected, since this does not reset the interrupt source flip flops. After a time of full scale control with Timer 0 disabled, the voltage will decrease, the calculated pulse width will again become acceptable and Port 2C0 will be set high. An interrupt will occur immediately. A charging pulse will start, but its duration will be random, depending on when the next Timer 1 interrupt occurs.

PWM - SUBROUTINE LDT1 - VERSION 2

ADDR	CODE			
8160	2B	DCX	H	Test calculated
1	7C	MOV	A, H	pulse width for
2	B7	ORA	A	zero or negative
3	23	INX	H	
4	3E	MVI	A, 02	Load (A) to disable
5	02			Timer 0 interrupt
6	FA	JMIN	8171	and jump if
7	71			pulse width ≤ 0
8	81			
9	7D	MOV	A, L	Pulse width > 0
A	D3	OUT	TIM1	Load Timer 1 for
B	15			new pulse width
C	7C	MOV	A, H	
D	D3	OUT	TIM1	
E	15			
F	3E	MVI	A, 03	Load (A) to enable
8170	03			Timer 0 interrupt
8171	D3	OUT	PORT 2C	Enable or disable
2	0E			Timer 0 interrupt
3	C9	RET		and enable
4				Timer 1 interrupt
5				
6				
7		REVISED LDT1		
8		FOR FULL SCALE CONTROL		
9				
A		ENTER WITH		
B		(HL) = CALCULATED		
C		PULSE WIDTH		
D		IF (HL) > ZERO		
E		LOAD TIMER 1 AND		
F		ENABLE BOTH INTERRUPTS		
8 0				
1		IF (HL) ≤ ZERO		
2		DISABLE TIMER 0 TO		
3		INHIBIT CHARGING		
4				
5				
6				
7				
8				Figure 6-34

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

6.3.5.2 Full Scale Control Experiments

To test the ability of full scale control to generate a low output signal enter:

400,RUN	Set 0.5 ms total period
MEM	Set proportional gain = 0
	and integral gain = 1
CLR	Enable integral control
STEP (or NEXT)	Request zero output

Timer 0 interrupt will be disabled, so Port 1A7 will be continuously low. (Observe this in the LED). The output voltage will be determined by the division of Vg across the voltmeter resistance and the 10K resistor R2.

$$v = \frac{V_g R_m}{R_m + R_2}$$

Note that the voltage can be changed by switching scales on the voltmeter (which changes the voltmeter's resistance). If you disconnect the voltmeter the voltage measured by the A/D converter will be almost exactly equal to Vg, the zero offset voltage of the open collector driver of the output port. The voltmeter will reduce that voltage slightly if a 3 volt scale is used and significantly on more sensitive scales. (If you are using an electronic voltmeter with a high impedance input the voltmeter will not affect the output voltage at all).

Now, reset the computer and check the integral stored at (83A4,A5). It will be some value between zero and the output voltage that was achieved. After this value was stored, INTEG attempted to reduce it by the observed error. The result was negative so it was not stored. A lower integral gain would allow it to be reduced further. The negative (or zero) pulse width passed to LDT1 gave full scale control and a minimum output.

Run the program again and enter:

MEM	Set proportional gain = 0 and integral gain = 1
CLR	Enable integral control
STEP	Request zero output
BRK	Disable integral control

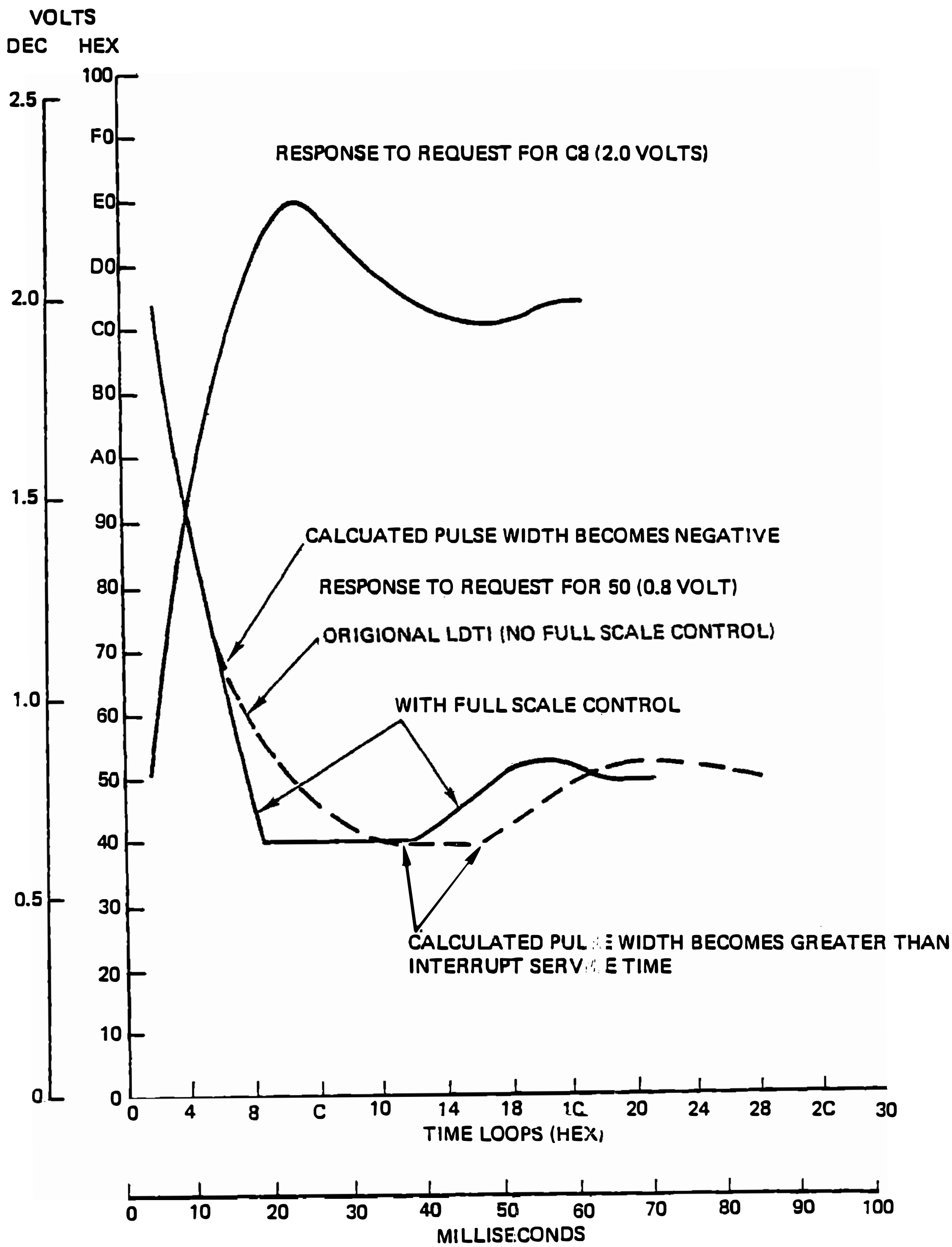
The voltage will rise to the minimum determined by the interrupt service processing. Without closed loop control no attempt is made to reduce the pulse width below zero, so LDT1 will enable Timer 0 interrupt. Enter:

100,MEM	Set proportional gain = 1

Now the proportional control will generate negative pulse widths and full scale control will be invoked.

Observe the response to voltage requests with various proportional and integral gains. Figure 6-35 compares the response to requests for 50 (hex) and C8, with the original version of LDT1 and the new version with full scale control. For this plot both gains are set to unity.

100,MEM	Set proportional gain = 1 and integral gain = 1
CLR	Enable integral control
C8,STEP	Request 2.0 volts
50,STEP	Request 0.8 volt and observe response
C8,STEP	Request 2.0 volts and observe response



Full Scale Response to Vol age Request

Figure 6-35

6.3.5.3 Maximum Output Full Scale Control

Clearly full scale control could also be exercised in the other direction. If proportional control could not achieve a desired voltage or demanded too great a pulse width, Timer 1 could be disabled to leave Port 1A7 on continuously. In fact, the present integral control system has full scale control, since the charging pulse width can be increased up to 7FFF, about 16 milliseconds or 32 times the 0.5 millisecond "total period". If the Timer 1 period is greater than the Timer 0 period, each new output pulse from Timer 0 retriggers Timer 1, whose count is then reloaded automatically and never reaches zero. No Timer 1 interrupts are generated and Port 1A7 stays high.

Full scale control is very commonly employed in proportional control systems where the designer recognizes a need for signals outside of the proportional range. Especially in pure proportional systems (i.e., no integral control) it is often desirable to use very high proportional gain to obtain fast response to small error signals. This usually limits the proportional range to fairly small error signals and demands full scale control for large errors.

6.3.5.4 Integral Overflow

We have limited the integral of error signals stored by INTEG to the range 0000 to 7FFF. This limitation tends to distort the results, since the lower limit is often encountered, but it is almost impossible to reach the upper limit. We are also wasting half of the range of a two byte variable. If this were important (which it is not) we could extend the range by allowing and correctly processing negative values of the integral.

At present INTEG refrains from storing the integral if it is negative. This applies the limits of 0000 to 7FFF. If this test is not applied the integral will temporarily go negative when the desired voltage is switched from a high value too a low value, giving a large negative error signal. This invokes full scale control. Provided that the measured voltage eventually becomes less than the desired voltage, giving a positive error signal, the integral will (after some time) become positive again and settle at a value that gives the correct pulse width.

To experiment with this, remove the following instructions from CLOSL and INTEG. (Replace each of them with NOP).

8186	E5	PUSH H	Save display data
8199	E1	POP H	Recover display data
81D9	F8	RM	Exit if integral <0

Removing PUSH H and POP H will cause CLOSL to return the calculated pulse width for display instead of the error signal and desired voltage. Removing RM will cause INTEG to store the integral regardless of its value. Now run the program with the following data and commands. (We will use pure integral control so that the displayed pulse width will be the integral value).

MEM	Set proportional gain = 0 and integral gain = 1
CLR	Enable integral control
FA,STEP	Request 2.5 volts
40,STEP	Request 0.64 volt

The two voltage requests may be repeated as often as necessary. After 40,STEP you will see an F appear momentarily in the left hand display digit, showing that the calculated pulse width has become negative. Thereafter the integral will become and remain positive.

20,STEP	Request 0.32 volt

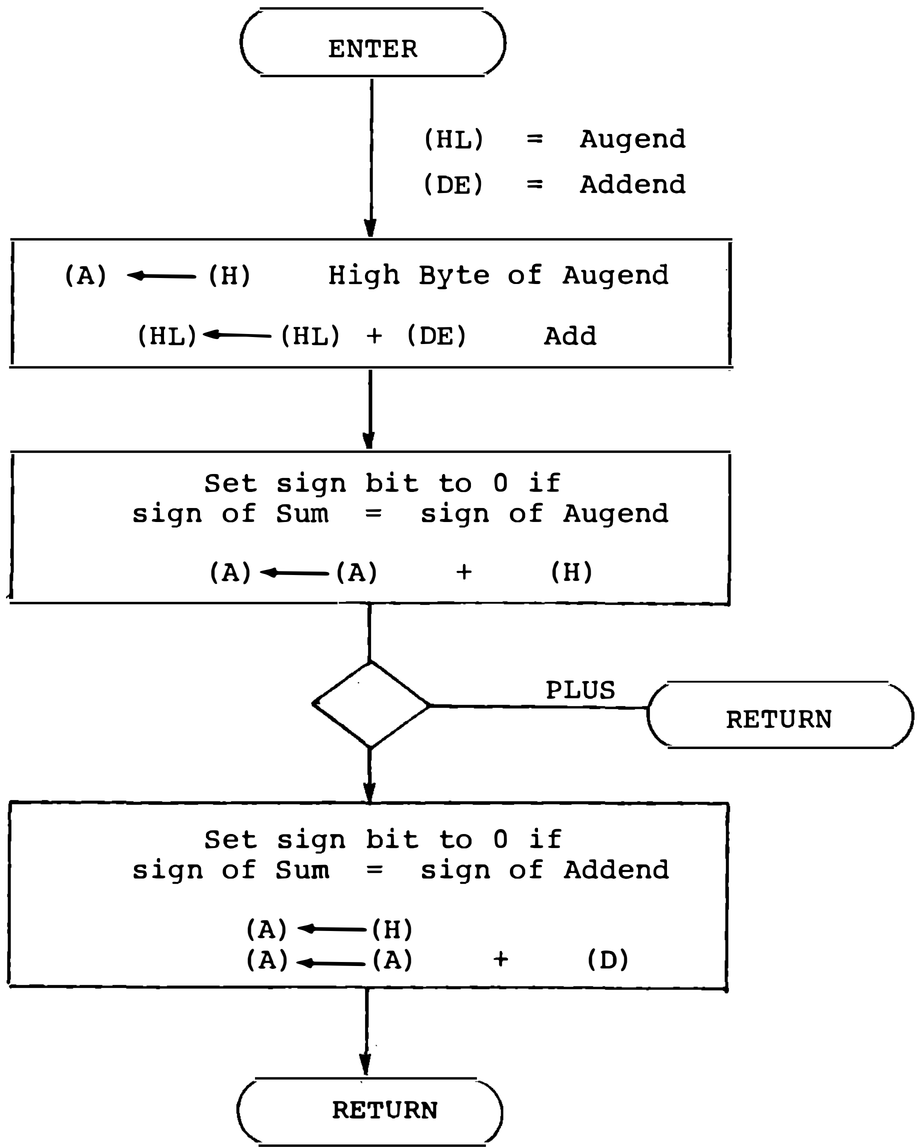
To reach this low a voltage full scale control must be invoked a large part of the time. The integral displayed at the left will show FFxx with an occasional appearance of 00xx.

STEP	Request zero output

Since a zero volt output cannot be achieved even with full scale control the error signal will always be negative. The integral will be repeatedly reduced from FFxx down to 80xx, then to 7Fxx where it suddenly appears to be a large positive value. Until this point, full scale control has kept Timer 0 interrupt disabled and Port 1A7 low. Now Timer 1 is loaded with a long pulse width (about 16 milliseconds) and Timer 0 is enabled. Port 1A7 is set high and the output rises above 2.55 volts. The large negative error signal is calculated, rapidly reducing the integral until it goes negative again. You can observe the peculiar behavior on the voltmeter or at the LED for Port 1A7. A computer, like a person, may go crazy when presented with an impossible task and no escape mechanism. The program works well as long as it is able to achieve its objective, but it needs the protection of the RM instruction for the impossible request.

Restore the three instructions that were deleted for this experiment.

Recognize from this experiment that a negative integral is perfectly acceptable; the fault occurs when a negative error added to a negative integral generates a positive result. This is referred to as arithmetic overflow, and can be detected. The result of an addition should always have the same sign as either the augend or the addend. If both are positive but their sum is negative, or vice versa, overflow has occurred. Figure 6-36 shows a simple procedure for testing the result of a double precision add. Replace DAD D and the instructions that test for negative results in INTEG with CALL ADTOV (81FO). Enter subroutine ADTOV (Figure 6-37). Now either positive or negative integrals will be stored unless overflow has occurred. There will be a difference in the response to large changes in the requested voltage, and the program is more satisfying aesthetically, but you will see that the difference is not important.



Subroutine ADTOV

Double Precision Add and Test for Overflow

Figure 6-36

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

PWM - SUBROUTINE INTEG - VERSION 3

ADDR		CODE	Instruction	Comment
8 1C	0	00	NOP/RET	
	1	EB	XCHG	(DE)← old integral
	2	69	MOV L, C	(L)← low byte error
	3	3A	LDA 83A8	(A)← n for
	4	A8		Error/2^n
	5	83		
	6	67	MOV H, A	(H)← n
81C	7	25	DCR H	Decrement count
	8	FA	JMIN 81D5	Exit loop after 0
	9	D5		
	A	81		
	B	78	MOV A, B	Set CY if error
	C	1F	RAR	negative
	D	7D	MOV A, L	Shift low byte of
	E	1F	RAR	error right
	F	6F	MOV L, A	
8 1D	0	B7	ORA A	Test for error = 0
	1	C2	JNZ 81C7	Loop unless 0
	2	C7		
	3	81		If shifted error = 0
	4	2C	INR L	Make error = 01
81D	5	60	MOV H, B	(H)← high byte error
	6	CD	CALL ADTOV	Add (HL)←(HL)+(DE)
	7	F0		Return MINUS if
	8	81		arithmetic overflow
	9	F8	RMINUS	
	A	22	SHLD 83A4	Store new integral
	B	A4		
	C	83		
	D	C9	RET	
	E	00	NOP	
	F	00	NOP	
8	0			
	1		ENTER WITH	
	2		(BC) = +	ERROR (B=00/FF)
	3		(HL) = OLD INTEGRAL	
	4		RETURN	
	5		(BC) PRESERVED	
	6		(DE) = OLD INTEGRAL	
	7		(HL) = NEW INTEGRAL	
	8			Figure 6-37a

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADTOV - ADD AND TEST OVER FLOW

ADDR		CODE		
8 1F	0	7C	MOV	A, H
	1	19	DAD	D
	2	AC	XRA	H
	3	F0	RPLUS	
	4	7C	MOV	A, H
	5	AA	XRA	D
	6	C9	RET	
	7			
	8			
	9		ENTER	
	A		(DE) =	OLD INTEGRAL
	B		(HL) =	ERROR / 2^n
	C			
	D		RETURN	
	E		(HL) =	NEW INTEGRAL
	F			(DE) + (HL)
8	0		MINUS	FLAG SET IF
	1		ARITHMETIC OVERFLOW	
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			
	A			
	B			
	C			
	D			
	E			
	F			
8	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			

Figure 6-37b

6.4 PROPORTIONAL - INTEGRAL - DIFFERENTIAL CONTROL

Consider a system with substantial inertia, such as steering a ship or aircraft. When the aiming point is changed or a sudden disturbance such as a wind gust or wave causes a large error signal, the proportional term in the control equation generates a large (possibly full scale) control force. The inertia prevents an instantaneous response, so after a brief time this force is further increased by the integral term. Now as the ship starts to respond the proportional term GpE decreases but the integral term $Gi\int E$ is still increasing. The ship now swings toward the aiming point, and its inertia will carry it beyond the desired point. An error in the opposite direction appears, the proportional term in the control equation becomes negative, and eventually the ship settles on its new course, provided the control system is stable. There may be significant and undesirable oscillations before the new course is achieved if high gains are used in the control system, and it is possible to have unstable operation where the oscillations are maintained indefinitely.

Differential or rate control can be applied to detect and respond to the fact that the ship is approaching the desired aiming point. As it begins to turn, even though the error signal may still be substantial, it is observed that the error is decreasing. This implies that a smaller control force should now be applied, or even that the control force should be reversed to overcome the ship's inertia. The control equation becomes:

$$F = G_p E + G_i \int E + G_d \Delta E$$

The differential term is not limited to reducing the control force as the error decreases, but also adds to the control force when the error is observed to increase. In a system subjected to disturbances the differential term may dominate the result, maintaining such small errors that the proportional term is generally very small and the integral term is almost constant.

Applying the differential control to a system having as little inertia as the capacitance in our PWM voltage control problem has very little effect. For the student interested in pursuing Proportional-Integral-Differential Control (PID) it is suggested that the filtered voltage returned by FILTR be used as the input to the closed loop control equations. Here FILTR gives the effect of a system with large inertia.

6.5 SUMMARY

In the preceding sections the most important concepts of feedback control systems have been demonstrated. Although controlling the voltage on a capacitor is a trivial and perhaps unexciting example, it gave us an easy way to observe and measure the behavior of the system. We have seen the response to a disturbance with proportional control; the need for integral control to provide a steady state force when external conditions are variable; and some of the problems such as overshoot, oscillation, and arithmetic overflow. We have also seen the use of the microcomputer to monitor its own performance. Chapter 7 will apply the same principles to control of a motor.

MICROCOMPUTER INTERFACING WORKBOOK

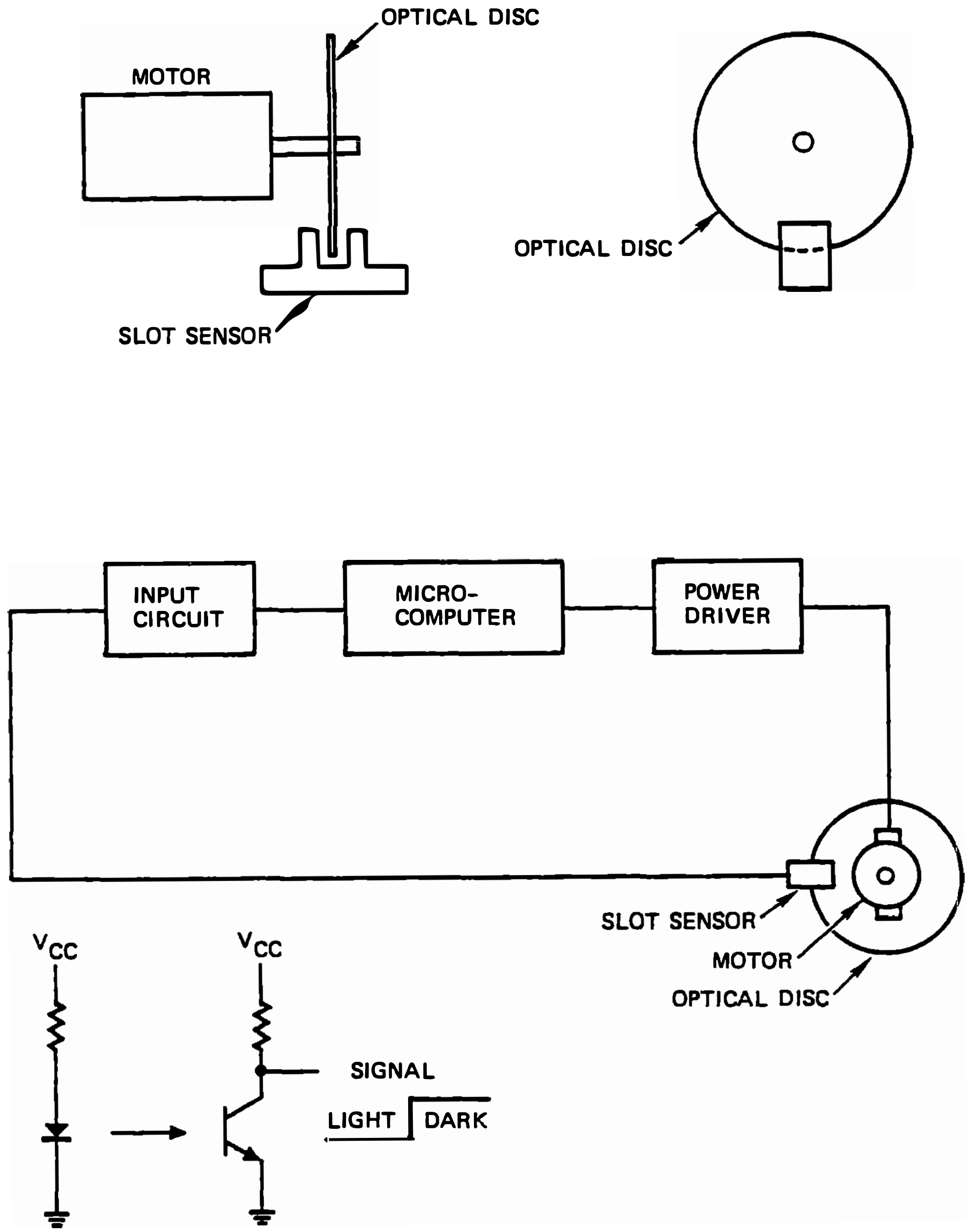
CHAPTER 7

MOTOR CONTROL

This page intentionally left blank.

7. MOTOR CONTROL

Power to a dc motor is to be controlled to set its speed. We will develop a technique to measure speed, using open loop control of power by pulse width modulation; then we will close the loop using proportional plus integral control. Both the program itself and our development of it will resemble the pulse width modulated voltage control of Chapter 6.



Motor and Slot Sensor

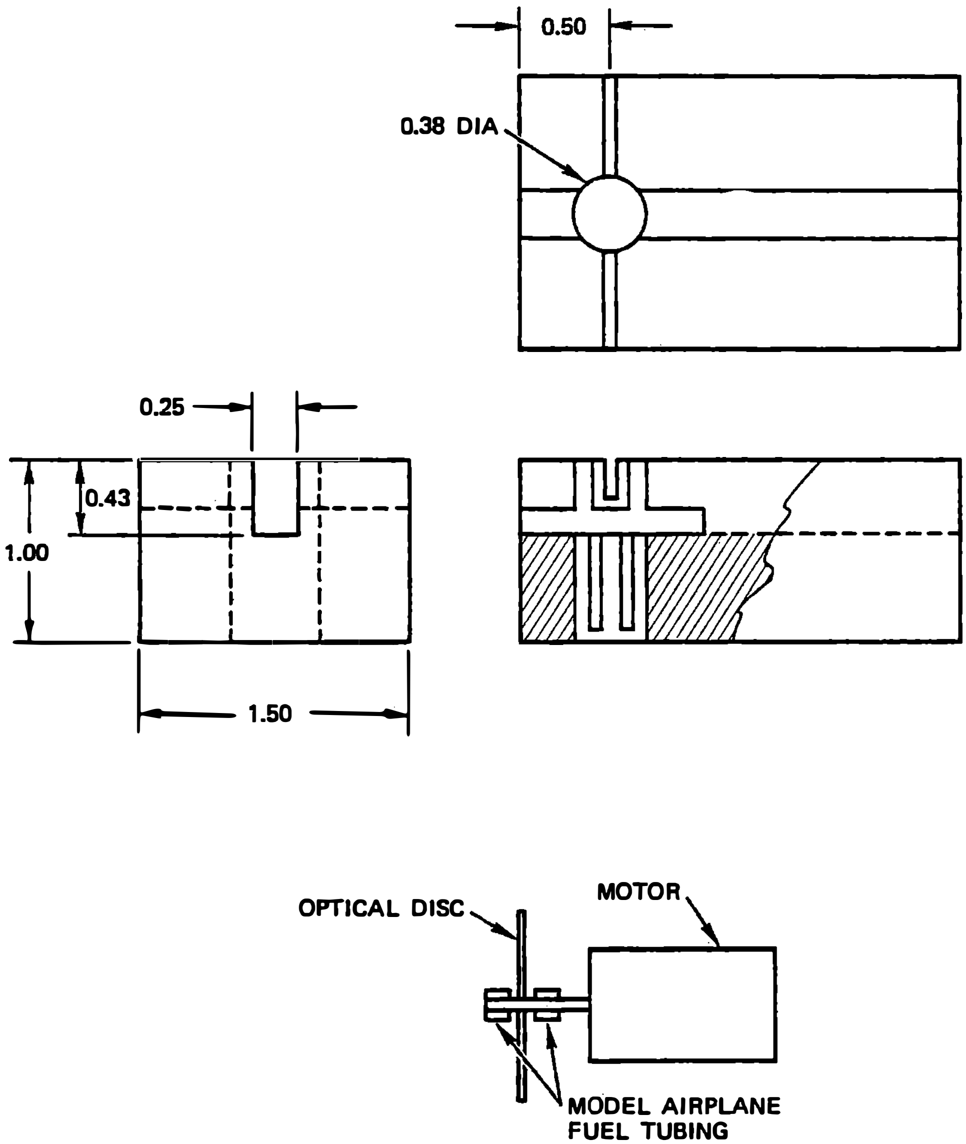
Figure 7-1

7.1 OPTICAL DISC AND SLOT SENSOR

Motor speed will be measured by observing an optical disc with transparent and opaque segments rotating between a light emitting divide and a phototransistor. Figure 7-1 suggests the physical arrangement and the system block diagram.

The "slot sensor" contains an infrared light emitting diode aimed at a phototransistor across a gap of 0.1 inch. When power is applied to the LED and the phototransistor as shown in Figure 7-1, the infrared light falling on the base of the phototransistor turns it on just as base current would in a normal transistor. The transistor current then drives the output signal low. When the light is blocked the phototransistor is turned off and the output signal becomes high.

If your configuration includes the Integrated Experiment Assembly, the motor and slot sensor are mounted and connected, with an amplifier for the slot sensor signal. In this case the connections and tests described in Sections 7.1.1 and 7.1.2 are not required. Read the material of Section 7.1.3, but refer to the "Selected Experiments" manual for making connections.



Motor, Sensor and Disc Mounting

Figure 7-2

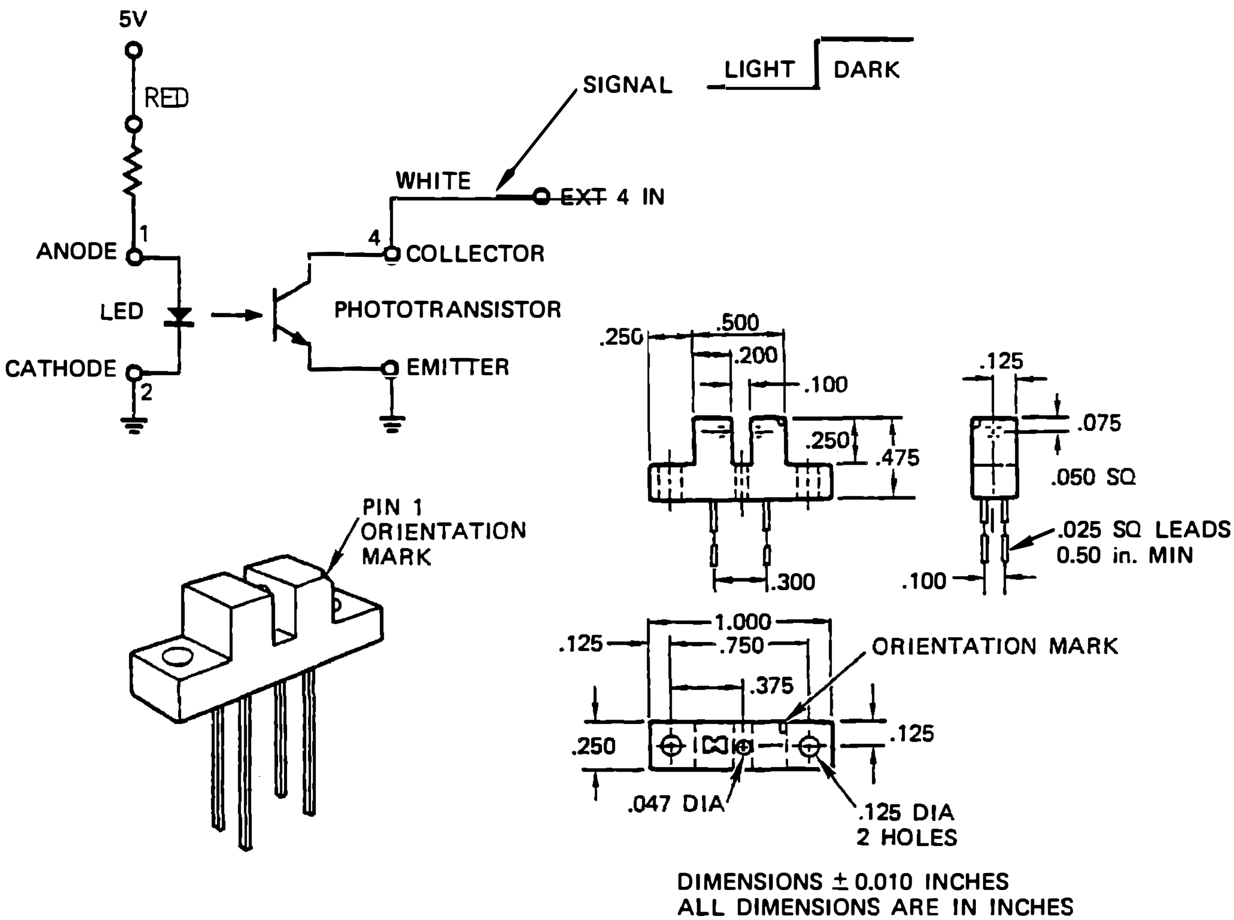
7.1.1 Motor, Sensor and Disc Mounting

The motor and slot sensor can be mounted on a block of styrofoam or balsa wood (Figure 7-2). Drill a 3/8 diameter hole through the block for the slot sensor leads. Cut a slot wide enough to grip the sensor (.25 inch) and deep enough to make the top of the sensor flush with the top of the block. This need not be at all precise.

Now cut a narrow slot across the width of the block to accept and guide the optical disc through the slot sensor opening.

Mount the optical disc on the motor in either of two ways. You can cut a small X at the center of the disc and force the motor shaft through that hole. The springy mylar will grip the shaft. If you find that the disc slips on the shaft, a small piece of tubing can be placed on the shaft on each side of the disc and squeezed together to grip the disc.

If you have cut the slot to fit the sensor closely it will hold it with no other mounting. If it is too loose, build up the projecting ends of the sensor with Scotch tape. The motor can be held in place on top of the block with tape or rubber bands.



CABLE			
PIN FUNCTION		COLOR	CONNECTION
1	LED ANODE	RED	+5 VOLTS
2	LED CATHODE	BLACK	GROUND
3	EMITTER	ORANGE	GROUND
4	COLLECTOR	WHITE	EXT 4 IN

Optical Slot Sensor

Figure 7-3a

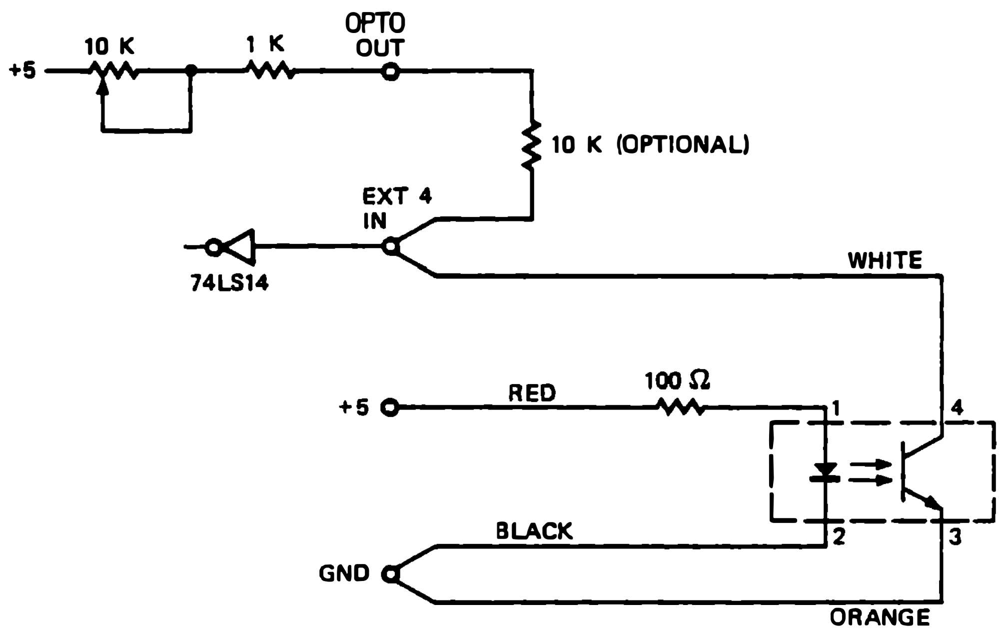

OPTO OUT
+5
10 K
1 K
10 K (OPTIONAL)
EXT 4 IN
74LS14
WHITE
+5
RED
100 Ω
1
4
BLACK
2
3
GND
ORANGE

Figure 7-3b

7.1.2 Slot Sensor Connection and Adjustment

The light emitting diode and phototransistor of the slot sensor are shown in Figure 7-3a. The pin numbers and color codes of the cable are shown. Note that a 100 ohm resistor is built into the red lead of the cable, so no external resistor is needed. Check that the cable is wired correctly, and plug the cable ends into the tie blocks as indicated in Figure 7-3b. Be very careful about these connections, because the slot sensor can be damaged or destroyed by reverse voltages.

Connect a 10K ohm resistor between EXT 4 and OPTO OUT. Together with the OPTO SENSE pot this provides a pullup resistance from EXT 4 to ground.

Now when a clear segment of the disc lies in the light path of the slot sensor, infrared light from the LED falls on the phototransistor and pulls the output signal close to ground. When a dark segment interrupts the light the phototransistor turns off and the voltage is pulled up close to 5 volts. Observe this with the voltmeter.

The 74LS14 Schmitt trigger circuit at the EXT 4 input requires that the signal switch below 0.6 volts to guarantee correct operation. The slot sensor may not have enough gain to achieve this with the pullup resistance. If it does not, remove the connection to OPTO OUT. Now the phototransistor will pull the input down when a clear segment is observed. When a dark segment is present the internal pullup resistance of the Schmitt trigger will pull the input voltage up to about 1.2 volts and the dark segment will be recognized. The circuit will operate correctly without the external pullup resistor, but will be more sensitive to noise pickup.

A test program is given in Figure 7-4. Connect EXT 4 to ANALOG IN in addition to the pullup resistor, and remove the voltmeter. The test program displays the state of the EXT 4 input in LED number 6, and uses the automatic A/D input to measure and display the voltage. With this program you can observe the switching and the actual voltage at the input as you rotate the optical disc and adjust the OPTO SENSE pot. Be sure that the voltage goes below 0.60 (3C hex) when a clear segment is observed. If it does not, remove the connection to OPTO OUT or substitute a higher valued resistor.

When a dark segment appears in the slot, the voltage must switch above 2.0 volts if an external pullup resistor is used. Without any pullup the Schmitt trigger should clamp the voltage at about 1.2 volts.

Be sure that the slot sensor operates correctly before going on with the program development. Unreliable operation of the detector will result in a useless control system.

TEST FOR SLOT SENSOR

ADDR		CODE		Instruction	Operand	Comments
8 20	0	3	E	MVI	A, 82	Program Ports
	1	8	2			Port 1B In
	2	D	3	OUT	CNT1	
	3	0	7			
	4	3	E	MVI	A, 92	
	5	9	2			
	6	D	3	OUT	CNT2	
	7	0	F			
	8	3	E	MVI	A, 94	Program and load
	9	9	4			Timer 2
	A	D	3	OUT	TIMCT	for A/D input
	B	1	7			
	C	3	E	MVI	A, 20	
	D	2	0			
	E	D	3	OUT	TIM2	
	F	1	6			
8 21	0	3	E	MVI	A, 03	Set Port 1C1 high
	1	0	3			for auto A/D
	2	D	3	OUT	PORT1C	Set Port 1C0 high
	3	0	6			to turn power
821	4	3	E	MVI	A, 08	transistor off
	5	0	8			
	6	D	3	OUT	PORT2C	Enable A/D interrupt
	7	0	E			without reset
	8	D	B	IN	PORT2B	Read interrupt
	9	0	D			status byte
	A	E	6	ANI	40	and display
	B	4	0			EXT 4 Input
	C	D	3	OUT	PORT1A	
	D	0	4			
	E	C	3	JMP	8214	Loop
	F	1	4			
8	0	8	2			
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					Figure 7-4a

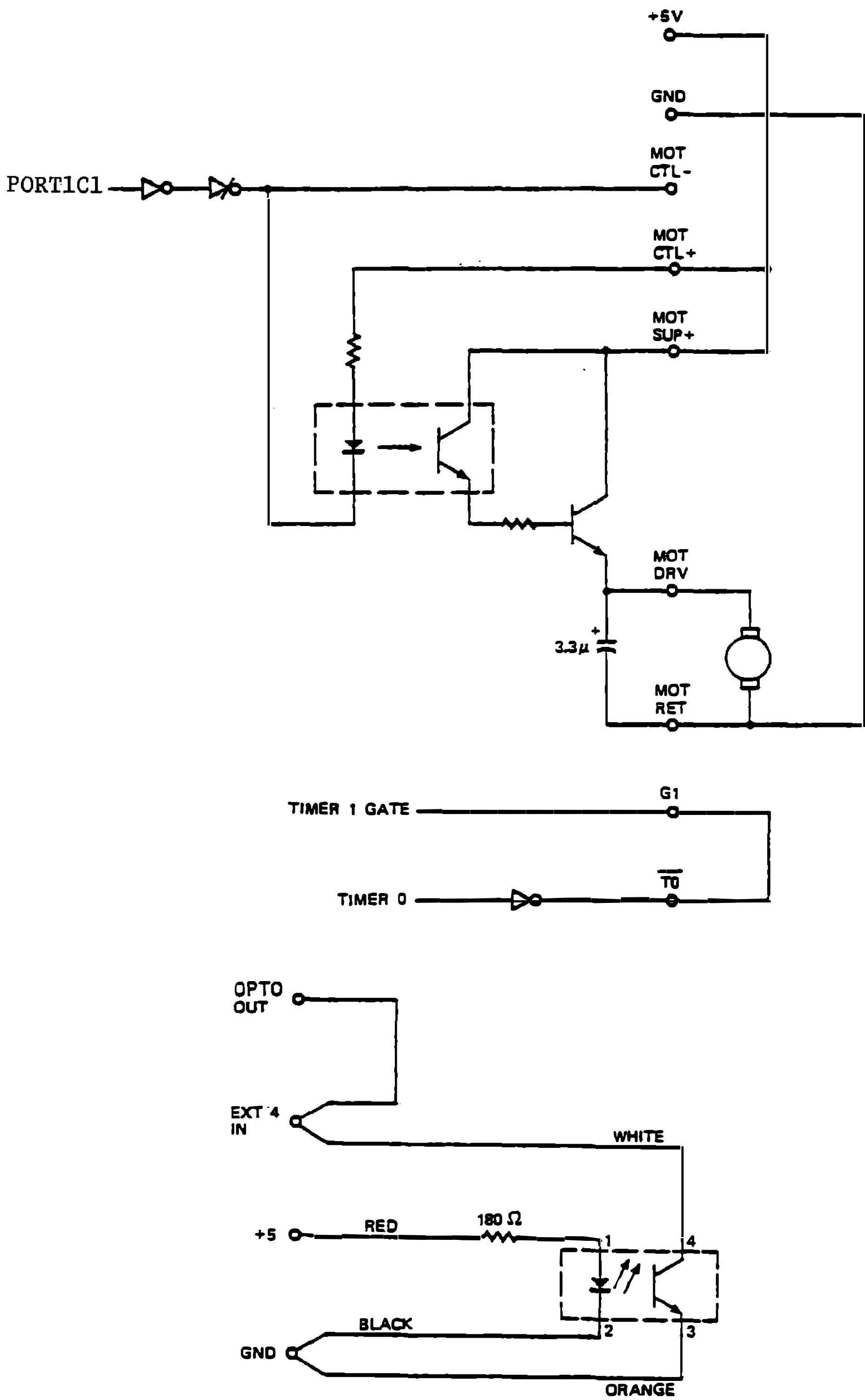
CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

TEST FOR SLOT SENSOR - A/D INTERRUPT

ADDR		CODE											
8230	0	F	5	P	U	S	H	P	S	W			
	1	D	B	I	N			P	O	R	T	1	B (L) ← voltage
	2	0	5										
	3	6	F	M	O	V		L	,	A			
	4	3	E	M	V	I		A	,	0	6		Reset
	5	0	6										
	6	D	3	O	U	T		C	N	T	2		
	7	0	F										
	8	7	D	M	O	V		A	,	L			
	9	C	D	C	A	L	L		D	B	Y	T	E
	A	9	5										
	B	0	2										
	C	F	1	P	O	P		P	S	W			
	D	F	B	E	I								
	E	C	9	R	E	T							
	F												
8	0												
	1												
	2												
	3												
	4												
	5												
	6												
	7												
	8												
	9												
	A												
	B												
	C												
	D												
	E												
	F												
8	0												
	1												
	2												
	3												
	4												
	5												
	6												
	7												
	8												Figure 7-4b



Motor Connections

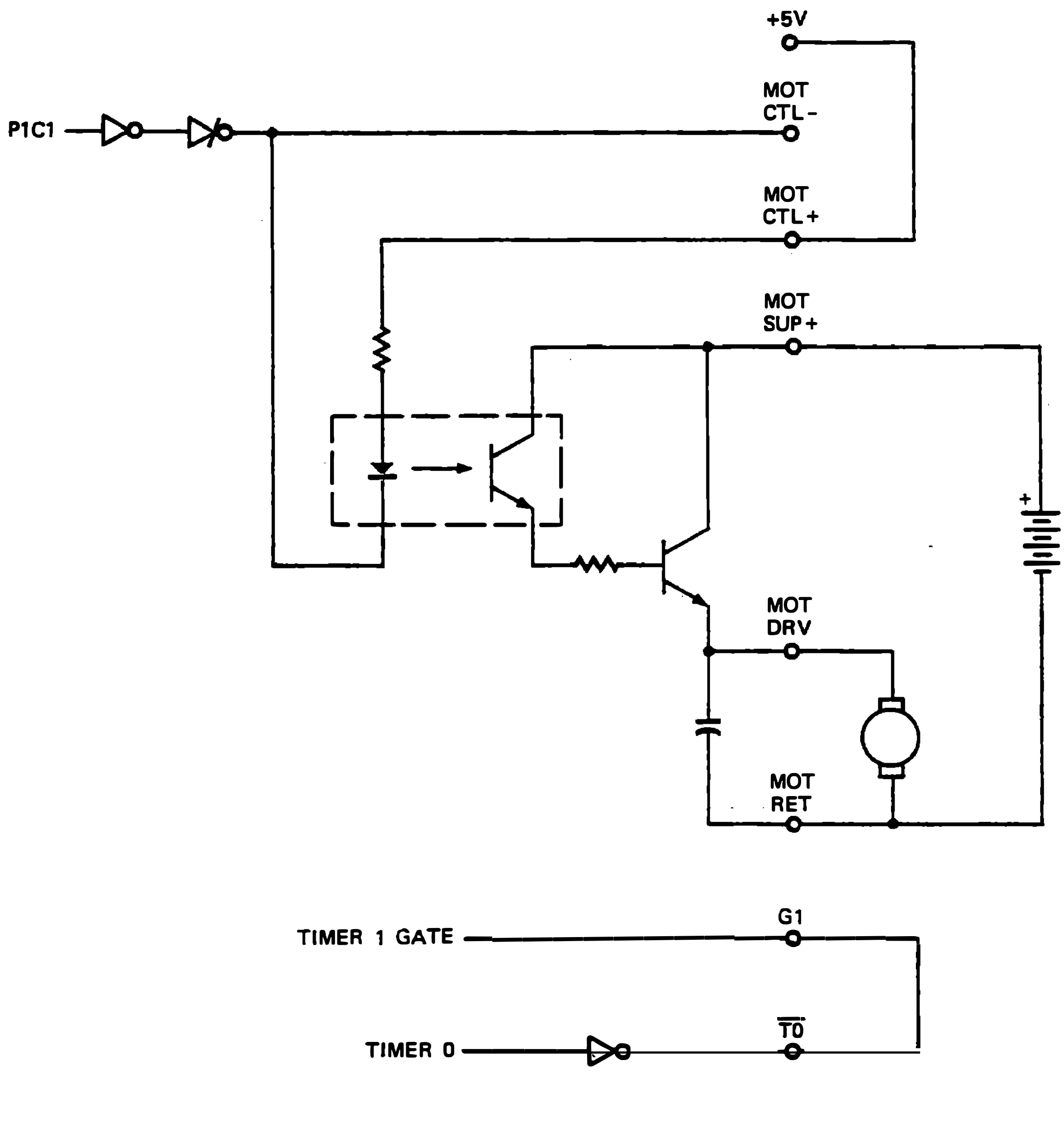
Figure 7-5

7.1.3 Motor Connection

The motor will be driven from the five volt supply using the power transistor as a switch for pulse width modulation control. Figure 7-5 shows the circuit connections for the motor and the slot sensor.

The power transistor and the optical coupler that drives it are isolated from the system power supplies to allow use of an external supply. For this experiment you can use the five volt system supply, although in general it is very poor design practice to place a noise generating load such as a motor on the computer's regulated supply. Figure 7-6 shows how the connections would be made with an external power source such as a lantern battery. Note that here there is no electrical connection between the motor and the computer.

Output PORT1C1 drives an inverter and an open collector inverter to control the optical coupler. When P1C1 is set low the open collector inverter draws current through the LED of the optical coupler. Infrared light from the LED turns on the phototransistor, which in turn provides base drive to the power transistor. This allows current to flow in the motor circuit either from the system power supply (as in Figure 7-5) or from the external supply (Figure 7-6).

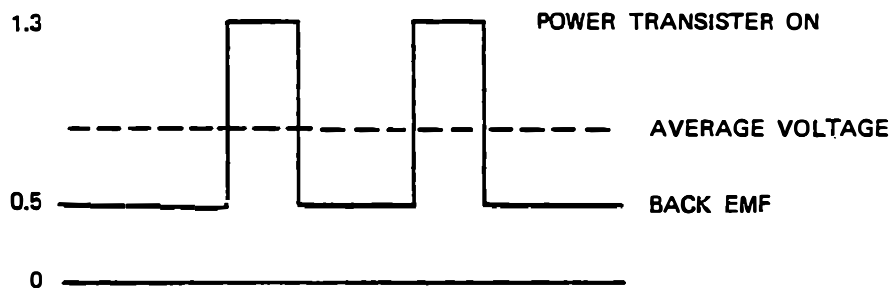


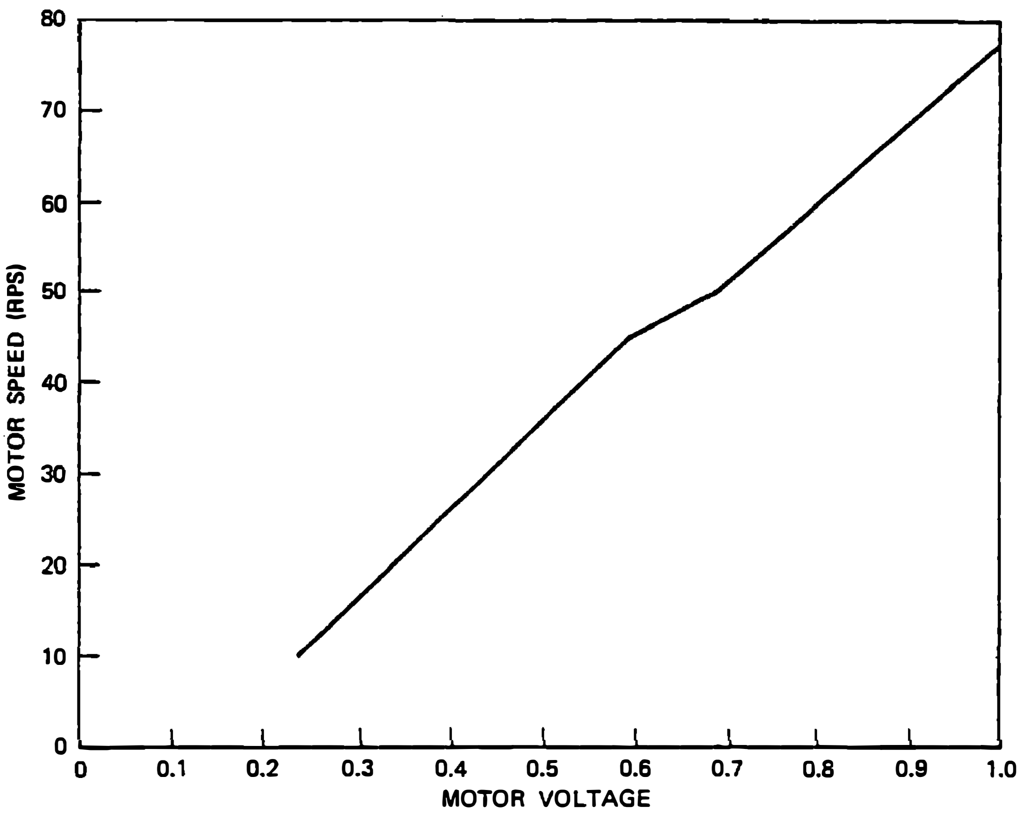
Motor Connections with External Power

Figure 7-6

Port 1C1 is set to input mode by system reset. In this condition it appears as a high input to the first inverter, so the open collector output is in the high impedance state and the optical coupler is switched off, so the motor does not run. When Port 1C is programmed for output during initialization its outputs become low, and power is applied to the motor. In general the initialization procedure should set Port 1C1 high fairly soon after the port has been programmed, so that the motor will not be turned on until intended.

Figures 7-5 and 7-6 also indicate that a connection is to be made from Timer 0 output to Timer 1 gate, for PWM control.





Motor Speed vs. Voltage

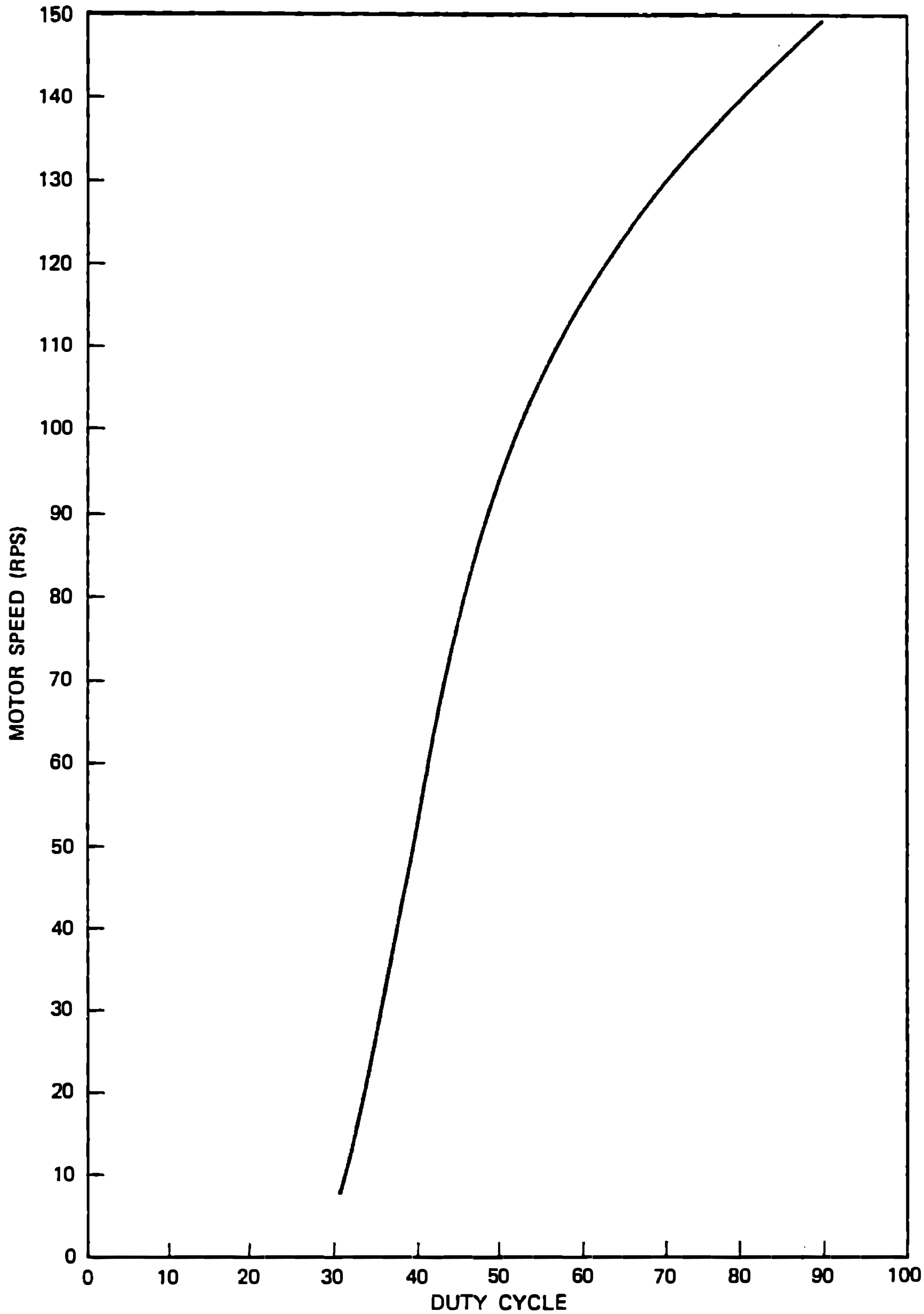
(Measured with closed loop control)

Figure 7-7

7.1.4 Motor Characteristics

The motor is a three pole commutated dc motor. Its speed with constant load is approximately proportional to the voltage across the motor, as indicated in Figure 7-7. The anomaly in Figure 7-7 in the vicinity of 0.6 to 0.7 volts is related to effects of synchronism between the driving pulses and the motor commutation. The data for Figure 7-7 was taken with the closed loop control program that is developed in this section. Other control schemes would show a linear relationship.

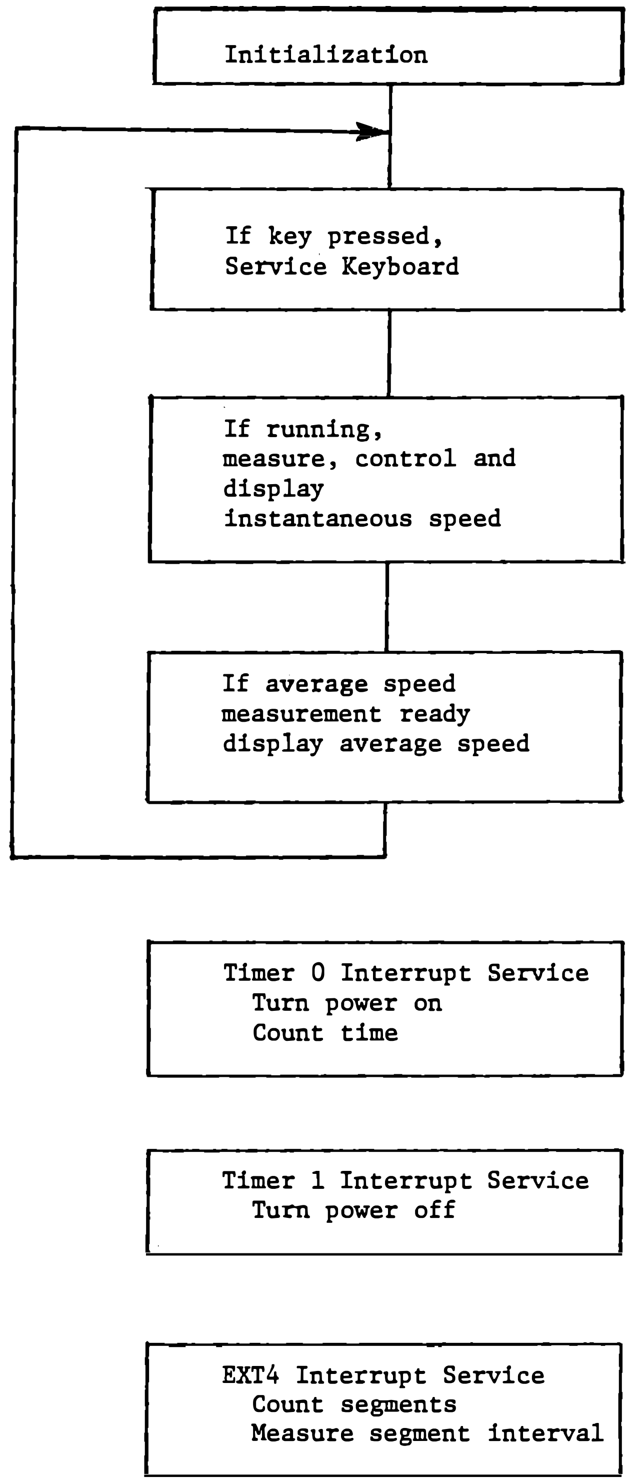
The average voltage measured at the motor is not linear with pulse width (or duty cycle) as the capacitor voltage was in Chapter 6. The motor itself generates a voltage, which depends on its speed. This is called "Back EMF". (EMF stands for ElectroMotive Force, which means voltage). This voltage is present whenever the motor is running, even though the power transistor is turned off part of the time. The voltage observed at the motor is shown at the top of Figure 7-7.



Motor Speed vs. Duty Cycle Open Loop

Figure 7-8

In Figure 7-8 the motor speed is shown as a function of pulse duty cycle. This was measured in an open loop control system. A linear relationship exists from 30% to 50% only. Below 30% duty cycle the motor will run only sporadically, while above 50% the slope decreases. No external load was placed on the motor for these tests, but the motor bearing friction represents a substantial load at the higher speeds. With a load on the motor the speed versus duty cycle would be linear over a larger range. Because this toy motor has plastic bearings different motors will behave very differently, and a single motor will change its behavior from time to time. Therefore you cannot expect to duplicate these results with any precision. The relationship shown here, with a small change in pulse width giving a large change in motor speed, will be typical.



Motor Control Program Structure

Figure 7-9

7.2 CONTROL SYSTEM DEVELOPMENT

We will develop a closed loop control system following the general design structure and development approach that were used in the preceding chapter for voltage control. In this program keyboard commands permit setting a pulse duty cycle for open loop control or setting a desired speed for closed loop control. There is provision for setting gains in the proportional plus integral control equation. The motor can be started by RUN and stopped by STEP.

As in the voltage control system the program comprises initialization, interrupt service, and a main loop which calls various subroutines as needed (see next page). A keyboard service module is called when the main loop detects a key being pressed. If the motor is running a speed control subroutine is called once during each pass through the main loop, and the instantaneous speed is displayed. Each time a new average speed measurement is completed the main loop calls DWORD to display the average speed.

Pulse width modulation is used for power control. Timer 0 runs continuously in mode 2 to define the pulse frequency, or total period. Power to the motor is turned on by Timer 0 interrupt. Timer 1 operates in mode 5 and is triggered by the output of Timer 0. The interval loaded to Timer 1 sets the pulse width (on-time) and its interrupt turns power off. Both instantaneous and average speed are measured and displayed. The control loop acts on the instantaneous speed measurement. The only new programming problem is the calculation of instantaneous speed.

Program Memory Assignments

8200 - 27	Initialization
8228 - 3F	Interrupt Manager
8240 - 5F	Timer Interrupt Service
8260 - 7F	EXT4 Interrupt Service
8280 - AF	Main Loop
82B0 - BF	Not used
82C0 - DF	Subroutine SPEED
82E0 - FF	Subroutine WIDTH
8100 - IF	KYTIM - Entry and Dispatch
8120 - 3F	Key Processing Modules
8160 - 7F	Subroutine LDT1
8180 - 5F	Subroutine DECBI
8190 - 9F	Subroutines SMULT, SCUML
81A0 - BF	Subroutine DIVID

Data Memory Assignments

83A0	Binary Time Count
83A1	Motor Control Byte
83A2	Binary Segment Count
83A3,A4	Decimal Segment Count
83A5,A6	Average Speed
83A7,A8	Timer 2 Data
83A9	Desired Speed
83AA	Integral Gain
83AB	Proportional Gain
83AC,AD	Error Integral

7.2.1 Speed Measurement

Speed is measured by observing the EXT4 interrupts generated by the optical disc. Each time a clear segment of the disc appears between the LED and the phototransistor of the slot sensor, the phototransistor is turned on, its output signal goes low, and an EXT4 interrupt occurs. In Chapter 5 we measured pulse interval time (Section 5.1) and frequency (Section 5.2). Both techniques are used in this program.

Average speed is measured as a frequency, by counting interrupts over a fixed period of time. Since the optical disc has 16 clear segments we will receive 16 EXT4 interrupts per revolution. If we were to count EXT4 interrupts during 1/16 of a second the count would represent revolutions per second. For average speed we would like better resolution, so we will count for 10/16 second, obtaining the average speed in tenths of a revolution per second. Thus a count of 0506 would represent 50.6 rps. For convenience the counting and display are in decimal.

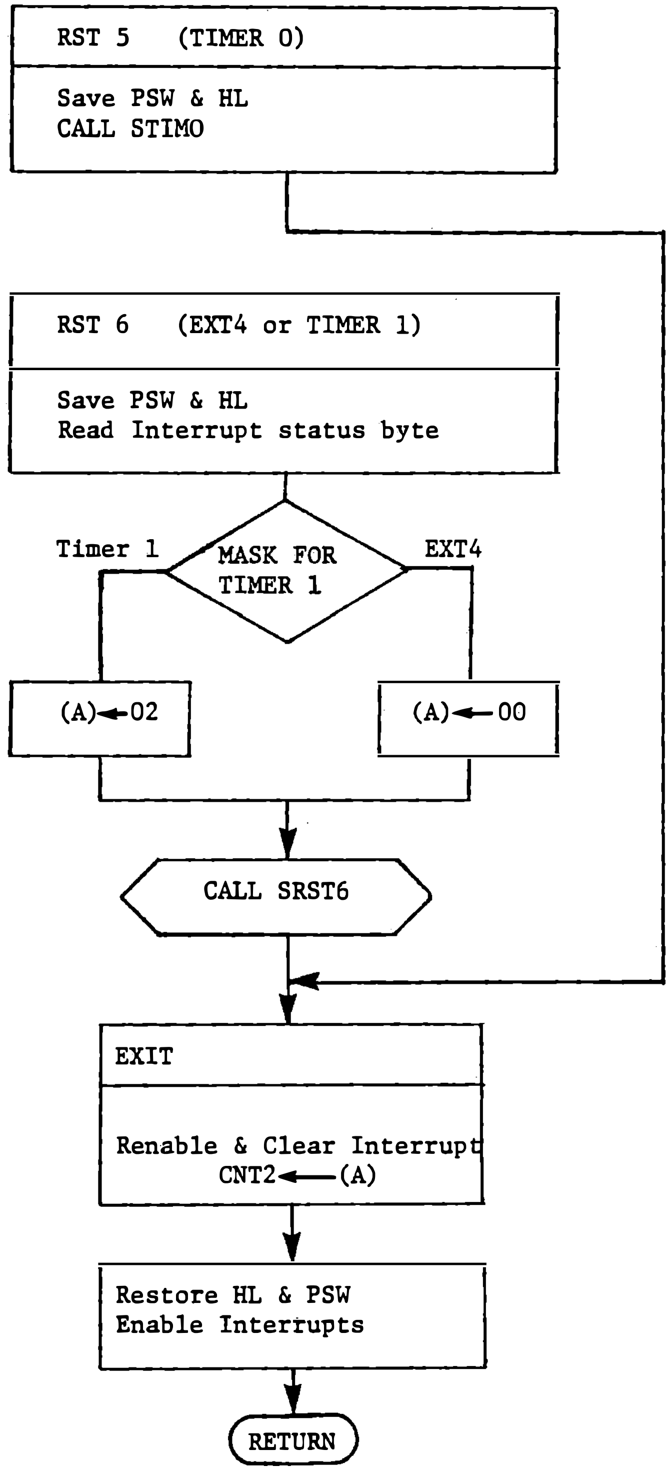
To control the speed well under variable load conditions we need more frequent measurements. Even the 1/16 second interval would be too infrequent for good speed control. Therefore the closed loop control is based on pulse interval measurement, giving "instantaneous speed" by division.

Speed = 16/(time per segment)

Time per segment = (clocks per segment)/2048000

Speed = 16 x 2048000/(clocks per segment)

The subject of binary division has not been treated previously in this course nor in Course 525. A subroutine, DIVID, is given in the program solution.



Motor Control Interrupt Manager

Figure 7-10

7.2.2 Interrupt Service

Three interrupts are used in the motor control system: Timer 0, Timer 1, and EXT4. Timer 0 is distinguished by its vector, RST5. Timer 1 and EXT4 both generate RST6 and must be distinguished by reading and masking the interrupt status byte.

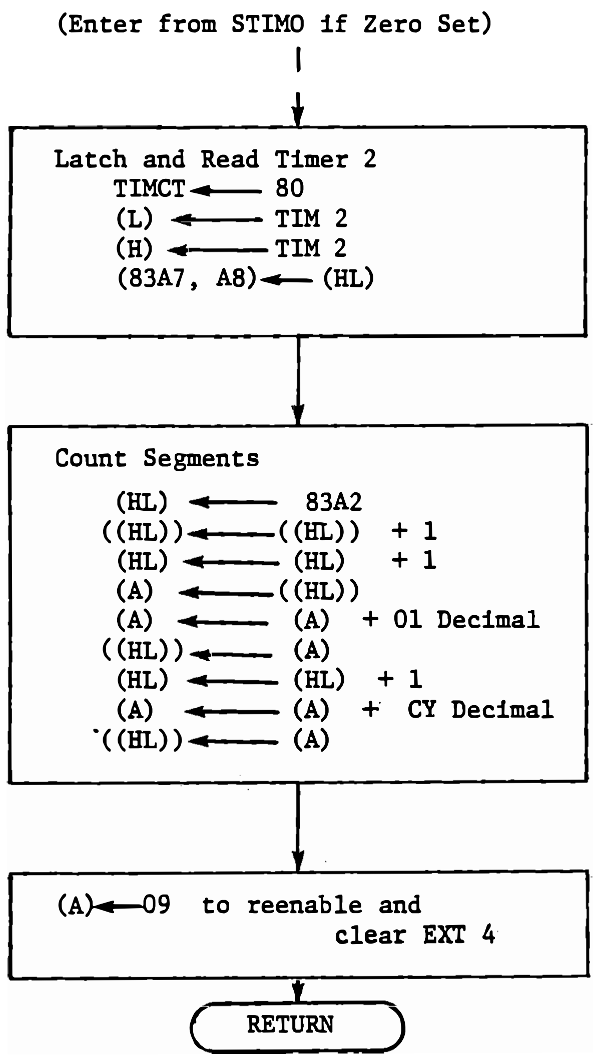
7.2.2.1 Interrupt Manager

Figure 7-10 shows the interrupt manager. Each entry saves appropriate registers (PSW and HL only). A service subroutine STIMO is called at RST5, and then a jump is made to the exit module. At RST6 the same registers are saved, the interrupt status byte is read and masked by:

```
IN      PORT2B
ANI     02
```

Then a service subroutine is called. At entry to this subroutine register A contains 02 if Timer 1 generated the interrupt. Otherwise (A) = 00 and the zero flag is set. The subroutine executes a jump if zero to service EXT4, or proceeds directly to service Timer 1.

Each service subroutine must return in register A the necessary control byte to clear and reenable its interrupt flip flop. The exit module writes this byte to CNT2, restores the environment, and returns to the interrupted instruction. Note that the service subroutines are restricted to using registers H, L, A, and the flags, or must preserve other registers.



EXT 4 Interrupt Service

Figure 7-11

7.2.2.2 EXT4 Service

EXT4 interrupt occurs each time a clear segment of the optical disc appears between the LED and phototransistor of the slot sensor. EXT4 service performs two functions in response. It latches and reads Timer 2, which is running continuously, and stores the data for use by the speed measurement subroutine. In that subroutine (called by the main program loop) we subtract the latest measurement from the preceding measurement to obtain the time difference. We could clear and restart Timer 2 at each EXT4 interrupt, thereby making each measurement complete in itself. We will see later that this would require more rather than fewer instructions in the speed measurement subroutine, as well as requiring additional instructions here.

Note an interesting point with regard to the mode selected for Timer 2. We usually think of mode 2 for a timer which is to run continuously. When no output signal or interrupt is needed we can use mode 0 instead, because the timer continues to count down after it reaches zero, although its output will remain high after the first time it reaches zero. Mode 0 is used in this program to demonstrate the point.

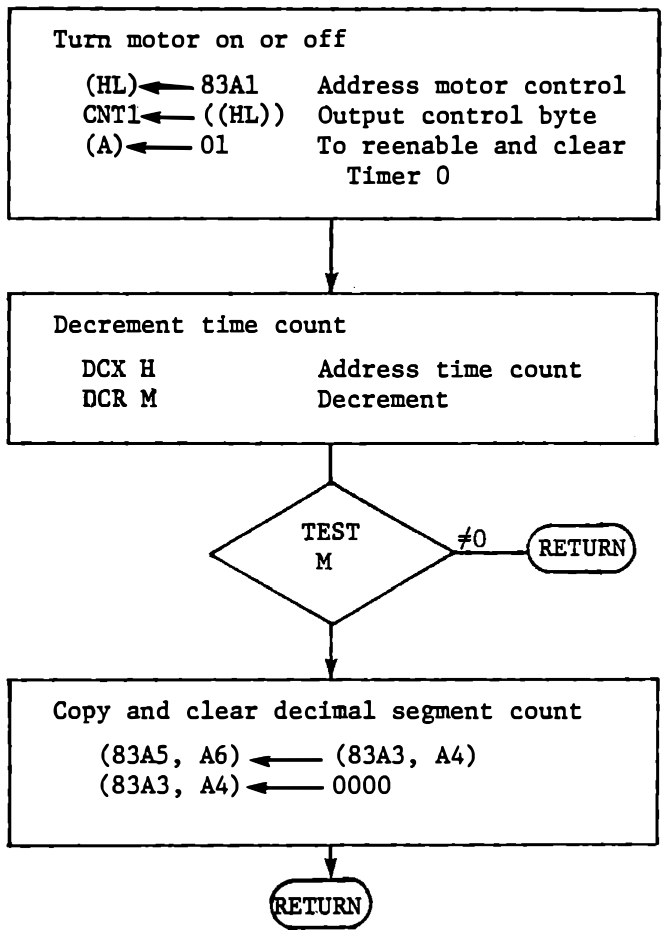
This page intentionally left blank.

7.2.2.3 Timer Service Subroutines

Pulse width modulation is controlled by Timer 0 and Timer 1 interrupts. Their service routines turn the power transistor on or off by writing a byte to Port 1C. If the byte is 00 it sets Port 1C1 low and turns the transistor on, while 02 sets Port 1C1 high and turns the transistor off. As in the previous program the Timer 0 interval is constant and defines the total period. If the motor is running, Timer 0 turns the power transistor on. The Timer 0 output also triggers Timer 1, whose interval sets the on-time. The Timer 1 interrupt turns the transistor off. Timer 1 has no other function.

Timer 1 Service

```
JZ      SEXT4           If zero set service EXT4
OUT     PORT1C          Output 02 to turn motor off
MVI     A,03            To reenable Timer 1
RET                     Exit
```



Timer 0 Service

Figure 7-12

Timer 0 has several functions. Since we often will want the motor to be stopped, the control byte to be written to Port 1C is stored in memory (at 83A1) by keyboard command. RUN stores 00 and STEP stores 02. This byte is loaded from memory and output at each Timer 0 interrupt. In addition Timer 0 decrements a time counter (at 83A0) and at zero it copies and clears the decimal segment count accumulated by EXT4 interrupts. The timing is arranged so that this count directly represents average speed. The completed count is stored at 83A5,A6 for display by the main loop.

There is a fortuitous time interval that is suitable for the PWM total period and also for measuring the 10/16 second interval for average speed. 256 counts in the binary timer counter equals 10/16 second if the PWM total period is set at:

$$10/(16 \times 256) = .00244140625 \text{ second}$$

Although this may appear to be an awkward time interval to obtain, in fact the system clock generates it very easily:

$$5000/2048000 = .00244140626 \text{ second}$$

Timer 0 is programmed in mode 2, decimal, high byte only, and loaded with 50. The interval, slightly less than 2.5 milliseconds, is quite suitable for pulse width modulation.

7.2.3 Initialization

Ports and Timers are to be programmed as follows:

8255 #1	A out B out C out
8255 #2	A in B in C out
Timer 0	High byte, mode 2, decimal
Timer 1	Both bytes, mode 5, binary
Timer 2	Both bytes, mode 0, binary

Timer 0 is to be loaded with 50 (high byte) to generate the 2.44 millisecond total period. Timer 2 must be loaded (in both bytes) to start it; the value does not matter since it will always count down from zero after it first reaches zero.

Recall that the motor is off when Port 1C1 is high, and running when that output is low. After system reset all ports are programmed for input, which gives an apparent high output and turns the motor off. As soon as Port 1C is programmed for output its output signals go low, the power transistor is turned on and power is applied to the motor. (You can demonstrate this by stepping through your initialization procedure). Since none of the control data have been entered we want to stop the motor during initialization; this can be done by a call to the CLR key processing module in KYTIM. The command keys are defined in Section 7.2.5.

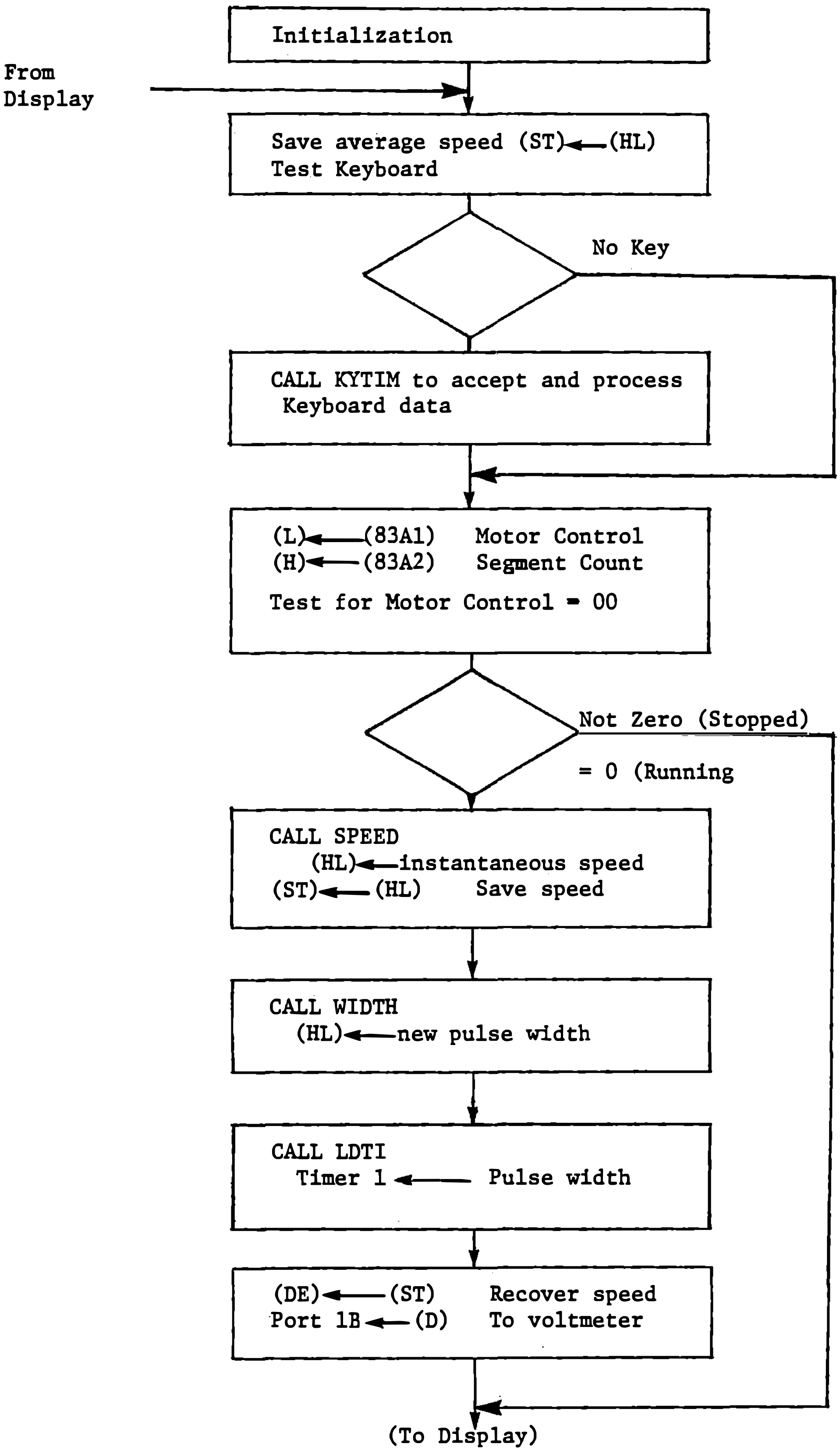
We have used RST5 and RST6 commands in the past to enable interrupts. This would work for RST5 in this program, but two calls to RST6 service would be required to enable both EXT4 and Timer 1. Even with two calls there is no certainty that Timer 1 would be enabled because it is only serviced if its interrupt is present. Therefore we enable the interrupts by writing 13 to Port 2C. This usually results in all three interrupts being serviced immediately, since writing to Port 2C does not clear the interrupt flip flops. During debugging it is desirable to replace this process with the RST instructions, which will permit stepping though the interrupt service routines.

	Final		Debug	
3E	MVI	A,13	EF	RST5
13			F7	RST6
D3	OUT	PORT 2C	F7	RST6
0E			00	NOP

While you step through the initialization the motor will run at full speed until the CLR function stops it. It is advisable to unplug the motor during this task.

You may have to force the service of Timer 1 by replacing the interrupt status byte after reading it. (Use REG, A, 02 after the status byte has been read by the IN PORT2B instruction).

When debugging of the initialization and interrupt service routines has been finished, replace the RST instructions with the load and output instructions. These are followed by a jump to the main loop.



Motor Control - Main Loop

Figure 7-13a

7.2.4 Main Program Loop

Figure 7-13 shows the main program loop. This is a more detailed version of Figure 7-9.

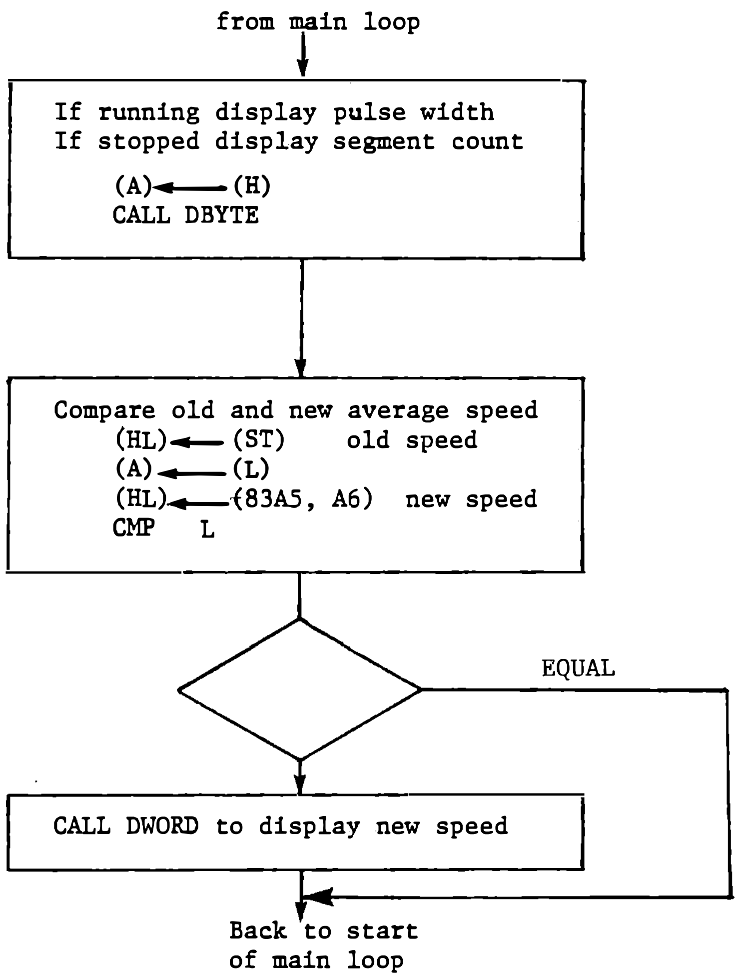
At the start of the loop register pair HL normally contains the average speed. (After initialization this value is meaningless). This value is saved, and the keyboard is tested by:

```
IN      PORTOA
INR     A
CNZ     KYTIM
```

Note that while Figure 7-13a indicates a conditional jump, a conditional call is actually used here.

As in the voltage control program the process operates in open loop mode while KYTIM waits for keyboard entry and processes the data entered.

The motor control byte stored for Timer 0 interrupt service is also used here to determine whether the motor is running. If it is stopped (control byte = 02) we skip all of the control functions and go to display the binary segment count. If the motor is running three subroutines are called. SPEED finds the instantaneous speed from a segment interval; WIDTH calculates a new pulse width; LDT1 loads Timer 1 with the new width.

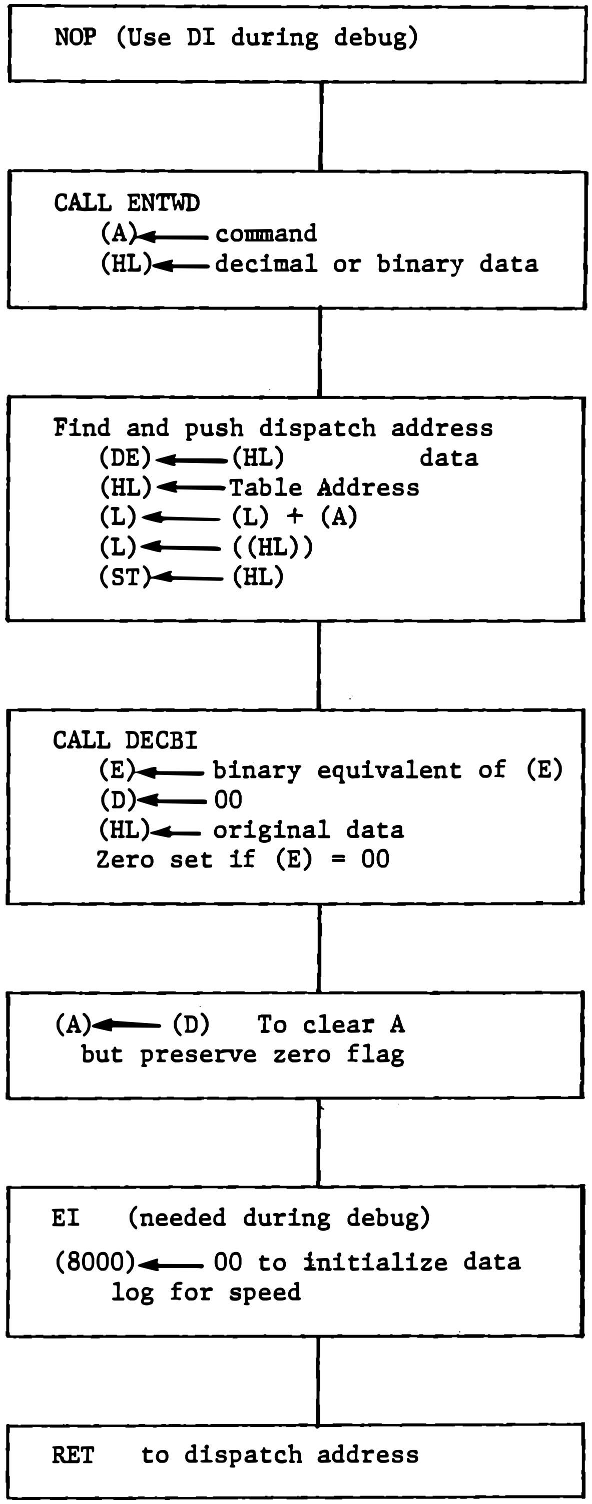


Motor Control - Display

Figure 7-13b

The high byte of the speed is output to the D/A converter. This allows observation of the speed with a voltmeter, which is interesting to watch when a load is placed on the motor. Either the high byte of the speed, or optionally the high byte of the pulse width, is displayed at the right, as shown in Figure 7-13b.

After displaying the instantaneous speed or the segment count the main loop may display the average speed. The average speed is stored by Timer 0 interrupt once every 5/8 second. Since the optical disc has 16 light segments, the decimal segment count during the 5/8 second directly represents speed in tenths of a revolution per second. Thus a display of 0506 means 50.6 rps. To avoid wasting time in the control loop the program tests for a change in the average speed before displaying it. To permit this test the old value is stored at the beginning of the main loop and recovered before the (possibly) new value is loaded. The result is that DWORD is called only once in about 250 passes through the main loop. (It is sufficient to compare the low bytes of the old and new speeds, since it is unlikely that successive measurements will be alike in the low byte).



KYTIM - Input and Dispatch

Figure 7-14

7.2.5 Keyboard Input Subroutine KYTIM

This version of KYTIM uses the same keyboard entry and dispatch techniques used in previous programs. A decimal to binary conversion is required, since some data are entered as decimal values but are needed as binary values. This is done by subroutine DECBI (Section 7.2.6) which is entered with keyboard data in (HL), and returns with the input data preserved and the binary equivalent of the low byte in (E) and also in (A). Register D is cleared. The zero flag is set if (E) = 00. After return register A is cleared by MOV A,D, so the zero flag is preserved and passed to the command processing module.

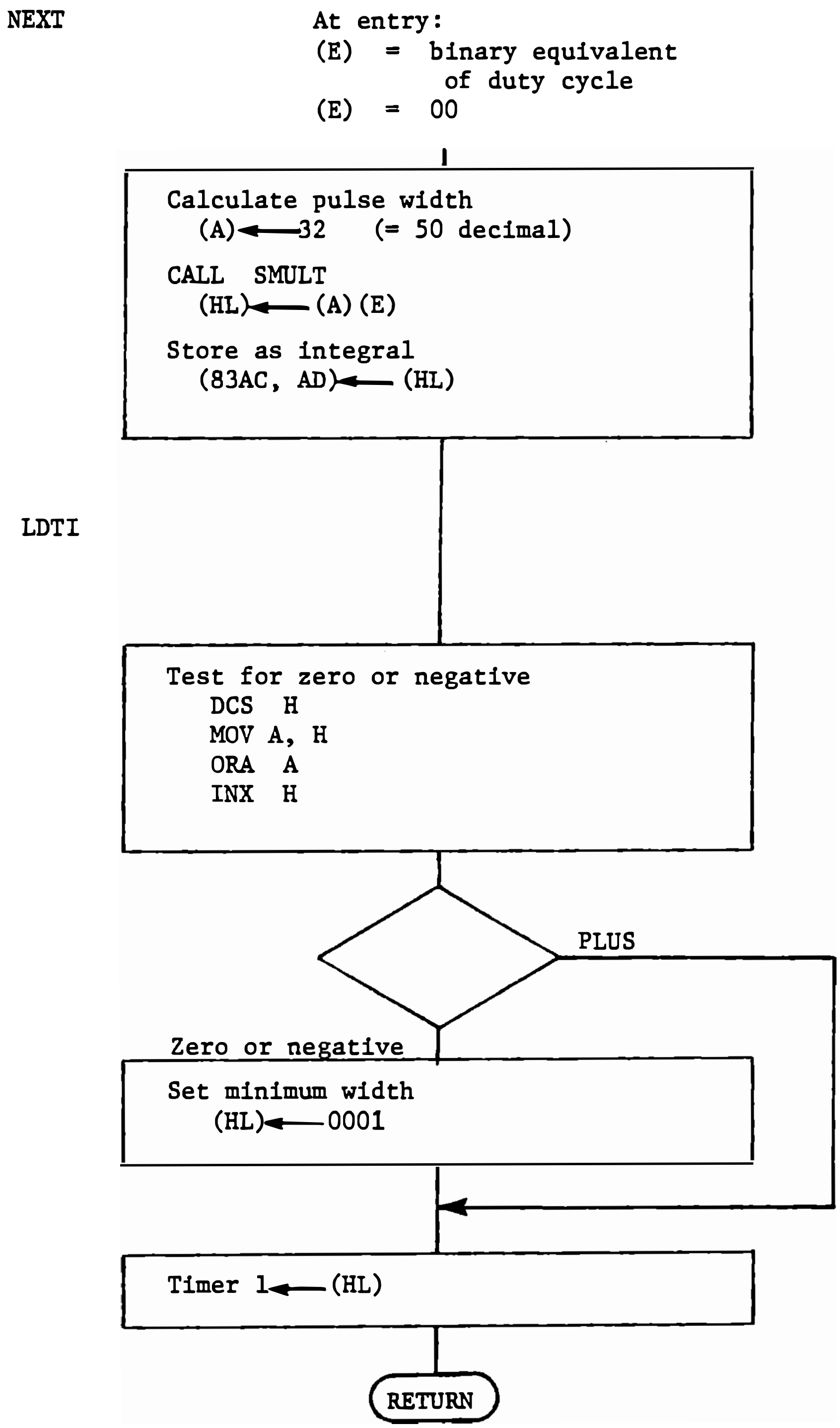
We will provide for logging the motor speed for subsequent review, as we logged the voltage in Chapter 6. Clear the content of memory location 8000 to initiate a new log at each keyboard entry.

This page intentionally left blank.

The following command key processing modules are needed:

CLR	Clear the binary segment count and stop the motor. This permits measuring the stopping distance.
STEP	Stop the motor (turn off the transistor and store 02 at 83A1)
RUN	Start the motor (store 00 at 83A1). If a desired speed is entered, store its binary equivalent at 83A9.
MEM	Store integral and proportional gain. These are entered in binary, and are to be stored at 83AA and 83AB. (83AA,AB) <--- (HL)
NEXT	Given a decimal duty cycle (converted to binary by DECBI) calculate pulse width. Store the results as

The total period for PWM has been set to 5000 (decimal) clocks. We can calculate a pulse width by multiplying the duty cycle (treated as an integer) by 50 (decimal). For instance, if a duty cycle of 40% is requested, 40 x 50 = 2000, which is 40% of 5000.



Load Timer 1 Modules

Figure 7-15

In fact the calculation will be done in binary, and the binary result is used. A multiplication subroutine SMULT (Section 7.2.7) returns (HL) = (A)*(E). Since DECBI has given (E) = binary equivalent of the input data, the processing for NEXT is:

```
MVI     A,32          Binary equivalent of 50
CALL    SMULT         (HL) <--- pulse width
SHLD    83AC          Store as integral
CALL    LDT1          Load Timer 1
```

As in the voltage control program the Timer 1 loading subroutine is also used in closed loop control. It tests for a zero or negative value in (HL) and replaces such a value with 0001 for minimum pulse width. Timer 1 is loaded with the contents of L and H.

7.2.6 Subroutine DECBI

A two digit decimal value can be converted to an equivalent binary value by:

$$\text{Binary value} = \text{Low digit} + 5/8\ (\text{High digit})$$

A convenient algorithm for this calculation is expressed as:

$$\begin{aligned}\text{Binary Value} &= \text{Decimal Value} \\ &+ 1/8\ \text{High digit} \\ &- 1/2\ \text{High digit}\end{aligned}$$

For convenience in the motor control program subroutine DECBI accepts and returns data as follows:

ENTER

(HL)	=	Keyboard data

RETURN

(HL)	=	Keyboard data preserved
(A) = (E)	=	Binary equivalent of low byte
(D)	=	00

Zero set if low byte is zero

CY clear

(BC) is preserved

A program solution is given in Figure 7-22d.

7.2.7 Subroutines SMULT, SCUML

SMULT multiplies two single byte values and returns the two byte product. SMULT clears the product at entry. SCUML is an alternate entry at which the product is not cleared, allowing cumulative multiplication.

ENTER

(A)	=	Multiplier
(E)	=	Multiplicand
(D)	=	00 if multiplicand is positive
(D)	=	FF if multiplicand is negative
(HL)	=	Previous product (SCUML only)

RETURN

(HL)	=	Product (A)*(E) for SMULT
(HL)	=	(HL) + (A)*(E) for SCUML
(DE)	destroyed	
(BC)	preserved	
(A)	= 00	

Zero set, Carry clear

The multiplication is carried out by repetitively shifting (A) right, and adding the multiplicand to the product if the bit shifted out is a one. After each shift of (A), and after the addition if it is performed, the multiplicand is shifted left. The subroutine returns when the content of A is zero. A program solution is given in Figure 7-22e.

7.2.8 Open Loop Operation

With the functions described so far you can operate the system in open loop mode. (Use an unconditional JMP around the control functions instead of JNZ. This permits debugging of the main loop, KYTIM, and interrupt service.) Write all of these program modules. Check the program flow through interrupt service, using RST5 and RST6 instructions. Check the program flow through KYTIM and see that the calculations and data storage are correct.

To run the motor enter a high duty cycle (60% or more) with NEXT and then press RUN. See that the motor can be speeded by higher duty cycles and slowed by lower duty cycles. Find the lowest duty cycle that will keep the motor running once it is started, and the lowest that will cause it to start. Record and plot speed versus duty cycle, and compare your result with Figure 7-8.

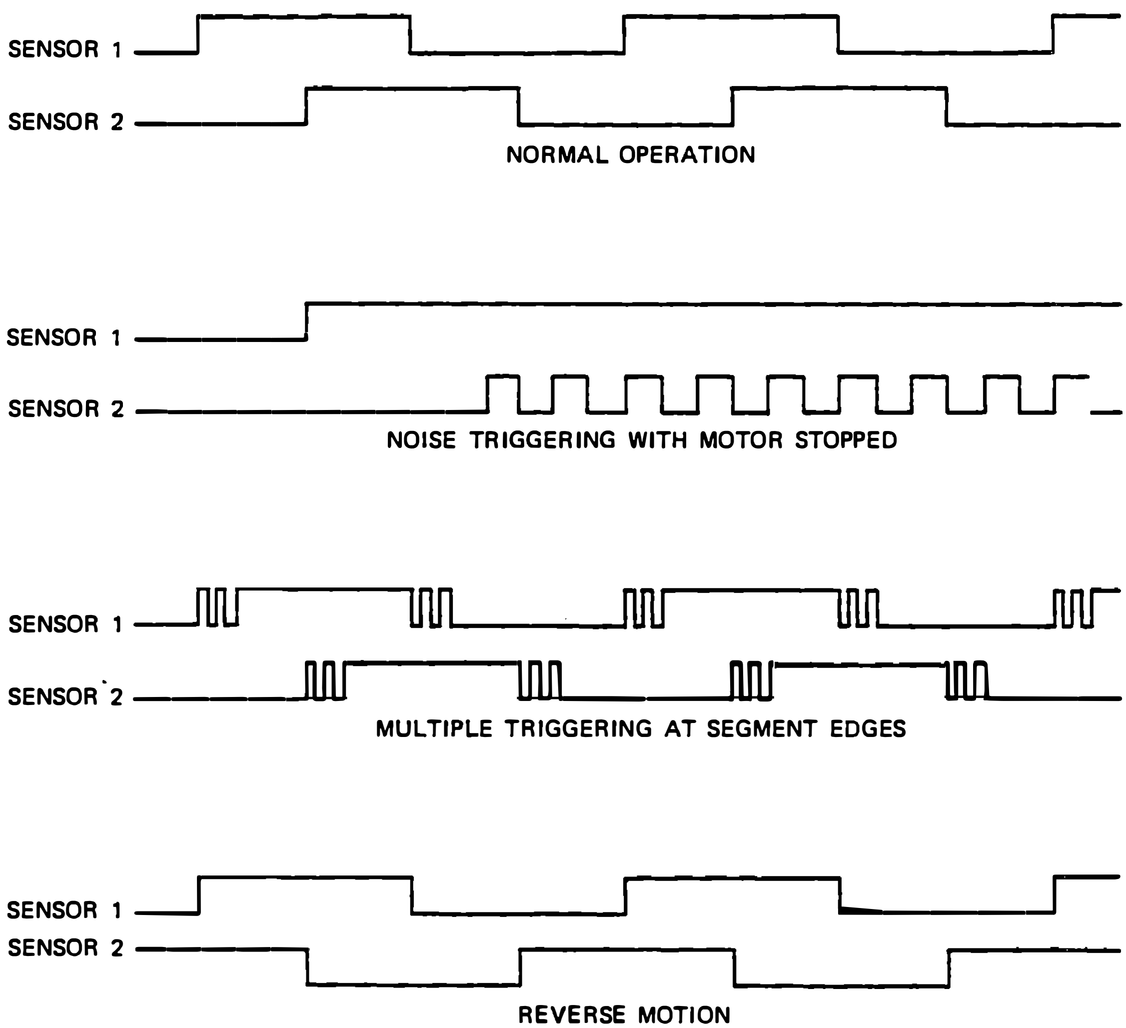
7.2.9 False Speed Indications

When the motor is stopped with an edge of a segment in the optical sensor light path, the phototransistor will be in an active state, neither fully on nor fully off. In this state the circuit is very susceptible to small signals, and noise pickup is likely to cause the EXT4 input to switch. The Timer 0 output, which is connected to Timer 1 gate in this experiment, is a source of such noise. If this source does cause switching, every Timer 0 cycle will result in an EXT4 interrupt. Then the binary and decimal segment counts will constantly be incremented. After 256 counts Timer 0 service will copy and clear the decimal count, so an average speed of 2.56

revolutions per second will be displayed. It is fairly difficult to find the exact sensor position necessary to obtain the continuous counting. If you turn the shaft slowly by hand it is easy to observe multiple counts for each segment, demonstrating that the false switching occurs.

Another false speed indication can occur when power is applied to the motor but the pulse width is insufficient to start it. At each pulse the motor shaft will turn slightly, but will be pulled back by the motor magnets when the pulse ends. If this occurs with a segment edge in the optical sensor light path the phototransistor will switch in time with the motor pulses.

There is no good solution to these problems with a single optical sensor. In any application where such false indications are intolerable it is necessary to use two sensors to observe the disc. They must be so located that the two sensors cannot both see segment edges at the same time. Now the program can test for a sequence of switching in the two sensors to ensure that only valid segment transitions are observed. Figure 7-16 shows the sequences under various conditions. A program could be designed so that an interrupt from sensor 1 disables itself and enables the sensor 2 interrupt, and the sensor 2 interrupt disables itself and enables sensor 1. Now interrupts can occur only when both sensors switch in sequence. Noise and multiple triggering are thereby excluded.



Motion Detection with Dual Sensors

Figure 7-16

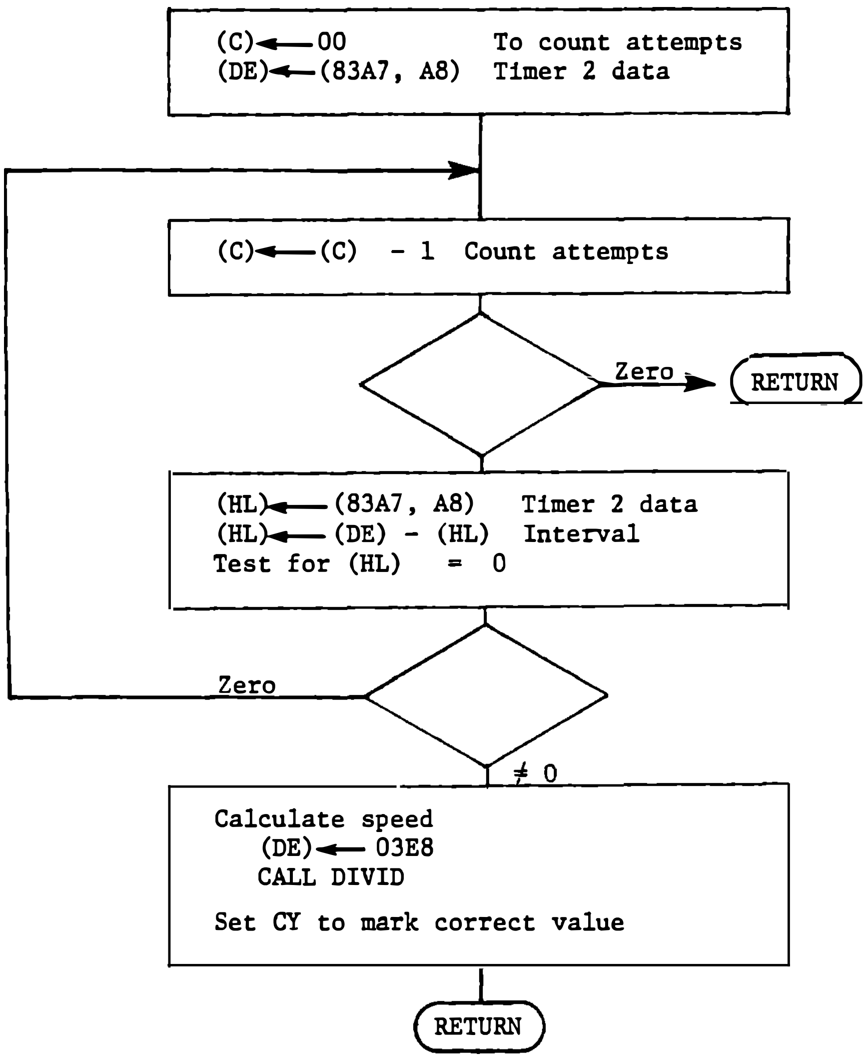
Dual sensors can also be used to detect direction of motion. The difference between the top and bottom diagrams in Figure 7-16 is readily analyzed to determine which way a shaft is turning. Since we have only a single sensor available, we cannot determine direction nor eliminate the false speed indication. This is a problem only when the motor is stopped or running very slowly. At speeds above 5 to 10 revolutions per second the single sensor is accurate. Although some schemes are available to reduce the probability of false speed indications with a single sensor, we will not attempt their implementation.

7.3 CLOSED LOOP MOTOR CONTROL

To close the loop we will calculate the instantaneous speed from the measurement of segment time, and apply the proportional plus integral control equation:

$$F = G_p E + \int G_i E$$

Five subroutines are used. SPEED obtains the interval time and calls DIVID to calculate the instantaneous speed. WIDTH subtracts the instantaneous speed from the desired speed to give the error signal, calls a cumulative multiplication subroutine SCUML to calculate a new error integral, and calls SCUML again to calculate a new pulse width. Finally LDT1 loads Timer 1 with the pulse width.



Subroutine SPEED

Figure 7-17

7.3.1 Subroutine SPEED

EXT4 service records the content of Timer 2 when the optical disc generates an interrupt. SPEED (shown in Figure 7-17) repeatedly loads this value (from 83A7,A8) and subtracts it from the previous value. If the result is zero no EXT4 interrupt has occurred, so the reading and subtraction are repeated. After 256 attempts it is assumed that the motor is not moving and SPEED returns with (HL) = 0000.

If successive values of Timer 2 data are different the subtraction of the later value from the earlier gives the time interval, since the timer counts down.

We will obtain the instantaneous speed by division. The speed in revolutions per second is given by:

(16 segments/rev.X2048000 clocks/sec.)/clocks per segment

The division subroutine DIVID (Section 7.3.3) is designed for use with left justified floating point numbers, although it does not handle the exponents. It returns a 16 bit result in (HL) with the most significant bit representing the integer part and 15 bits representing a fraction. It can handle numbers that are not left justified provided that the dividend is not greater than twice the divisor.

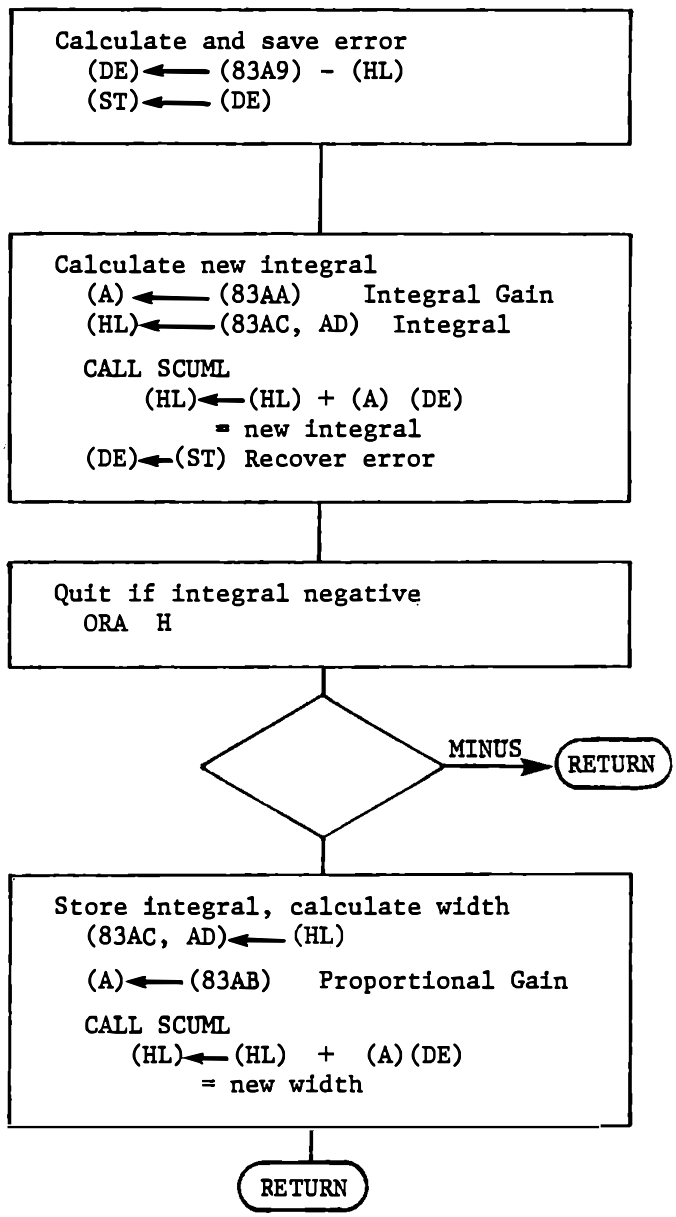
This page intentionally left blank.

It turns out that if we enter DIVID with a dividend of 03E8 (in DE) and the segment interval (in HL) as divisor, it returns a quotient (in HL) representing speed as XX.XX. That is, H contains the integer part and L contains the fractional part. The table below shows the relationship between speed and clocks per second. For all speeds that this motor can achieve the number of clocks per segment (the divisor) is more than half of 03E8, so DIVID will return a correct result.

Carry is set before return from SPEED to indicate that a valid measurement has been made. This is in fact ignored in the present main program, but could be used to invoke full scale control.

Speed and Clocks per Segment

Motor Speed rps	Segments per Second	Seconds per Segment	System Clocks per Segment Decimal	Hex
2	32	.031250	64000	FA00
5	80	.012500	25600	6400
10	160	.006250	12800	3200
20	320	.003125	64000	1900
50	800	.001250	2560	0A00
100	1600	.000625	1280	0500
150	2400	.000417	853	0355



Subroutine WIDTH

Figure 7-18

7.3.2 Subroutine WIDTH

At entry to WIDTH the instantaneous speed is in (HL) as XX.XX . The calculations are limited to single byte values with sign, so we calculate error from the high byte, rounding from the low byte of speed.

```
MOV    A,L     Set CY if low byte
RAL            Greater than 1/2
LDA    83A9    Desired speed
SBB    H       Subtract speed
MOV    E,A     (E) <--- error
SBB    A       (D) <--- FF if negative
MOV    D,A     (D) <--- 00 if positive
```

The multiplication subroutine SCUML demands that register D contains 00 if (E) is positive, FF if negative. The error is saved in the stack because it is needed for both the integral and proportional calculations, and SCUML destroys the contents of DE. SCUML returns (HL) = (HL) + (A)*(E). To calculate a new integral we load the integral gain to A and the old integral to HL and call SCUML. At return HL contains the new integral and A contains zero. The error is recovered by POP D, and the integral is tested by ORA H, which sets the minus flag if the integral is negative. RM (Return if Minus) after the test avoids storing a negative integral.

This page intentionally left blank.

Provided that the integral is positive, a new pulse width is calculated by loading A with the proportional gain and calling SCUML again.

At this call we have:

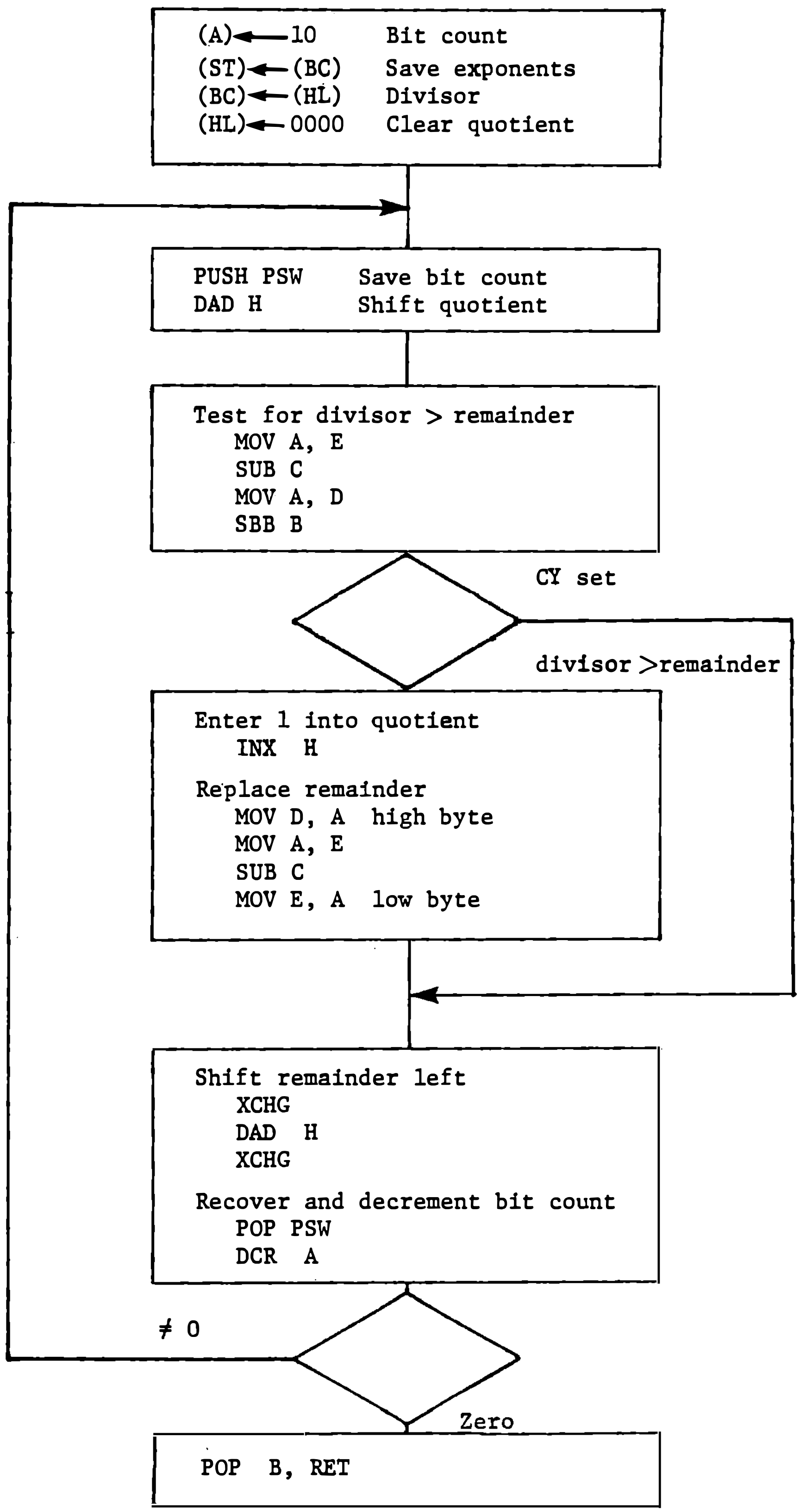
(A)	=	Proportional gain
(E)	=	Error
(HL)	=	Integral

SCUML returns (HL) = (HL) + (A)*(E) which is the new pulse width:

$$F = G_pE + \int G_iE$$

The main loop will call LDT1 to load Timer 1 with the new pulse width.

You may want to elaborate this program to permit a negative integral and test for arithmetic overflow. If a positive error changes the integral from positive to negative, the motor is running too slowly (or is stalled), so return a large pulse width, such as 1200H (92% duty cycle). If a negative error changes the integral from negative to positive, return a minimum pulse width (0001) to slow the motor. The erroneous integral should not be stored. The same test and correction should be made after adding in the proportional term.



Subroutine DIVID

Figure 7-19

7.3.3 Subroutine DIVID

Binary division is performed by repeatedly comparing the divisor to the dividend or remainder. If the divisor is less than the dividend or remainder, it is subtracted to form a new remainder and a one is entered into the quotient. Otherwise the old remainder is retained and a zero is entered into the quotient. Now both remainder and quotient are shifted left if more bits are to be processed, and the process is repeated.

Figure 7-19 shows subroutine DIVID. Registers B and C are saved in the stack; in a floating point division program which uses DIVID these registers hold the exponents. Here we do not really need to save them.

Register A is loaded with a bit count; the divisor is copied to (BC) and the quotient (HL) is cleared. The subroutine could be shortened here, because 16 left shifts will clear the old data from (HL). In some fixed point applications, however, a different bit count might be used.

Since all registers are used the bit count is saved in the stack during each loop. The quotient is shifted left at the beginning of the loop rather than at the end because it should not be shifted after the final bit has been processed. The left shift enters a zero into the quotient.

This page intentionally left blank.

A two byte subtraction sets carry if the divisor is greater than the remainder, in which case the old remainder and the zero bit shifted into the quotient are retained. If the subtraction does not generate carry the low bit of the quotient is made one by INX H, and the remainder is replaced. The high byte of the new remainder is available in A, but the low byte must be generated.

Now the remainder is shifted left and the bit count is recovered and decremented. When the bit count reaches zero the quotient is compete in (HL).

Because this subroutine was developed for floating point arithmetic it expects left justified values--the highest bit of the dividend and of the divisor should be one. This implies that the dividend is less than twice the divisor. The 16 bit quotient represents a single bit to the left of the binary point and 15 bits to the right. The algorithm is valid for numbers which are not left justified only if the dividend is less than twice the divisor. As we have seen, this relationship does hold in the motor control program.

		CY	ZERO	A	B	C	D	E	H	L
KYTIM -	Entry	x	x	x	x	x	x	x	x	x
	Return	x	x	x	00	Command	x	x	Keyboard Data except if NEXT	
SPEED -	Entry	x	x	x	x	x	x	x	x	x
	Return	See Notes	Set	00	Saved	00	x	x	Instantaneous Speed	
WIDTH -	Entry	x	x	x	x	x	x	x	Instant speed	
	Return	Clear	Set	00	Saved		x	x	Pulse Width	
LDTI -	Entry	x	x	x	x	x	x	x	Pulse Width	
	Return	Clear	x	=(H)	Saved		Saved		See Notes	
DECBI -	Entry	x	x	x	x	x	x	x	Keyboard Data	
	Return	Clear	See Notes	Binary of low byte	Saved		00	Binary of low byte	Keyboard Data Preserved	
SMULT -	Entry	x	x	M'plier	x	x	00 or FF	M'cand	x	x
	Return	Clear	Set	00	Saved		x	x	(A)*(E)	
							(See Notes)			
SCUML -	Entry	x	x	M'plier	x	x	00 or FF	M'cand	Previous Value	
	Return	Clear	Set	00	Saved		x	x	(HL) + (A)*(E)	
							(See Notes)			
DIVID -	Entry	x	x	x	x	x	Dividend		Divisor	
	Return	Saved	Set	00	Saved		x	x	Quotient	
							(See Notes)			

Subroutine Register Usage

Figure 7-20

7.3.4 Summary of Subroutines

The subroutines used in the motor control program are briefly summarized below. Figure 7-20 shows the register usage for each subroutine. In the figure the data required at entry and returned at exit are listed. An X for entry indicates that the register content does not matter; an X for return indicates that the register content is destroyed.

KYTIM accepts (via ENTWD) optional decimal or hexadecimal data and a command. Data entered are stored in assigned memory locations as described in Section 7.2.5. Valid commands are:

NEXT	-	Set duty cycle (enter in decimal)
STEP	-	Stop motor (optional - set speed)
RUN	-	Start motor (optional - set speed)
CLR	-	Stop motor, clear segment count
MEM	-	Store proportional and integral gains

SPEED calculates instantaneous speed. Obtains interval time from Timer 2 data stored in memory by EXT4 interrupt. Returns carry set except if no interval time is observed after 256 attempts. In that case speed is reported as 0000.

WIDTH calculates and stores integral of error, and returns new pulse width.

LDT1 loads pulse width to Timer 1. If entry value is zero or negative LDT1 replaces entry value with 0001 before loading timer, and returns MINUS flag set.

DECBI converts a packed decimal input byte in (L) and returns the binary equivalent of the low byte in (DE). The zero flag is set if the low byte was zero.

SMULT calculates (A) times (E) and returns the two byte product in HL. At entry register D must contain 00 if (E) is positive or FF if (E) is negative. Thus (DE) is a two byte twos complement value.

SCUML calculates (HL) + (A) (E). It is an alternate entry to SMULT, and omits clearing the product before multiplying.

DIVID calculates (DE)/(HL). The dividend must be no greater than twice the divisor. The quotient is represented as one bit left of the binary point and a 15 bit fraction.

7.3.5 Speed Logging

The solution for motor speed control in Figures 7-21 and 7-22 does not include speed logging. You can add this feature, using a technique similar to that in the logging voltmeter. To do this you will want a subroutine that records only once in ten or sixteen cycles through the main loop, because of the fairly long reaction times of the motor. Section 7.3.6, following the program solution, discusses some of the results you will see with the program given.

MOTOR CONTROL - INITIALIZE

ADDR		CODE											
8 20	0	3	E	M	V	I	A	,	8	0			Program Ports
	1	8	0										Port 1B Out
	2	D	3	O	U	T	C	N	T	1			
	3	0	7										
	4	3	E	M	V	I	A	,	9	2			
	5	9	2										
	6	D	3	O	U	T	C	N	T	2			
	7	0	F										
	8	3	E	M	V	I	A	,	2	5			Program Timer 0
	9	2	5										High byte
	A	D	3	O	U	T	T	I	M	C	T		Mode 2
	B	1	7										decimal
	C	3	E	M	V	I	A	,	5	0			Load Timer 0
	D	5	0										for 2.4414 ms
	E	D	3	O	U	T	T	I	M	0			(256 interrupts
	F	1	4										= 10/16 second)
8 21	0	3	E	M	V	I	A	,	7	A			Program Timer 1
	1	7	A										Both bytes
	2	D	3	O	U	T	T	I	M	C	T		Mode 5
	3	1	7										Binary
	4	3	E	M	V	I	A	,	B	0			Program Timer 2
	5	B	0										Both bytes
	6	D	3	O	U	T	T	I	M	C	T		Mode 0
	7	1	7										Binary
	8	A	F	X	R	A	A						Start Timer 2
	9	D	3	O	U	T	T	I	M	2			
	A	1	6										
	B	D	3	O	U	T	T	I	M	2			
	C	1	6										
	D	C	D	C	A	L	L		C	L	R		Clear segment
	E	2	0										count and
	F	8	1										stop motor
8 22	0	3	E	M	V	I	A	,	1	3			Enable interrupts
	1	1	3										Timer 0, Timer 1,
	2	D	3	O	U	T	P	O	R	T	2	C	EXT 4
	3	0	E										
	4	C	3	J	M	P	8	2	8	0			
	5	8	0										
	6	8	2										
	7												
	8												Figure 7-21a

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

MOTOR CONTROL - INTERRUPT SERVICE

ADDR		CODE			Comment
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
822	8	F5	PUSH	PSW	RST 5 —
	9	E5	PUSH	H	Timer 0 interrupt
	A	CD	CALL	STIM0	
	B	40			
	C	82			
	D	C3	JMP	EXIT	
	E	39			
	F	82			
8 2.3	0	F5	PUSH	PSW	RST 6
	1	E5	PUSH	H	Timer 1 or EXT4
	2	DB	IN	PORT2B	Read interrupt
	3	0D			status byte
	4	E6	ANI	02	Mask for
	5	02			Timer 1
	6	CD	CALL	STIM1	Call service routine
	7	58			
	8	82			
823	9	D3	OUT	CNT2	EXIT - Reenable
	A	0F			and clear interrupt
	B	E1	POP	H	7
	C	F1	POP	PSW	
	D	FB	EI		
	E	C9	RET		
	F	00			
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				

Figure 7-21b

MOTOR CONTROL - SERVICE TIMERS

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR	CODE		Instruction	Comment
8240	2	1	LXI H, 83A1	STIMO
1	A	1		Address load
2	8	3		and output
3	7	E	MOV A, M	motor control byte
4	D	3	OUT PORT 1C	to set PIC1 low
5	0	6		if RUN, high if STOP
6	3	E	MVI A, 01	To reenable
7	0	1		Timer 0 at exit.
8	2	B	DCX H	Address and
9	3	5	DCR M	decrement
A	C	0	RNZ	time counter
B	2	A	LHLD 83A3	At 00 = 5/8 second
C	A	3		copy decimal
D	8	3		segment count
E	2	2	SHLD 83A5	to display area
F	A	5		
8250	8	3		
1	2	1	LXI H, 0000	Clear decimal
2	0	0		segment counter
3	0	0		
4	2	2	SHLD 83A3	
5	A	3		
6	8	3		
7	C	9	RET	
8258	C	A	JZ SEXT4	STIM1
9	6	0		Jump if EXT4
A	8	2		Else (A) = 02
B	D	3	OUT PORT 1C	Turn motor off
C	0	6		
D	3	E	MVI A, 03	To reenable Timer 0
E	0	3		
F	C	9	RET	
8 0				
1				
2				
3				
4				
5				
6				
7				
8				Figure 7-21c

EXT 4 INTERRUPT SERVICE

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR	CODE					
8260	3E		MVI	A,80		Latch, read,
1	80					and store
2	D3		OUT	TIMCT		Timer 2 data
3	17					
4	DB		IN	TIM2		
5	16					
6	6F		MOV	L,A		
7	DB		IN	TIM2		
8	16					
9	67		MOV	H,A		
A	22		SHLD	83A7		
B	A7					
C	83					
D	21		LXI	H,83A2		Address and
E	A2					increment binary
F	83					segment count
8270	34		INR	M		
1	23		INX	H		Address and
2	7E		MOV	A,M		increment decimal
3	C6		ADI	01		segment count
4	01					
5	27		DAA			
6	77		MOV	M,A		
7	23		INX	H		
8	7E		MOV	A,M		
9	CE		ACI	00		
A	00					
B	27		DAA			
C	77		MOV	M,A		
D	3E		MVI	A,09		To reenable and
E	09					clear EXT 4
F	C9		RET			To exit
8 0						
1						
2						
3						
4						
5						
6						
7						
8						Figure 7-21d

MOTOR CONTROL - MAIN LOOP

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR	CODE	Instruction	Comment
8280	E5	PUSH H	Save avg speed
1	DB	IN PORT 0A	Test keyboard,
2	00		(FF if no key)
3	3C	INR A	
4	C4	CNZ KYTIM	If key pressed
5	0[illegible]		accept and
6	81		process input
7	2A	LHLD 83A1	(H)← Segment Count
8	A1		(L)← Motor Control
9	83		02 = Stop 00 = Run
A	7D	MOV A, L	
B	B7	ORA A	
C	C2	JNZ 82A0	If stopped go to
D	A0		display segment
E	82		count
F	00	NOP	
8290	CD	CALL SPEED	(HL)← instantaneous
1	C0		speed
2	82		
3	E5	PUSH H	(ST)← speed
4	CD	CALL WIDTH	(HL)← pulse width
5	E0		
6	82		
7	CD	CALL LDT1	Load Timer 1
8	60		for pulse width
9	81		
A	D1	POP D	(DE)← speed
B	7A	MOV A, D	Output speed
C	D3	OUT PORT 1B	to D/A for
D	05		voltmeter
E	00	NOP / XCHG	XCHG to display
F	00	NOP	instantaneous speed
8 0			
1		CONTINUED	
2			
3			
4			
5			
6			
7			
8			

Figure 7-21e

MICROCOMPUTER TRAINING SYSTEM — CODING SHEET — INTEGRATED COMPUTER SYSTEMS

MOTOR CONTROL - MAIN LOOP - DISPLAY

ADDR	CODE	INSTRUCTION	OPERAND	COMMENTS
82A0	7C	MOV	A, H	If running display
1	CD	CALL	DBYTE	speed or width
2	95			If stopped display
3	02			segment count
4	E1	POP	H	(HL) ← old avg speed
5	7D	MOV	A, L	
6	2A	LHLD	'83A5'	(HL) ← latest
7	A5			average speed
8	83			from interrupts
9	BD	CMP	L	Test for equal
A	C4	CNZ	DWORD	Display average
B	D1			speed if ready
C	02			
D	C3	JMP	8280	Back to start
E	80			
F	82			
82B0				82B0-BF not used
1				
2				
3				
4				
5				
6				
7				
8				
9				
A				
B				
C				
D				
E				
F				
8 0				
1				
2				
3				
4				
5				
6				
7				
8				

Figure 7-21f

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

SUBROUTINE SPEED

ADDR		CODE	Instruction	Comment
8 2C	0	0E	MVI C,00	Loop counter
	1	00		
	2	2A	LHLD 83A7	
	3	A7		(DE)← Timer 2 Data
	4	83		
	5	EB	XCHG	
82C	6	0D	DCR C	Count tries and
	7	C8	RZ	exit if not moving
	8	2A	LHLD 83A7	(HL)← Timer 2 Data
	9	A7		
	A	83		
	B	7B	MOV A,E	
	C	95	SUB L	(HL) ← initial time
	D	6F	MOV L,A	− new time
	E	7A	MOV A,D	= segment interval
	F	9C	SBB H	
8 2D	0	67	MOV H,A	
	1	B5	ORA L	Test and loop if
	2	CA	JZ 82C6	time not changed
	3	C6		
	4	82		
	5	11	LXI D,03E8	03E8/time
	6	E8		= speed as XX.XX
	7	03		
	8	CD	CALL DIVID	(HL)←(DE)/(HL)
	9	A0		= speed
	A	81		
	B	37	STC	Mark as correct
	C	C9	RET	result
	D			
	E			
	F			
8	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			

Figure 7-21g

MOTOR CONTROL - SUBROUTINE WIDTH

ADDR		CODE	INSTRUCTION	COMMENTS
8 2E	0	7D	MOV A, L	(HL) = Speed
	1	17	RAL	MSB of low byte
	2	3A	LDA 83A9	to CY for rounding
	3	A9		(A) ← desired speed
	4	83		
	5	9C	SBB H	
	6	5F	MOV E, A	(DE) ← speed error
	7	9F	SBB A	as two bytes
	8	57	MOV D, A	
	9	D5	PUSH D	Save error
	A	3A	LDA 83AA	(A) ← G_i
	B	AA		(integral gain)
	C	83		
	D	2A	LHLD 83AC	(HL) ← old integral
	E	AC		
	F	83		
8 2F	0	CD	CALL SCUML	(HL) ← (HL) + (A)(DE)
	1	93		$= \Sigma G_i E$
	2	81		
	3	D1	POP D	(DE) ← Error
	4	B4	ORA H	Quit if integral
	5	F8	RMINUS	negative
	6	22	SHLD 83AC	Store integral
	7	AC		
	8	83		
	9	3A	LDA 83AB	(A) ← G_p
	A	AB		(proportional gain)
	B	83		
	C	CD	CALL SCUML	(HL) ← (HL) + (A)(DE)
	D	93		$= \Sigma G_i E + G_p E$
	E	81		
	F	C9	RET	
8	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			Figure 7-21h

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

MOTOR CONTROL - KYTIM

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE	Instruction	Operand	Comment
8 10	0	00	NOP		for DI during debug
	1	CD	CALL	ENTWD	(A)← command
	2	46			(HL)← keyboard data
	3	03			
	4	EB	XCHG		(DE)← keyboard data
	5	21	LXI	H, 8108	Address dispatch table
	6	08			
	7	81			
	8	85	ADD	L	Dispatch address to stack
	9	6F	MOV	L, A	
	A	6E	MOV	L, M	
	B	E5	PUSH	H	
	C	EB	XCHG		(HL)← keyboard data
	D	CD	CALL	DECBI	(E)← binary of (L)
	E	80			(D) ← 00
	F	81			Zero set if (E) = 00
8 11	0	7A	MOV	A, D	Clear A, preserve flag
	1	FB	EI		Needed during debug
	2	32	STA	8000	Clear log address
	3	00			to start a new log
	4	80			
	5	00	NOP		
	6	00	NOP		
811	7	C9	RET		
811	8	30	MEM		Store gains
	9	17	REG		Undefined
	A	17	ADDR		Undefined
	B	23	STEP		Stop motor
	C	27	RUN		Start motor
	D	34	NEXT		Set duty cycle
	E	17	BRK		Undefined
	F	20	CLR		Clear count, stop
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				Figure 7-22a

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

KYTIM - CLR, STEP, RUN, MEM, NEXT

ADDR		CODE		Label	Mnemonic	Operand	Comments
8 12	0	3	2		STA	83A2	CLR - Clear binary
	1	A	2				segment count and
	2	8	3				stop motor
812	3	3	E		MVI	A, 02	STEP - Stop motor
	4	0	2				
	5	D	3		OUT	PORT1C	
	6	0	6				
812	7	3	2		STA	83A1	RUN - Store motor
	8	A	1				control byte
	9	8	3				00 = RUN 02 = STOP
	A	C	8		RZ		Exit if no data
	B	7	B		MOV	A, E	Store binary
	C	3	2		STA	83A9	equivalent of
	D	A	9				desired speed
	E	8	3				
	F	C	9		RET		
8 13	0	2	2		SHLD	83AA	MEM - Store gains
	1	A	A				
	2	8	3				
	3	C	9		RET		
813	4	3	E		MVI	A, 32	NEXT - Duty cycle
	5	3	2				Multiply binary duty
	6	C	D		CALL	SMULT	cycle (DE)
	7	9	0				x 32 for pulse width
	8	8	1				
	9	2	2		SHLD	83AC	Store as integral
	A	A	C				of error
	B	8	3				
	C	C	D		CALL	LDT1	Load timer 1
	D	6	0				for pulse width
	E	8	1				
	F	C	9		RET		
8	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						Figure 7-22b

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

A D D	R	CODE		Opcode	Operand	Comment
8 16	0	2B		DCX	H	Test for
	1	7C		MOV	A, H	pulse width ~ 0
	2	B7		ORA	H	
	3	23		INX	H	
	4	F2		JPLUS	816A	
	5	6A				
	6	81				
	7	21		LXI	H, 0001	If pulse width ≤ 0
	8	01				make width 0001
	9	00				
816	A	7D		MOV	A, L	Load timer 1
	B	D3		OUT	TIM1	with pulse width
	C	15				
	D	7C		MOV	A, L	
	E	D3		OUT	TIM1	
	F	15				
8 17	0	C9		RET		
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	A					
	B					
	C					
	D					
	E					
	F					
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					Figure 7-22c

MOTOR CONTROL - DECBI

ADDR	CODE	Instruction	Remarks
8 18 0	7D	MOV A, L	Low byte of input
1	E6	ANI F0	(A)← high digit
2	F0		
3	1F	RAR	(A)← 1/2 high digit
4	5F	MOV E, A	(E)← 1/2 high
5	1F	RAR	(A)← 1/4 high
6	1F	RAR	(A)← 1/8 high
7	85	ADD L	(A)← dec + 1/8
8	93	SUB E	(A)← dec - 3/8
9	5F	MOV E, A	= binary
A	16	MVI D, 00	(D)← 00
B	00		
C	C9	RET	
D			
E			
F			
8 0		ENTER WITH	
1		(HL) = INPUT DATA	
2		ASSUME (L) = PACKED DECIMAL	
3		RETURN	
4		(A) = (E) =	BINARY EQUIVALENT
5			OF (L)
6		(D) = 00	
7		B, C, H, L	PRESERVED
8			
9		CY CLEAR	
A		ZERO SET	IF (E) = 00
B			
C			
D			
E			
F			
8 0			
1			
2			
3			
4			
5			
6			
7			
8			

Figure 7-22d

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR	CODE	INSTRUCTION	COMMENT
8190	21	LXI H, 0000	SMULT
1	00		Clear product
2	00		
8193	B7	ORA A	SCUML Clear CY
4	C8	RZ	Exit when multiplier = 0
5	1F	RAR	(CY)←LSB of multiplier
6	D2	JNC 819A	Skip add if LSB=0
7	9A		If LSB = 1 add
8	81		multiplicand
9	19	DAD D	into product
819A	EB	XCHG	Shift multiplicand
B	29	DAD H	
C	EB	XCHG	
D	C3	JMP SCUML,	Loop to test
E	93		multiplier
F	81		
8 0			
1			
2		ENTER SMULT	OR SCUML
3		(A) = MULT	IPLIER
4		(E) = MULT	IPLICAND
5		(D) = 00 O	R FF(NEGATIVE)
6			
7		RETURN	
8		SMULT (HL	)←(A) * (E)
9		SCUML (HL	)←(HL)+(A)*(E)
A			
B		(A) = 00	
C		ZERO SET ;	CARRY CLEAR
D			
E		(BC) PRES	ERVED
F			
8 0			
1			
2			
3			
4			
5			
6			
7			
8			Figure 7-22e

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

DIVID (HL)<--(DE)/(HL)

ADDR		CODE		LABEL	INSTRUCTION	OPERAND	COMMENT
81A	0	3	E		MVI	A,10	(A)← bit element
	1	1	0				
	2	C	5		PUSH	B	Save (BC)
	3	4	4		MOV	B,H	(BC)← divisor
	4	4	D		MOV	C,L	
	5	2	1		LXI	H,0000	Clear quotient
	6	0	0				
	7	0	0				
81A	8	F	5	DV1	PUSH	PSW	Save bit count
	9	2	9		DAD	H	Shift quotient
	A	7	B		MOV	A,E	Test for
	B	9	1		SUB	C	divisor > remainder
	C	7	A		MOV	A,D	
	D	9	8		SBB	B	
	E	D	A		JC	DV2	Jump if
	F	B	6				divisor > remainder
81B	0	8	1				
	1	2	3		INX	H	Enter 1 into quotient
	2	5	7		MOV	D,A	
	3	7	B		MOV	A,E	(DE)← new remainder
	4	9	1		SUB	C	
	5	5	F		MOV	E,A	
81B	6	E	B	DV2	XCHG		
	7	2	9		DAD	H	Shift remainder
	8	E	B		XCHG		
	9	F	1		POP	PSW	(A)← bit count
	A	3	D		DCR	A	
	B	C	2		JNZ	DV1	
	C	A	8				
	D	8	1				
	E	C	1		POP	B	
	F	C	9		RET		
8	0						
	1				ENTER	(DE)	= DIVIDEND
	2					(HL)	= DIVISOR
	3				DIVIDEND		< 2*DIVISOR
	4				RETURN	(HL)	= QUOTIENT
	5				AS	.XXXX	
	6					(DE)	= REMAINDER
	7					(BC)	- PRESERVED
	8				(A) = 00	ZE	RO SET, CY PRESERVA

Figure 7-22f

7.3.6 Motor Control Program Operation

To operate closed loop control you must enter three data bytes; proportional gain, integral gain, and desired speed. Gains are entered as two consecutive bytes with MEM.

0801	MEM	Set proportional gain = 8
		Set integral gain = 1
50	RUN	Set desired speed 50 rps

The system allows speed requests from one to 99 rps, but will not operate successfully at speeds much less than 10 rps. Experiment to see the average speed respond to various speed requests. Connect the voltmeter from the D/A output to ground to see an analog indication of instantaneous speed. Although the original program design intent was to display instantaneous speed, this varies so much and so quickly that it is not very readable, so a display of the pulse width is generally more interesting. In the given solution this is obtained simply by omitting an XCHG instruction at 829D. An oscilloscope display of the pulse width is especially interesting.

Observe the response of average speed as you place a load on the motor. This is most easily done by squeezing the motor shaft between your fingers. Do not attempt it by dragging on the optical disc, because if you start it slipping it will wear the hole that mounts it on the shaft.

When you apply the load, the motor will slow down but the closed loop control will apply more power to restore the speed. The range of loads over which speed can be maintained is fairly impressive, although unfortunately we have no means of measuring load.

When you release the load the motor will speed up rapidly, and the control system will hunt for the desired speed just as the voltage control system hunted for a desired voltage. The hunting is easily observed from the sound of the motor. Try different gain values and observe their effect.

If you stall the motor by holding the shaft it will not restart. This is because the integral will increase to a large positive value while the motor is stalled. In following cycles of the main loop a value greater than 7FFF will be calculated for the integral; this is taken to be negative and will not be stored. LDT1 will substitute a minimum pulse width which will not start the motor. You can restart the motor by pressing NEXT, which clears the error integral, allowing closed loop control to function again. SPEED returns to the main loop with carry clear if the motor is stalled. Develop a program modification to enter a 50% duty cycle if the motor is stalled.

The CLR key was defined to stop the motor and clear the binary segment count. This allows observation of the coasting distance after power is removed. Measure the relationship between speed and coasting distance.

7.4 MOTOR CONTROL BY VARIABLE VOLTAGE

We can control the motor by varying the voltage amplitude instead of using pulse width modulation. The output voltage from the power transistor can be controlled by varying the drive to its optical coupler. Remove the connection from MOT CTL+ to +5 volts, and connect ANALOG OUT to MOT CTL+. Now the program can vary the voltage. Only two trivial changes to the program are required:

a) Timer 1 has no function, since power is to be turned on whenever the motor is running. Disable the Timer 1 inerrupt by changing the initialization step that originally enabled it.

b) Output the high byte of pulse width to the D/A converter instead of the high byte of speed.

These two changes are shown in Figure 7-23.

Now run the program as before.

0801	MEM	Set gain
50	RUN	Request speed

Before the motor will start the integral must build up to a much higher value than for pulse width modulation. This is principally because the optical coupler that drives the power transistor must receive a voltage input above about 1.5 volts before it will start to turn the power transistor on. The control will not be as smooth, because the optical coupler makes a much larger change in the motor voltage. Try different speed requests and see how small a change in the control voltage is required. Connect your voltmeter across the motor and observe that the very small change in control voltage displayed by the computer makes a much larger change in the motor voltage.

You can also switch between PWM control and voltage control by Keyboard command. Define REG to enter a speed for variable voltage control; RUN to enter speed for PWM operation. REG must disable Timer 1 interrupt; RUN must enable it. During PWM operation output FF to the D/A converter, allowing PORT1C1 to control the motor. One possible solution is given in Figure 7-24.

PATCHES FOR VARIABLE VOLTAGE CONTROL

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A D D	R	CODE		Instruction	Comments
822	0	3E		MVI A,11	Enable EXT4
	1	11	*		and Timer 0
	2	D3		OUT PORT 2C	interrupts
	3	0E			(not Timer 1)
	4				
	5				
	6				
	7				
	8				
	9				
	A				
829	B	7C	*	MOV A,H	Voltage Control
	C	D3		OUT PORT 1B	("Pulse Width")
	D	05			(instead of speed)
	E				to D/A Output
	F				
8	0		*	CHANGES FROM PWM	
	1			PROGRAM (FIGURE 7-21)	
	2			FOR VARIABLE VOLTAGE	
	3			MOTOR CONTROL	
	4				
	5			CONNECT MOT CTL+	
	6			TO ANALOG OUT	
	7			INSTEAD OF +5V	
	8				
	9				
	A				
	B				
	C				
	D				
	E				
	F				
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				Figure 7-23

RUN (=PWM) AND REG (=VOLTAGE) IN KYTIM

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		MNEMONIC	OPERAND	COMMENTS
8 14	0	3	E	MVI	A, 80	RUN = PWM
	1	8	0			
8 14	2	3	2	STA	83A1	REG = Voltage
	3	A	1			(Store 00 or FC)
	4	8	3			
	5	1	7	RAL		(A)←03 if RUN
	6	3	E	MVI	A, 01	(A)←02 if REG
	7	0	1			
	8	1	7	RAL		
	9	F	3	DI		Enable or Disable
	A	D	3	OUT	CNT2	Timer 1 Interrupt
	B	0	F			
	C	F	B	EI		Exit if no speed
	D	C	8	RZ		entered
	E	7	B	MOV	A, E	Store binary
	F	3	2	STA	83A9	equivalent of
8 15	0	A	9			desired speed
	1	8	3			
	2	C	9	RET		
	3					
	4					
	5					
	6			PATCHES	TO	DISPATCH
	7			TABLE	IN	KYTIM
	8					
8 11	9	4	2	REG		
	A					
	B					
8 11	C	4	0	RUN		
	D					
	E					
	F					
8	0			MOTOR	CONTROL	BYTE
	1			IS	NOW DEFINED	AS
	2					
	3			02 =	STOP	(CLR or STEP)
	4			80 =	PWM	(RUN)
	5			00 =	VOLTAGE	(REG)
	6					
	7					
	8					Figure 7-24a

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADD	R	CODE		Instruction	Comment
8 28	0	E5		PUSH H	Save Avg Speed
	1	DB		IN PORT 0A	
	2	00			
	3	3C		INR A	
	4	C4		CNZ KYTIM	
	5	00			
	6	81			
	7	2A		LHLD 83A1	(H)← Segment Count
	8	A1			(L)← Motor Control
	9	83			02=Stop 00=Voltage
	A	7D		MOV A, L	80 = PWM
	B	87	*	ADD A	Set CY if PWM
	C	C2		JNZ 82A0	Zero unless stop
	D	A0			
	E	82			
	F	F5	*	PUSH PSW	Save CY if PWM
8 29	0	CD		CALL SPEED	(HL)← instantaneous
	1	C0			speed
	2	82			
	3	00	*	NOP	
	4	CD		CALL WIDTH	(HL)← Pulse Width
	5	E0			or
	6	82			(H)← Variable Voltage
	7	CD		CALL LDT1	Load Timer 1
	8	60			for on time
	9	81			
	A	00	*	NOP	} FF if PWM
	B	F1	*	POP PSW	Voltage if
	C	9F	*	SBB A	voltage control
	D	B4	*	ORA H	To D/A output
	E	D3	*	OUT PORT 1B	for MOT CTL+
	F	05	*		
8	0				
	1				
	2		*	CHANGED FROM FIGURE 7-21e	
	3				
	4			REMAINDER OF MAIN LOOP	
	5			(82A0-82AF)	
	6			SAME AS FIGURE 7-21f	
	7				
	8				Figure 7-24b

7.5 POWER TRANSISTOR DISSIPATION

A transistor dissipates (wastes) much more power when operating in its linear region than when it is used as a switch. You can measure this effect by attaching the thermistor to the power transistor and measuring its temperature. You must time share the digital/analog converter between the driving and measuring functions. Each time an average speed measurement is completed, make a temperature measurement. Use a tracking voltmeter approach, storing the D/A value in memory and trying only one value on each attempt. This gives the least interference with the motor drive. After the temperature measurement, restore the motor drive value to the D/A converter. Figure 7-25 shows a program solution for the Main Loop and a new subroutine to display speed and measure temperature. In the given solution the thermistor voltage is displayed in hexadecimal, with no conversion to degrees.

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

ADDR	CODE	INSTRUCTION	COMMENT
82A 0	47	MOV B, A	(B)← Motor Drive
1	7D	MOV A, L	(A)← Old avg speed
2	2A	LHLD 83A5	(HL)← Latest
3	A5		average speed
4	83		
5	BD	CMP L	
6	C4	CNZ SPTMP	If new average
7	C0		speed, display
8	81		speed & temperature
9	C3	JMP 8280	
A	80		
B	82		
C			
D			
E			
F			
8 0		REPLACE LAST PART OF	
1		MOTOR CONTROL MAIN LOOP	
2		(FIGURE 7-21b)	
3			
4		SPTMP (FIGURE 7-25b)	
5		DISPLAYS AVERAGE SPEED	
6		AND ATTEMPTS A/D	
7		MEASUREMENT OF	
8		TEMPERATURE	
9			
A			
B			
C			
D			
E			
F			
8 0			
1			
2			
3			
4			
5			
6			
7			
8			Figure 7-25a

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR	CODE	MNEMONIC	OPERAND	COMMENTS
8 1C 0	3A	LDA	83AE	(A)← Previous temperature
1	AE			measurement
2	83			
3	D3	OUT	PORT1B	to D/A
4	05			
5	4F	MOV	C, A	Save value
6	3C	INR	A	Increase for
7	32	STA	83AE	next trial
8	AE			
9	83			
A	C5	PUSH	B	
B	CD	CALL	DWORD	Display average
C	D1			speed
D	02			
E	C1	POP	B	
F	DB	IN	PORT2B	Test D/A
8 1D 0	0D			comparator
1	E6	ANI	08	Set zero if D/A
2	08			too low
3	78	MOV	A, B	Restores motor
4	D3	OUT	PORT1B	drive to D/A
5	05			
6	C8	RZ		Exit if D/A low
7	79	MOV	A, C	Display good
8	CD	CALL	DBYTE	value for
9	95			temperature
A	02			
B	3D	DCR	A	Decrease for
C	32	STA	83AE	next trial
D	AE			
E	83			
F	C9	RET		
8 0				
1		DISPLAYS		(HL) = SPEED
2		MAKES ONE		TRIAL
3		MEASUREMENT OF VOLTAGE		
4		WITH TRACKING VOLTMETER		
5		DISPLAYS VOLTAGE IF		
6		GREATER THAN INPUT		
7		SENDS (B) TO D/A BEFORE		
8		RETURN		

Figure 7-25b

7.6 REVIEW

In the Motor Control exercise of Chapter 7 we have used many of the interfacing techniques covered in this course:

- A/D Input by interval measurement (for instantaneous speed).
- A/D Input by frequency counting (for average speed).
- A/D Input by voltage conversion (for temperature).
- D/A Output by pulse width modulation.
- D/A Output by parallel conversion to voltage.
- Use of a counter to measure a time interval.
- Use of a counter to generate a timed-interrupt.
- Use of an optical coupler (the slot sensor) for input, generating an interrupt in response to a mechanical input.
- Use of an optical coupler to drive an external device (the motor) and exclude noise from the computer.

We have also used all of the Input/Output techniques:

- Programmed Input - Keyboard data and temperature measurement.
- Programmed Output - Variable voltage control.
- DMA Output - Displays
- Interrupt Driven Input - Speed measurement.
- Interrupt Driven Output - PWM control.

We have used the microcomputer to operate a closed loop control system and simultaneously monitor its performance by displaying average speed and temperature. We have provided for modifying the system operation by changing gains in the control equation and by switching between PWM control and variable voltage control. Finally, we time-shared a single device (the D/A - A/D converter) for input and output operations. The fact that all of these functions are accomplished within 512 bytes of program memory (8100 - 82FF) demonstrates the capability of a very small microcomputer in a control system.

MICROCOMPUTER INTERFACING WORKBOOK

APPENDIX A

REFERENCE FIGURES

This page intentionally left blank.

APPENDIX A

This Appendix contains the following figures for reference:

This page intentionally left blank.

PORT ADDRESSES AND ASSIGNMENTS

ADDRESS	PORT NAME	FUNCTION
00	PORT 0A	MTS Keyboard Input
01	PORT 0B	Unassigned except 0B0
02	PORT 0C	See column at right
03	CNT 0	Control Port for MTS 8255
04	PORT 1A	LED and Driver Outputs
05	PORT 1B	D/A Output or A/D Input
06	PORT 1C	See column at right
07	CNT 1	Control Port for 8255 # 1
0C	PORT 2A	Unassigned
0D	PORT 2B	Interrupt Status Input
0E	PORT 2C	Interrupt Enable Output
0F	CNT 2	Control Port for 8255 # 2
14	TIM 0	Timer 0
15	TIM 1	Timer 1
16	TIM 2	Timer 2
17	TIM CT	Control Port for 8253

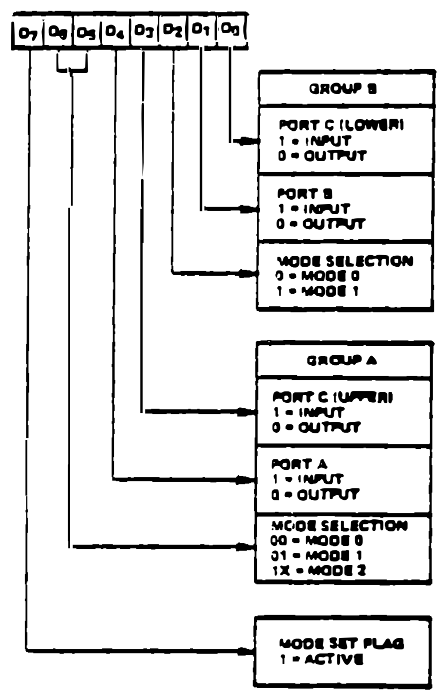
SPECIAL ASSIGNMENTS FOR 0C AND 1C		
0C7	Display Control	(1 = On)
0C6	Enable Command Keys	(0 = On)
0C5	Enable Keys 8-F	(0 = On)
0C4	Enable Keys 0-7	(0 = On)
0C3	Zero Indicator	
0C2	Carry Indicator	
0C1	Monitor Enable	
0C0	Cassette Modem Out	
0B0	Cassette Modem In	
1C7-4	Unassigned	
1C3	Interrupt (If Enabled by 2C6)	
1C2	Unassigned	
1C1	Motor Drive Buffer	(1 = On)
1C0	D/A Control (1 = Automatic A/D)	

8255 PROGRAMMING CONTROL BYTES (WRITE TO 8255 CONTROL PORT)

CONTROL BYTE	PORT A	PORT B	PORT C0-C3	PORT C4-C7	USE WITH 8255 # 0	1	2
80	Out	Out	Out	Out		D/A	
81	Out	Out	In	Out		●	
82	Out	In	Out	Out		A/D	•
83	Out	In	In	Out		A/D	
88	Out	Out	Out	In		D/A	
89	Out	Out	In	In		●	
8A	Out	In	Out	In		A/D	
8B	Out	In	In	In		A/D	
90	In	Out	Out	Out	•	D/A	
91	In	Out	In	Out	•	●	
92	In	In	Out	Out	•	A/D	•
93	In	In	In	Out	•	A/D	
98	In	Out	Out	In		D/A	
99	In	Out	In	In		●	
9A	In	In	Out	In		A/D	
9B	In	In	In	In		A/D	

• ~~Generally only these control bytes should be used for normal operation.~~

● Forbidden configurations



Mode Definition Format

Figure A-1

This page intentionally left blank.

STATUS AND COMMAND BYTES

INTERRUPT SOURCE	STATUS BYTE OBTAINED BY IN PORT 2B (see Note 2)									COMMAND BYTE WRITTEN BY OUT CNT 2 (see Note 1)	
	BINARY								HEX	DISABLE	ENABLE
Timer 0	0	X	X	X	X	X	X	1	01	00	01
Timer 1	0	X	X	X	X	X	1	X	02	02	03
Timer 2	0	X	X	X	X	1	X	X	04	04	05
A/D Comparator	0	X	X	X	1	X	X	X	08	06	07
EXT 4	0	X	X	1	X	X	X	X	10	08	09
EXT 5	0	X	1	X	X	X	X	X	20	0A	0B
Port 1C3	(see Note 3)									0C	0D

Note 1: Disable or enable command byte must be output to CNT 2 to clear the interrupt flip flop for Timer 0, Timer 1, EXT 4, or EXT 5. Disable or enable for A/D Comparator clears the interrupt in automatic A/D mode only.

Note 2: The hex values shown assume all other bits are 0. ANI (hex value) will give zero if the interrupt is not present.

Note 3: Port 1C3 does not appear in the status byte. It is read as XXXX1XXX by IN PORT1C. It is cleared by reading PORT1A in strobed input mode (mode 1 or mode 2) or by writing to PORT1A in strobed output mode (mode 1 or mode 2). Otherwise it can be cleared or set by writing 06 or 07 to CNT1. The interrupt enable for Port 1C3 is cleared or set by writing 0C or 0D to CNT2, but this does not change the data at Port 1C3.

Status and Command Bytes

Figure A-2

This page intentionally left blank.

```
Program 8255's - 1B out

  3E     MVI     A,80
  80
  D3     OUT     CNT1
  07
  3E     MVI     A,92
  92
  D3     OUT     CNT2
  0F
```

```
Program 8255's - 1B in

  3E     MVI     A,82
  82
  D3     OUT     CNT1
  07
  3E     MVI     A,92
  92
  D3     OUT     CNT2
  0F
```

Standard Programming for 8255's

Figure A-3

This page intentionally left blank.

Timer Control Byte Structure

```
7  6  5  4  3  2  1  0

                        0 = binary count
                        1 = decimal count

                        000   Mode 0
                        001   Mode 1
                        X10   Mode 2
                        X11   Mode 3
                        100   Mode 4
                        101   Mode 5

                    00   Latching command (see Section 3.6.3)
                    01   Read/Load least significant byte only
                    10   Read/Load most significant byte only
                    11   Read/Load least significant byte
                         First, then least significant byte

                    00   Select timer 0
                    01   Select timer 1
                    10   Select timer 2
                    11   Illegal
```

Timer Control Byte Structure

Figure A-4

This page intentionally left blank.

Timer 0

	Mode 0	1	2	3	4	5
Latch	00	00	00	00	00	00
Read/Load LSB	10	12	14	16	18	1A
Read/Load MSB	20	22	24	26	28	2A
Read/Load Both (LSB first)	30	32	34	36	38	3A

Timer 1

	Mode 0	1	2	3	4	5
Latch	40	40	40	40	40	40
Read/Load LSB	50	52	54	56	58	5A
Read/Load MSB	60	62	64	66	68	6A
Read/Load Both (LSB first)	70	72	74	76	78	7A

Timer 2

	Mode 0	1	2	3	4	5
Latch	80	80	80	80	80	80
Read/Load LSB	90	92	94	96	98	9A
Read/Load MSB	A0	A2	A4	A6	A8	AA
Read/Load Both (LSB first)	B0	B2	B4	B6	B8	BA

Control Bytes shown set binary counting

Add 1 for decimal counting

Write control byte to TIMCT, Port 17

Latching control byte does not affect mode

Timer Control Bytes

Figure A-5

This page intentionally left blank.

Mode	Output after mode set	Starts counting	Output goes low	Output goes high	Count restarted	Comments
0 Interrupt	Low	When final byte loaded	At mode set	At zero	By reloading	Output is set low by setting mode or by reloading.
1 One Shot	High	After gate rising edge	After gate rising edge	At zero	By gate rising edge	Can be preloaded during counting. Present period not affected.
2 Rate Generator	High	When final byte loaded	At count=1	At zero	At zero or by gate rising edge	New value effective for next period.
3 Square Wave	High	When final byte loaded	At n/2 or (n + 1)/2	At zero	At zero or by gate rising edge	If loaded while counting new period is effective for next half of total period.
4 Software Strobe	High	When final byte loaded	At zero	At next clock after zero	By reloading	
5 Hardware Strobe	High	After gate rising edge	At zero	At next clock after zero	By gate rising edge	If loaded while counting new period is effective after next gate rising edge.

8253 Timer Modes

Figure A-6

This page intentionally left blank.

Binary Count (Hex representation)	Decimal Count	Time (milliseconds)
0100	256	0.125
0200	512	0.250
0400	1024	0.500
0800	2048	1.000
1000	4096	2
1800	6144	3
2000	8192	4
2800	(10240)	5
3000	(12288)	6
3800	(14336)	7
4000	(16384)	8
4800	(18432)	9
5000	(20480)	10
A000	(40960)	20
F000	(61440)	30
0000	(65536)	32
		Time (seconds)
1F40	8000	1/256
0FA0	4000	1/512
07D0	2000	1/1024

Count to Time Conversion

Figure A-7

This page intentionally left blank.

MICROCOMPUTER INTERFACE WORKBOOK

APPENDIX B

CASSETTE INTERFACE INSTRUCTIONS
AND
PROGRAM CASSETTE LIBRARY

I M P O R T A N T

THE TAPE PROVIDED IS A MASTER TAPE. IT SHOULD BE COPIED TO A WORKING TAPE WHICH SHOULD BE USED FOR LOADING OF PROGRAMS. KEEP THE MASTER TAPE IN A SAFE PLACE SO THAT IF YOUR WORKING TAPE BECOMES DAMAGED OR WORN OUT, YOU CAN MAKE ANOTHER COPY. THE WORKING TAPE SHOULD BE A HIGH-QUALITY C-60 OR C-30 CASSETTE.

This page intentionally left blank.

B.1 CASSETTE INTERFACE INSTRUCTIONS

The MTS monitor program contains the software routines for loading and storing binary data using a tape cassette unit. The input routine, SERIN, resides at location 03AE and the output routine, SEROT, resides at 0371.

The MTS includes a tape cassette modem for recording programs on tape and loading programs from tape. Full instructions are included in Course 525A, and are briefly summarized here. Your ITS circuit board may include a duplicate modem, provided for use with an earlier version of the MTS, but this is not connected to the microcomputer input and output ports.

B.1.1 Instructions for Reading from Tape

(1) Find the program (or data) on the tape and listen for the solid tone which PRECEDES the program.

(2) Connect the cable to EAR on the tape unit and the connector labelled with an inward pointing arrow on the MTS.

(3) Set the AUTO/STEP toggle switch to AUTO.

(4) Enter the beginning load address as the memory address e.g., if the beginning address is 8100, press ADDR, then 8100, then MEM.

(5) Enter the start address of SERIN into the Program Counter: ADDR 03AE.

(6) With about 80% of full volume, press the PLAY button on the tape unit. Wait until the DATA LED emits a constant glow.

(7) Press RUN

The MTS displays will blank out and the DATA LED will flicker as data are received.

(8) A display of 03CF in the left MTS display indicates a successful load.

(9) A display of Err indicates an unsuccessful load, so repeat the procedure.

(10) Check the LRC by REG A. Register A should contain 00.

B.1.2 Instructions for Writing to Tape

(1) Advance the tape to the point where you want the program (data) stored. If using the beginning of the tape, make sure you have advanced the tape beyond the leader.

(2) Connect the cable to MIC or AUX on the tape unit and to the connector labelled with an outward arrow on the MTS.

(3) Set the AUTO/STEP toggle switch to AUTO.

(4) Enter the starting address of the program to be recorded as the memory address: ADDR 8100 MEM.

(5) Enter the stopping address (the next address after the end of the program) as a breakpoint: ADDR 82A0 BRK.

(6) Enter the start address of SEROT into the program counter: ADDR 0371.

(7) Press the RECORD and PLAY buttons on the tape unit simultaneously. Wait 5-10 seconds.

(8) Press RUN.

The MTS displays will blank out.

(9) A display of 0382 indicates a successful transfer.

CASSETTE LIBRARY PROGRAM DIRECTORY

Side 1 of Tape Cassette

1. Display Keyboard Input
   Figure 2-3

2. Time of Day
   Figure 3-17

3. Pulse Width Modulation
   Figure 4-7

4. Recorded Music Player - Figures 4-5
   Figures 4-15, 4-16, 4-17, and 4-19

5. Function Generators
   Figures 4-44, 4-45, and 4-46

6. A/D Input with FILTR
   Figure 5-30

7. Thermometer with Data Log
   Figures 5-38, 5-40, and 5-45

Side 2 of Tape Cassette

8. Thermostat with Deadband
   Figure 6-6

9. PWM Voltage Control
   Figures 6-10, 6-18, 6-28, 6-34, and 6-37

10. Motor Speed Control
    Figure 7-21

11. PONG
    Appendix B, Figure B-4

12. RS232C Message Handler
    Appendix C

Program Cassette Directory
Figure B-1

B.2 PROGRAM CASSETTE LIBRARY

Your program cassette library is a cassette tape that contains solutions for the longer 535 course exercises. The cassette is included only to free the student from the time-consuming key-in procedure. It is recommended that you use the cassette only after you have attempted your own solutions to the exercises. The tape also contains a game program and a communications program.

The listings of the programs are available in the appropriate course section. These are identified in the following pages, and a list of library programs is shown in Figure B-1.

B.2.1 Display Keyboard Input: Figure 2-3

This program provides a trivial test to determine whether the ITS and MTS are properly connected. It also gives the user experience in reading a tape.

Play the tape until a steady tone is heard.

When you hear the tone, promptly stop the tape and do the following:

Press RESET

Press ADDR 8200 MEM

Press ADDR 03AE

Connect the tape player earphone output to the connector labelled with an inward pointing arrow at the top right of the MTS circuit board.

Start the tape and observe a steady light in the LED labelled DATA, near the input connector. Immediately press RUN.

When the computer starts to receive data, the AUDIO and DATA lights will flicker. This will continue while the program is being loaded. At the end of the program these lights will become steady, and then the display will show 03CF C5.

Stop the tape.

Check that the program has been read correctly by pressing REG A. The display will show 03CF A-00. This implies that no errors have been detected.

To run the program, press RESET, RUN. The LED's on the ITS circuit board will all be lighted. Pressing any key on the MTS will extinguish one of the LED's while the key is held down.

B.2.2 Time of Day Program - Figure 3-17

This program uses the system's crystal clock and interval timer 0 to keep the time of day.

To load the program:

Set the red toggle switch to AUTO

Press RESET

Press ADDR 8200 MEM

Press ADDR 03AE

When the steady tone is heard, stop the tape, connect the input, and start playing the tape. Observe the DATA light and press RUN.

When the program has been loaded, check for errors by pressing REG A. The display should show 03CF A-00.

Before running the program, press RESET. Enter the time of day as follows:

Press REG H and enter hours.

Press NEXT and enter minutes into register L.

Press RUN. The display will show hours, minutes, and seconds.

The following programs are loaded by the same procedure as described previously.

B.2.3 Pulse Width Modulation - Figure 4-7

This program is described in Section 4.2 of Course 536. The recorded program is identical to that given in Figure 4-7. To use the program, make connections as shown in Figure 4-2. Then press RUN and observe the voltmeter output, which should show about 2.5 volts. Enter any two digit decimal number followed by NEXT, and observe that the output is proportional to the decimal value entered. The program starts at address 8200.

B.2.4 Recorded Music Player - Figures 4-15, 4-16, 4-17, and 4-19

This program plays a tune from memory. It is described in Section 4.3.3 The program is given in Figure 4-15, with a patch for a visual display from Figure 4-19. The table of timer intervals for the chromatic scale and two American folk tunes are given in Figures 4-16 and 4-17.

To use the program, press RESET, RUN. To play the first tune, or to repeat a tune, press NEXT. To start a different tune, enter its address and press NEXT. Two tunes are included in the tape, at addresses 8300 and 8330.

For students who want to develop other music programs, it is suggested that the interrupt service and timer table from this program be used, and other program segments be developed in the 8000 to 81FF section of memory.

B.2.5 Function Generator - Figures 4-44, 4-45, and 4-46

This program generates triangular and exponential waveforms, outputt~~t~~ing the signal through the Digital to Analog converter. Its development is described in Section 4.7.

The taped program is shown in Figures 4-44, 4-45, and 4-46. This is the final program including both waveforms.

Observe the output with a voltmeter at ANALOG OUT. To operate the program, press RESET, RUN. The exponential waveform will appear. Press REG to obtain a triangular waveform, or MEM to obtain the exponential. Variable data may be entered as described in Section 4.7.2.8 and Section 4.7.3.7.

Load the program at address 8100.

B.2.6 Analog to Digital Input with Digital Noise Filter - Figure 5-30

This program measures an input voltage and uses a digital filter to reduce noise present at the input. It is described in Sections 5.4 and 5.5

Connect a voltage source (not exceeding 2.5 volts) to ANALOG IN. Be careful that the input is positive with respect to ground -- negative inputs will damage the A/D converter. The ANALOG IN pot should be set fully to the left. The program will display the filtered voltage at the left and the input voltage next to it. Both displays are in hexadecimal. Adjust the ANALOG IN pot if necessary. Load the program at 8200.

B.2.7 Thermometer with Data Log - Figures 5-38, 5-40, and 5-45

This program measures the input voltage from a thermistor, applies the noise filter, converts the voltage to decimal degrees Celsius, and displays the temperature. It also records the filtered voltage at preset intervals in a data log, and permits subsequent review of the temperature history.

The voltage to temperature conversion is described in Section 5.6 The data log and review are described in Section 5.6.7.

Operate the program by pressing RESET, RUN. Then enter a timing interval in hexadecimal seconds, followed by RUN. This will display and record the temperature.

To review the temperatures one at a time, press RESET, RUN, NEXT. Each time you press NEXT, the next temperature will be displayed. To play the temperatures back continuously, press RESET and RUN. Then enter a timing interval and press STEP.

The program is loaded at address 8100.

B.2.8 Thermostat with Deadband - Figure 6-6

This program controls a heater to maintain the thermistor temperature within set limits. It is described in Section 6.1.2. Make the connections shown in Figure 6-1.

To run the program, press RESET, RUN. Enter a temperature limit followed by NEXT. This limit is expressed as decimal degrees and tenths, so to make the limit 25.0 degrees enter 250 NEXT.

The heater will be turned on if the measured temperature is below the limit. It will be turned off when the temperature rises two degrees above the limit.

Load the program at address 8200.

B.2.9 Closed Loop Voltage Control with Proportional Plus Integral Feedback - Figures 6-10, 6-18, 6-28, 6-34, and 6-37

This program controls an output voltage by pulse width modulation, and adjusts the output with proportional plus integral feedback.

Connections for testing the program are shown in Figure 6-10.

Program development and operation are described in Sections 6.2 and 6.3. The recorded program is the final program developed in Chapter 6 and includes segments from various figures listed above. The complete program listing is given in this appendix.

All of the experiments described in Sections 6.2.2.8, 6.2.5, and 6.3.4 can be performed with this program. Some results will be slightly different than shown in the text, because of the revised subroutines.

The program is to be loaded at address 8100.

B.2.10 Motor Speed Control - Figure 7-21

This program controls the speed of a motor with proportional plus integral closed loop control, sensing motor speed with a rotating disc and an optional coupler.

Electrical connections are shown in Figure 7-5.

All of Chapter 7 is devoted to development of this program. Operating instructions are given in Section 7.3.6. The program is shown in Figure 7-21.

Load the program at address 8100.

B.2.11 PONG

This is a game for two players. Operating instructions and program listing are shown in this appendix. Load the program at address 8200.

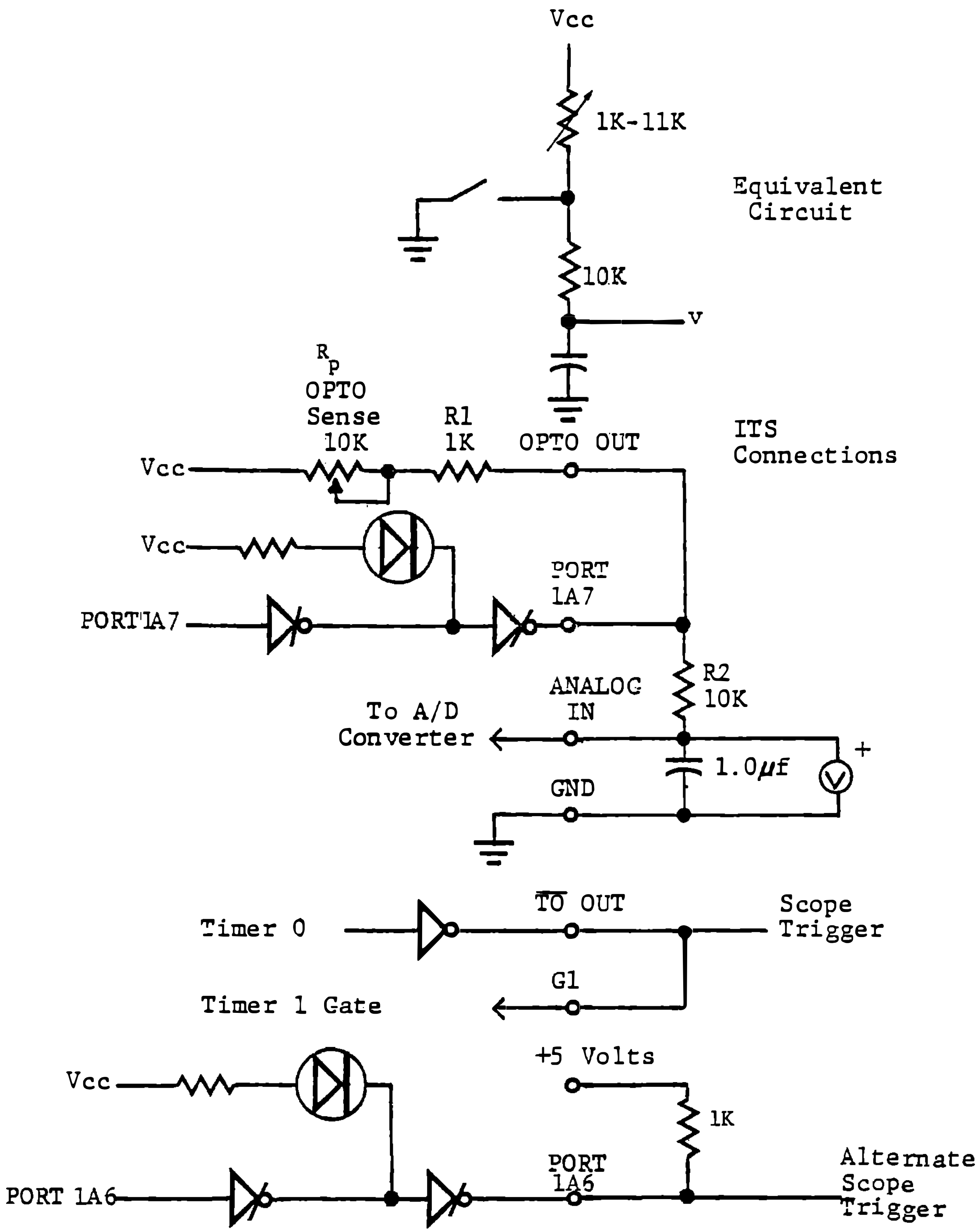
B.2.12 RS232C Message Handler - Appendix C

This program and required hardware are described in Appendix C. Load the program at address 8200.

Program Segments for Closed Loop Voltage Control

6-10	Connections
6-18a	Initialization
6-18b	Interrupt Service
6-18c	VOLTM
6-18d	FILTR
6-18e	FILTR and SHFTN
6-18f	Set Initial Values
6-28a	Main Loop
6-28b	KYTIM
6-28c	KYTIM
6-28d	KYTIM
6-34	LDT1, Version 2
6-28f	CLOSL, Version 2
6-37a	INTEG, Version 3
6-28h	PROPG
6-37b	ADTOV

Figure B-2



Connections for PWM Experiment

(Figure 6-10)

Figure B-2a

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR	CODE				
8 20 0	3E	MVI	A, 82	Program 8255 #1
1	82			Port 1B in for A/D
2	D3	OUT	CNT1	
3	07			
4	3E	MVI	A, 92	Program 8255 #2
5	92			
6	D3	OUT	CNT2	
7	0F			
8	3E	MVI	A, 34	Program Timer 0
9	34			Both bytes
A	D3	OUT	TIMCT	Mode 2
B	17			Binary
C	3E	MVI	A, 7A	Program Timer 1
D	7A			Both bytes
E	D3	OUT	TIMCT	Mode 3
F	17			Binary
8 21 0	3E	MVI	A, 94	Program Timer 2
1	94			Low byte
2	D3	OUT	TIMCT	Mode 2
3	17			Binary
4	3E	MVI	A, 08	Load Timer 2
5	08			for ÷ 8
6	D3	OUT	TIM2	
7	16			
8	21	LXI	H, 83A0	Initialize memory
9	A0			for FILTR
A	83			
B	36	MVI	M, 04	n = 4
C	04			
D	AF	XRA	A	Clear 2^n E_i
E	23	INX	H	Address 83A1
F	77	MOV	M, A	
8 22 0	23	INX	H	Address 83A2
1	77	MOV	M, A	
2	3C	INR	A	Set Port 1C0
3	D3	OUT	PORT1C	for automatic
4	06			A/D input
5	C3	JMP	82B0	Jump past
6	B0			interrupt service
7	82			and FILTR
8				Figure 6-18a

PWM VOLTAGE CONTROL

ADDR	CODE			Comment
8 0			INTERRUPT	SERVICE
1				
2				
3				
4				
5				
6				
7				
822 8	F5		PUSH PSW	RST5 - Timer 0
9	3E		MVI A,80	Set Port 1A7 high
A	80			to start charging
B	D3		OUT PORT 1A	All other bits low
C	04			
D	C3		JMP 8240	Jump past RST6
E	40			
F	82			
8 23 0	F5		PUSH PSW	RST6 - Timer 1
1	3E		MVI A,00	Set Port 1A7 low
2	00			to start discharge
3	D3		OUT PORT 1A	(all other bits also)
4	04			Note identical timing
5	3E		MVI A,03	as in RST5
6	03			
7	D3		OUT CNT2	Reenable and clear
8	0F			Timer 1 interrupt
9	F1		POP PSW	Exit
A	FB		EI	
B	C9		RET	
C	00		NOP	
D	00		NOP	
E	00		NOP	
F	00		NOP	
8 24 0	3E		MVI A,01	Reenable and clear
1	01			Timer 0 interrupt
2	D3		OUT CNT2	
3	0F			
4	F1		POP PSW	Exit
5	FB		EI	
6	C9		RET	
7				
8				Figure 6-18b

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

ADDR	CODE		INSTRUCTION	COMMENT
8250	E5		PUSH H	Save (HL)
1	21		LXI H, 8000	Address storage
2	00	*		location for
3	80	*		data log address
4	34		INR M	Increment address
5	C2		JNZ 8259	but not beyond FF
6	59			
7	82			
8	35		DCR M	Restore FF
8259	6E		MOV L, M	(HL) ← data log address
A	36		MVI M, FF	Store FF in case
B	FF			A/D not ready
C	DB		IN PORT 2B	Read interrupt
D	0D			status byte
E	E6		ANI 08	Mask for
F	08			A/D comparator
8260	CA		JZ 826D	Jump if
1	6D			A/D not ready
2	82			
3	DB		IN PORT 1B	Read A/D voltage
4	05			
5	32		STA 83AA	Store voltage
6	AA			in fixed location
7	83			and log in
8	77		MOV M, A	variable location
9	3E		MVI A, 06	Reset A/D counter
A	06			Leave interrupt
B	D3		OUT CNT 2	disabled
C	0F			
826D	7E		MOV A, M	(A) ← voltage
E	E1		POP H	Restore (HL)
F	C9		RET	
8 0				
1		*	(8000) CONTAINS	
2			DATA LOG ADDRESS	
3			RETURN (A) = VOLTAGE	
4			ZERO FLAG NOT SET	
5			IF A/D NOT READY	
6			(A) = FF ZERO FLAG SET	
7				
8				Figure 6-18c

SUBROUTINE FILTR

INTEGRATED COMPUTER SYSTEMS — MICROCOMPUTER TRAINING SYSTEM — CODING SHEET

ADDR	CODE		
8270	D5	PUSH D	Save (DE)
1	C5	PUSH B	Save (BC)
2	46	MOV B, M	(B) ← n
3	48	MOV C, B	(C) ← n
4	23	INX H	Address $2^n E_{i-1}$
5	E5	PUSH H	Save address
6	5E	MOV E, M	
7	23	INX H	(DE) ← $2^n E_{i-1}$
8	56	MOV D, M	
9	6B	MOV L, E	(HL) ← $2^n E_{i-1}$
A	62	MOV H, D	
827B	29	DAD H	
C	0D	DCR C	
D	C2	JNZ 827B	(HL) ← $2^{2n} E_{i-1}$
E	7B		
F	82		
8280	4F	MOV C, A	(C) ← V_i
1	7D	MOV A, L	
2	93	SUB E	(DE) ← (HL) − (DE)
3	5F	MOV E, A	$= 2^n(2^n-1)E_{i-1}$
4	7C	MOV A, H	
5	9A	SBB D	
6	57	MOV D, A	
7	69	MOV L, C	
8	26	MVI H, 00	(HL) ← V_i
9	00		
A	CD	CALL SHFTN	Divide by 2^n
B	A0		(DE) ← $(2^n-1)E_{i-1}$
C	82		
D	EB	XCHG	
E	19	DAD D	(DE) ← $V_i + (2^n-1)E_{i-1}$
F	EB	XCHG	$= 2^n E_i$
830		CONTINUED	AT 8290
1			
2		ENTER (A) =	V_i (new value)
3		((HL)) =	n (filter constant)
4		((HL) + 1)	$2^n E_{i-1}$
5		((HL) + 2)	
6		RETURN	
7		((HL) + 3) =	(A) = (H) = E_i
8			(L) = V_i

Figure 6-18d

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		Instruction	Operand	Comment
829	0	E	3	XTHL		(HL) ← address $2^n E_i$
	1	7	3	MOV	M, E	Store $2^n E_i$
	2	2	3	INX	H	
	3	7	2	MOV	M, D	
	4	2	3	INX	H	Address E_i
	5	1	B	DCX	D	To round up only if
	6	C	D	CALL	SHFTN	fractional part > 1/2
	7	A	0			
	8	8	2			
	9	7	7	MOV	M, A	Store E_i
	A	E	1	POP	H	(L) ← V_i
	B	6	7	MOV	H, A	(H) ← E_i
	C	C	1	POP	B	Restore B, C, D, E
	D	D	1	POP	D	
	E	C	9	RET		Exit
	F	0	0	NOP		
82A	0	4	8	MOV	C, B	SHFTN (c) ← n
	1	A	F	XRA	A	Loop – clear carry
	2	7	A	MOV	A, D	
	3	1	F	RAR		Shift (DE) right
	4	5	7	MOV	D, A	to divide by 2
	5	7	B	MOV	A, E	
	6	1	F	RAR		
	7	5	F	MOV	E, A	
	8	0	D	DCR	C	Loop n times
	9	C	2	JNZ	82A1	to divide by 2^n
	A	A	1			
	B	8	2			
	C	D	0	RNC		Exit if LSB = 0
	D	1	3	INX	D	Else round up
	E	7	B	MOV	A, E	(A) ← less
	F	C	9	RET		significant byte
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					Figure 6-18e

CODING SHEET

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM

PWM VOLTAGE - SET INITIAL VALUES

ADDR	CODE	INSTRUCTION	COMMENTS
8 2B 0	11	LXI D,0400	Set total period
1	00		to 500 msec
2	04		
3	CD	CALL RUN	Run key function of
4	38		KYTIM "G", loads timer 0
5	81		for total period
6	11	LXI D,00C8	Set voltage
7	C8		to 2.00 volts
8	00		
9	CD	CALL NEXT	Next key function of
A	20		KYTIM "U", calculates the
B	81		width, loads timer 1
C	EF	RST 5	
D	F7	RST 6	
E	C3	JMP 82D0	
F	D0		
8 2C 0	82		
1			
2			
3			
4			
5			
6			
7			
8			
9			
A			
B			
C			
D			
E			
F			
8 0			
1			
2			
3			
4			
5			
6			
7			
8			

Figure 6-18f

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		LABEL	INSTRUCTION	OPERAND	COMMENT
8 2D	0	D	B		IN	PORTOA	Test keyboard
	1	0	0				(FF if no key)
	2	3	C		INR	A	
	3	C	4		CNZ	KYTIM	Call for keyboard
	4	0	0				entry and process
	5	8	1				
	6	C	D		CALL	VOLTM	Measure voltage
	7	5	0				(A)← A/D voltage
	8	8	2				(FF if not ready)
	9	F	5		PUSH	PSW	Save input
	A	C	D		CALL	CLOSL	Calculate error
	B	8	0				Adjust pulse width
	C	8	1				
	D	C	D		CALL	DWORD	Display error
	E	D	1				and voltage request
	F	0	2				
8 2E	0	F	1		POP	PSW	(A)← input voltage
	1	2	1		LXI	H, 83A0	Memory address
	2	A	0				for FILTR
	3	8	3				
	4	C	D		CALL	FILTR	
	5	7	0				(L)← raw voltage
	6	8	2				(H)← filter voltage
	7	1	1		LXI	D, 83FF	
	8	F	F				
	9	8	3				
	A	C	D		CALL	DWD2	
	B	D	4				
	C	0	2				
	D	C	3		JMP	82D0	
	E	D	0				
	F	8	3				
8	0						
	1			IDENTICAL			TO
	2			FIGURE		6-23a	
	3						
	4						
	5						
	6						
	7						
	8						Figure 6-28a

PWM - SUBROUTINE KYTIM

ADDR	CODE		Instruction	Comment
8 10 0	0	0	NOP	
1	C	D	CALL ENTWD	(A) ← Command
2	4	6		(HL) ← total period
3	0	3		or voltage
4	E	B	XCHG	(DE) ← data
5	2	1	LXI H, 8108	Address dispatch
6	0	8		table -10H
7	8	1		
8	8	5	ADD L	Add command
9	6	F	MOV L, A	
A	6	E	MOV L, M	(5T) ← dispatch address
B	E	5	PUSH H	
C	F	3	DI	
D	D	B	IN PORT 1A	
E	0	4		Set Port 1A6 high
F	F	6	ORI 40	for scope trigger
8 11 0	4	0		without changing
1	F	B	EI	Port 1A7
2	D	3	OUT PORT 1A	
3	0	4		
4	A	F	XRA A	Clear A and CY
5	0	0	NOP	
6	0	0	NOP	
7	C	9	RET	To dispatch address
8	4	0	MEM	
9	5	0	REG	
A	1	7	ADDR	Trigger scope
B	2	1	STEP	Store desired voltage
C	3	8	RUN	Set total period
D	2	0	NEXT	Store voltage, set width
E	4	5	BRK	
F	4	7	CLR	
8 0				
1				
2				
3				
4				
5				
6				
7				
8				

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

Figure 6-28b

ADDR	CODE	INSTRUCTION	COMMENT
8 12 0	37	STC	NEXT (mark with CY)
812 1	21	LXI H, 8000	STEP (CY clear)
2	00		Address data log
3	80		and load with its
4	75	MOV M, L	own address
5	EB	XCHG	
6	22	SHLD 83A6	
7	A6		
8	83		
9	D0	RNC	Exit if STEP
A	11	LXI D, FFEE	To subtract 12H
B	EE		(0.18 volts) from
C	FF		desired voltage
D	19	DAD D	(HL) ← $v - V_g$
E	29	DAD H	(HL) ← pulse width
F	00	NOP	
8 13 0	22	SHLD 83A4	Store calculated
1	A4		pulse width
2	83		as integral
3	CD	CALL LDT1	Load Timer 1
4	60		with calculated
5	81		pulse width
6	C9	RET	
7	00	NOP	
813 8	7B	MOV A, E	RUN
9	D3	OUT TIM0	Load Timer 0
A	14		for total period
B	7A	MOV A, D	
C	D3	OUT TIM0	
D	14		
E	C9	RET	
F	00	NOP	
8 0			
1		IDENTICAL	TO
2		FIGURE 6-18i	
3			
4			
5			
6			
7			
8			Figure 6-28c

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

KYTIM - MEM, BRK, CLR, REG

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

ADDR		CODE		Instruction	Operand	Comments
8 14	0	E	B	XCHG		MEM
	1	2	2	SHLD	83A8	Store gains for CLOSL
	2	A	8			
	3	8	3			
	4	C	9	RET		
814	5	3	E	MVI	A, C9	BRK - Open loop
	6	C	9			Enter RET in INTEG
814	7	3	2	STA	81C0	CLR - Close loop
	8	C	0			Enter NOP in INTEG
	9	8	1			
	A	2	1	LXI	H, 8000	Start a new log
	B	0	0			
	C	8	0			
	D	7	5	MOV	M, L	
	E	C	9	RET		
	F	0	0	NOP		
8 15	0	D	3	OUT	CNT2	REG - Force output low
	1	0	F			Disable Timer 0 interrupt
	2	C	D	CALL	RSTDY	Monitor function
	3	4	4			Delay ~30 milliseconds
	4	0	2			plus interrupt time
	5	2	1	LXI	H, 8000	Start a new log
	6	0	0			
	7	8	0			
	8	7	5	MOV	M, L	
	9	7	6	HLT		Wait for Timer 1
	A	3	E	MVI	A, 01	Enable and clear
	B	0	1			Timer 0 interrupt
	C	D	3	OUT	CNT2	
	D	0	F			
	E	C	9	RET		
	F	0	0	NOP		
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					Figure 6-28d

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR	CODE	INSTRUCTION	COMMENT
816 0	2B	DCX H	Test calculated
1	7C	MOV A, H	pulse width for
2	B7	ORA A	zero or negative
3	23	INX H	
4	3E	MVI A, 02	Load (A) to disable
5	02		Timer 0 interrupt
6	FA	JMIN 8171	and jump if
7	71		pulse width ≤ 0
8	81		
9	7D	MOV A, L	Pulse width > 0
A	D3	OUT TIM1	Load Timer 1 for
B	15		new pulse width
C	7C	MOV A, H	
D	D3	OUT TIM1	
E	15		
F	3E	MVI A, 03	Load (A) to enable
817 0	03		Timer 0 interrupt
817 1	D3	OUT PORT2C	Enable or disable
2	0E		Timer 0 interrupt
3	C9	RET	and enable
4			Timer 1 interrupt
5			
6			
7		REVISED LDT1	
8		FOR FULL SCALE CONTROL	
9			
A		ENTER WITH	
B		(HL) = CALCULATED	
C		PULSE WIDTH	
D		IF (HL) > ZERO	
E		LOAD TIMER 1 AND	
F		ENABLE BOTH INTERRUPTS	
8 0			
1		IF (HL) ≤ ZERO	
2		DISABLE TIMER 0 TO	
3		INHIBIT CHARGING	
4			
5			
6			
7			
8			Figure 6-34

PWM - SUBROUTINE CLOSL - VERSION 2

ADDR		CODE													
8 18	0	2	A	L	H	L	D		8	3	A	6		CLOSL
	1	A	6											(L) ← desired voltage;
	2	8	3											At entry (A) = measured
	3	9	5	S	U	B		L						(A) ← −Error
	4	6	7	M	O	V		H	,	A				(H) ← −Error
	5	C	8	R	Z									Exit if error = 0
	6	E	5	P	U	S	H		H					Save display data
	7	2	F	C	M	A								
	8	4	F	M	O	V		C	,	A				
	9	3	F	C	M	C								(BC) ← +Error
	A	9	F	S	B	B		A						as two bytes
	B	4	7	M	O	V		B	,	A				
	C	0	3	I	N	X		B						
	D	2	A	L	H	L	D		8	3	A	4		(HL) ← old integral
	E	A	4											
	F	8	3											
8 19	0	C	D	C	A	L	L		I	N	T	E	G	Calculate and
	1	C	0											store new integral
	2	8	1											(HL) ← (HL) + (BC)/2ⁿ
	3	C	D	C	A	L	L		P	R	O	P	G	add proportional term
	4	E	0											(HL) ← (HL) + K*(BC)
	5	8	1											
	6	C	D	C	A	L	L		L	D	T	1		Load Timer 1 with
	7	6	0											new pulse width
	8	8	1											
	9	E	1	P	O	P		H						Recover display data
	A	C	9	R	E	T								
	B													
	C													
	D													
	E													
	F													
8	0													
	1													
	2													
	3													
	4													
	5													
	6													
	7													
	8													Figure 6-28f

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR	CODE	Instruction	Operand	Comments
8 1C 0	00	NOP/RET		
1	EB	XCHG		(DE)← old integral
2	69	MOV	L, C	(L)← low byte error
3	3A	LDA	83A8	(A)← n for
4	A8			Error/2^n
5	83			
6	67	MOV	H, A	(H)← n
81C 7	25	DCR	H	Decrement count
8	FA	JMIN	81D5	Exit loop after 0
9	D5			
A	81			
B	78	MOV	A, B	Set CY if error
C	1F	RAR		negative
D	7D	MOV	A, L	Shift low byte of
E	1F	RAR		error right
F	6F	MOV	L, A	
8 1D 0	B7	ORA	A	Test for error = 0
1	C2	JNZ	81C7	Loop unless 0
2	C7			
3	81			If shifted error = 0
4	2C	INR	L	Make error = 01
81D 5	60	MOV	H, B	(H)← high byte error
6	CD	CALL	ADTOV	Add (HL)←(HL)+(DE)
7	F0			Return MINUS if
8	81			arithmetic overflow
9	F8	RMINUS		
A	22	SHLD	83A4	Store new integral
B	A4			
C	83			
D	C9	RET		
E	00	NOP		
F	00	NOP		
8 0				
1		ENTER	WITH	
2		(BC)	= +	ERROR (B=00/FF)
3		(HL)	= OL	D INTEGRAL
4		RETURN		
5		(BC)	PRES	ERVED
6		(DE)	= OL	D INTEGRAL
7		(HL)	= NE	W INTEGRAL
8				

Figure 6-37a

PWM - SUBROUTINE PROPG

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

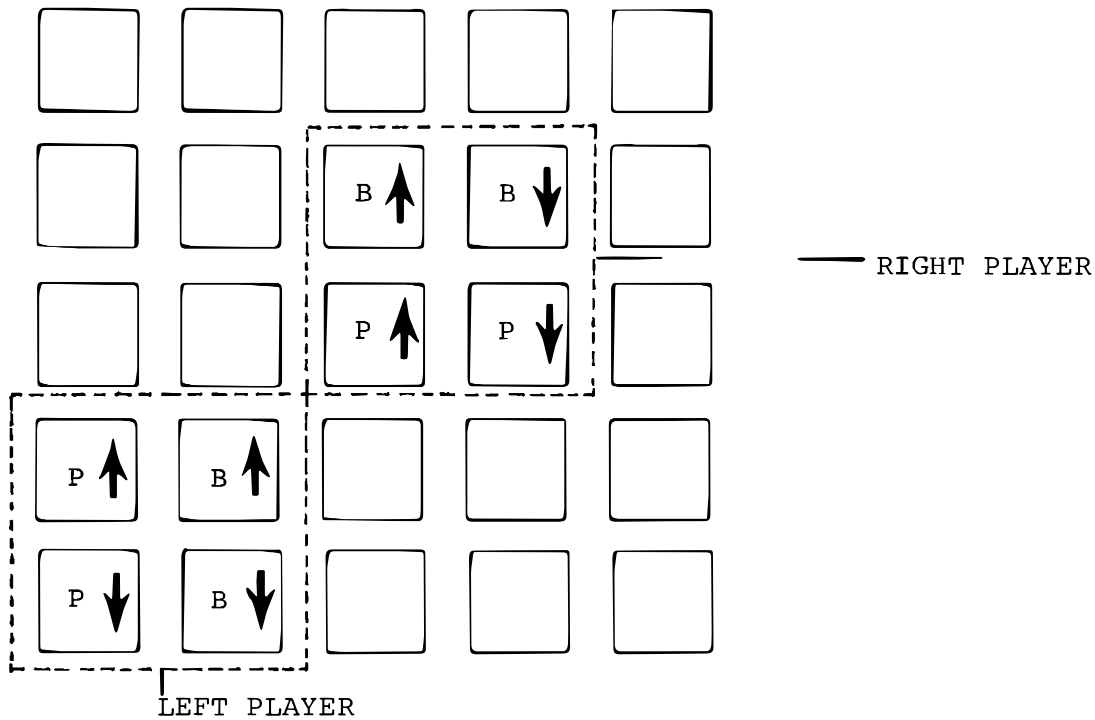
ADDR		CODE			
8 1E	0	3A		LDA 83A9	(A) ← K
	1	A9			for K error
	2	83			
81E	3	3D		DCR A	Decrement K
	4	F8		RMINUS	Exit after zero
	5	09		DAD B	Add error into
	6	C3		JMP 81E3	pulse width
	7	E3			from INTEG and
	8	81			loop until done
	9				
	A				
	B				
	C			ENTER WITH	
	D			(BC) = + ERROR	
	E			(HL) = PULSE WIDTH	
	F			UPDATED BY INTEG	
8	0				
	1			RETURN	
	2			(BC) PRESERVED	
	3			(DE) PRESERVED	
	4			(HL) = NEW PULSE WIDTH	
	5			= INTEG + K ERROR	
	6				
	7				
	8				
	9				
	A				
	B				
	C				
	D				
	E				
	F				
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				Figure 6-28h

Figure 6-37b

STORE INTO MEMORY STARTING AT 8200.

DEPRESS. RST RUN

CONTROL KEYS ARE AS FOLLOWS:



P↑ MOVE PADDLE UP

P↓ MOVE PADDLE DOWN

B↑ MOVE BALL UP

B↓ MOVE BALL DOWN

LOCATION 830C CONTROLLS BALL SPEED.

OC = NORMAL SPEED; 08 = Fast.

FIRST PLAYER TO SCORE 15 POINTS WINS.

TO START NEW GAME, DEPRESS ANY KEY.

Pong Game

Figure B-4

ADDR		CODE				
820	0	2	1	START	LXI H, LEFT	Clear score - Load
	1	F	E			HL with score location)
	2	8	2			
	3	A	F		XRA A	
	4	4	F		MOV C, A	Clear ball control
	5	7	7		MOV M, A	
	6	2	3		INX H	
	7	7	7		MOV M, A	
820	8	3	E	INIT	MVI A, 40H	Set paddle positions
	9	4	0			to middle
	A	3	2		STA D1	(1=top, 40=middle
	B	F	8			8=bottom)
	C	8	3			
	D	3	2		STA D8	
	E	F	F			
	F	8	3			
821	0	3	E		MVI A, 8FH	Set all SCAN lines
	1	8	F			on keyboard to 0
	2	D	3		OUT PORT 0C	
	3	0	2			
	4	0	6		MVI B, 40H	Set initial ball
	5	4	0			position to middle
	6	2	1		LXI H, D4	Set ball position in HL
	7	F	B			
	8	8	3			
821	9	7	8	LOOP	MOV A, B	Load ball into display
	A	B	6		ORA M	
	B	7	7		MOV M, A	
	C	C	D		CALL DELAY	Delay for 1 frame
	D	0	B			
	E	8	3			
	F	D	B		IN PORT 0A	Get left player
822	0	0	0			paddle info
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					Figure B-4a

CODING SHEET
MICROCOMPUTER TRAINING SYSTEM
INTEGRATED COMPUTER SYSTEMS

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

PONG - TEST FOR MOVE PADDLE

ADDR	CODE	LABEL	MNEMONIC	OPERAND	COMMENT
8 0					
822 1	E6		ANI	11H	Bit 4=up, Bit 0=down
2	11				
3	EA		JPE	NCHL	If both the same,
4	2F				ignore (no change
5	82				left)
6	E6		ANI	01	See if down
7	01				
8	11		LXI	D, D1	Load paddle location
9	F8				
A	83				
B	CD		CALL	CHP	Change paddle position
C	00				
D	83				
E	12		STAX	D	Save
822 F	DB	NCHL	IN	PORTOA	Get right player paddle info
823 0	00				
1	E6		ANI	0CH	Bit 2=up, Bit 3=down
2	0C				
3	EA		JPE	NCHR	If both the same,
4	3F				ignore (no change
5	82				right)
6	E6		ANI	08H	See if down
7	08				
8	11		LXI	D, D8	Load paddle position
9	FF				
A	83				
B	CD		CALL	CHP	Change paddle position
C	00				
D	83				
E	12		STAX	D	Save
823 F	7D	NCHR	MOV	A, L	See if end of shield
8 0					
1					
2					
3					
4					
5					
6					
7					
8					

Figure B-4b

ADDR	CODE		LABEL	OPCODE	OPERAND	COMMENTS
824 0	F	E		CPI	F8H	If left of net
1	F	8				
2	C	A		JZ	LBC	then → LBC;
3	7	C				
4	8	2				
5	F	E		CPI	FFH	Else if right side of net
6	F	F				
7	C	A		JZ	RBC	then → RBC;
8	8	6				
9	8	2				
A	A	F		XRA	A	Else clear old position
B	7	7		MOV	M, A	
C	B	9		CMP	C	Who has the ball?
D	C	A		JZ	NBC	No one → no ball change
E	7	8				
F	8	2				
825 0	0	D		DCR	C	Count (less) frame
1	7	9		MOV	A, C	
2	3	F		CMC		
3	1	7		RAL		
4	D	A		JC	LFTC	If $C_y = 1$ then → LFTC else → RTC
5	6	3				
6	8	2				
7	D	B	RTC	IN	PORTOA	
8	0	0				
9	E	6		ANI	C0H	Bit 7 = down, Bit 6 = up
A	C	0				
B	E	A		JPE	NCH	If $A_7 = A_6$ then → no change
C	7	1				
D	8	2				
E	E	6		ANI	80H	If down then
825 F	8	0				
8 0						
1						
2						
3						
4						
5						
6						
7						
8						Figure B-4c

MICROCOMPUTER TRAINING SYSTEM CODING SHEET — INTEGRATED COMPUTER SYSTEMS

PONG - MOVE BALL (continued)

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR	CODE	LABEL	MNEMONIC	OPERAND	COMMENTS
8260	C3		JMP	SAVB	→ go change
1	6C				
2	82				
8263	DB	LFTC	IN	PORT0A	Check for left player
4	00				change
5	E6		ANI	22H	Bit5=up; Bit 1=down
6	22				
7	EA		JPE	NCH	If $A_5 = A_1$
8	71				then → no change
9	82				
A	E6		ANI	02H	If down
B	02				then
C	78	SAVB	MOV	A,B	Load old ball position
D	CD		CALL	CHB	Change ball position
E	01				
F	83				
8270	47		MOV	B,A	Save in ball's reg.
8271	79	NCH	MOV	A,C	Get change reg.
2	E6		ANI	0FH	If count ≠ 0
3	0F				then → no ball change
4	C2		JNZ	NBC	
5	78				
6	82				
7	4F		MOV	C,A	Else set count = 0
8278	2B	NBC	DCX	H	Change ball position -
9	C3		JMP	LOOP	either up or down
A	19				
B	82				
C	7E	LBC	MOV	A,M	If paddle position ≠
D	B8		CMP	B	ball position
E	C2		JNZ	LLOSE	then → left lose
F	98				
8280	82				
1					
2					
3					
4					
5					
6					
7					
8					

Figure B-4d

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

ADDR		CODE	LABEL	MNEMONIC	OPERAND	COMMENT
8	0					
828	1	0E		MVI	C,83H	Else new ball counter
	2	83				
	3	C3		JMP	REV	Go reverse direction
	4	8D				
	5	82				
828	6	7E	RBC	MOV	A,M	If paddle position ≠
	7	B8		CMP	B	ball position
	8	C2		JNZ	RLOSE	then → right lose
	9	9E				
	A	82				
	B	0E		MVI	C,03H	Set new ball counter
	C	03				
828	D	3A	REV	LDA	NBC	Reverse ball direction
	E	78				by complementing
	F	82				INX H/DCX H, bit 3
829	0	EE		XRI	08H	via XOR (Classical
	1	08				Kludge)
	2	32		STA	NBC	
	3	78				
	4	82				
	5	C3		JMP	NBC	Go change ball position
	6	78				
	7	82				
829	8	21	LLOSE	LXI	H,LEFT	Add 1 to right's score
	9	FE				
	A	82				
	B	C3		JMP	SCOR	
	C	A1				
	D	82				
	E	21	RLOSE	LXI	H,RIGHT	Add 1 to left's score
	F	FF				
82A	0	82				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					

Figure B-4e

PONT - SCORE (continued)

ADDR		CODE				
8	0					
82A	1	FE	SCOR	MOV	A,M	Load score and
	2	C6		ADI	01H	add 1
	3	01				
	4	27		DAA		with decimal adjust
	5	77		MOV	M,A	Save
	6	FE		CPI	15	If (A)≠15
	7	15				
	8	C2		JNZ	NWIN	then → no winner yet
	9	D5				
	A	82				
	B	2C		INR	L	Else
	C	21		LXI	H,D8	If L+1=0
	D	FF				
	E	83				
	F	CA		JZ	RTL	then right lost
8 2B	0	B4				
	1	82				
	2	2E		MVI	L,FBH	else left lost
	3	FB				
82B	4	E5	RTL	PUSH	H	Save position
	5	CD		CALL	CLEAR	Clear display
	6	87				
	7	02				
	8	E1		POP	H	
	9	36		MVI	M,"E"	
	A	79				
	B	2B		DCX	H	
	C	36		MVI	M,"S"	
	D	6D				
	E	2B		DCX	H	
	F	36		MVI	M,"D"	
8 2C	0	3F				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					

Figure B-4f

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

PONG - START NEW GAME

ADDR		CODE	LABEL	MNEMONIC	OPERAND	COMMENTS
8	0					
82C	1	2B		DCX	H	
	2	36		MVI	M, "L"	
	3	38				
	4	3E		MVI	A, FFH	Hold display
	5	FE				
	6	CD		CALL	DIS	
	7	0D				
	8	83				
	9	DB	NKY	IN	PORT 0A	Wait for key to
	A	00				begin new game
	B	B7		ORA	A	
	C	EA		JPE	NKY	
	D	C9				
	E	82				
	F	CD		CALL	CLEAR	Clear display
82D	0	87				
	1	02				
	2	C3		JMP	STRT	Start new game
	3	00				
	4	82				
	5	CD	NWIN	CALL	CLEAR	
	6	87				
	7	02				
	8	21		LXI	H, LEFT	Load left losing score
	9	FE				
	A	82				
	B	11		LXI	D, D6	Display on right
	C	FD				
	D	83				
	E	7E		MOV	A, M	
	F	CD		CALL	DBY2	via monitor
82E	0	98				
82E	1	02				
	2					
	3					
	4					
	5					
	6					
	7					
	8					

Figure B-4g

PONG - START NEW GAME (continued)

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE	LABEL	MNEMONIC	OPERAND	COMMENTS
8	0					
	1					
82E	2	23		INX	H	Get right losing score
	3	7E		MOV	A, M	
	4	CD		CALL	DBY2	Display on left
	5	98				
	6	02				
	7	3E		MVI	A, 40H	Hold for a while
	8	40				
	9	CD		CALL	DIS	
	A	0D				
	B	83				
	C	CD		CALL	CLEAR	
	D	87				
	E	02				
	F	AF		XRA	A	
82F	0	4F		MOV	C, A	Clear ball counter
	1	C3		JMP	INIT	Go back
	2	08				
	3	82				
82F	4	FE	DOWN	CPI	08H	then do not change
	5	08				else clear carry
	6	C8		RZ		and change position
	7	B7		ORA	A	(A.. → A..-2)
	8	1F		RAR		
	9	1F		RAR		
	A	1F		RAR		
	B	C9		RET		
	C					
	D					
82F	E		LEFT			Left's misses (right's score)
82F	F		RIGHT			Right's misses (Left's score)
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					

Figure B-4h

PONG - SUBROUTINES

CODING SHEET · MICROCOMPUTER TRAINING SYSTEM · INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		LABEL	INSTRUCTION	COMMENTS
830	0	1	A	CHP	LDAX D	Change paddle
	1	C	A	CHB	JZ DOWN	Assume we have
	2	F	4			position for ball
	3	8	2			
	4	F	E		CPI 01H	If up and on top
	5	0	1			
	6	C	8		RZ	then do not change
	7	1	7		RAL	
	8	1	7		RAL	
	9	1	7		RAL	
	A	C	9		RET	
	B	3	E	DELAY	MVI A, 0CH	Delay for 1 frame
830	C	0	C	**		** BALL SPEED**
830	D	1	1	DIS	LXI D, 07B4	Main delay
	E	B	4			
	F	0	7			
831	0	1	D	DEL	DCR E	Delay by decrement
	1	C	2		JNZ DEL	
	2	1	0			
	3	8	3			
	4	1	5		DCR D	
	5	C	2		JNZ DEC	
	6	1	0			
	7	8	3			
	8	3	D		DCR A	
	9	C	2		JNZ DIS	
	A	0	D			
	B	8	3			
	C	C	9		RET	
	D					
	E					
	F			**	(CHANGE BALL SPEED	
3	0				BY CHANGING LOCATION	
	1				830C FROM 0C TO 08)	
	2					
	3					
	4					
	5					
	6					
	7					
	8					Figure B-4i

This page intentionally left blank.

MICROCOMPUTER INTERFACING WORKBOOK

APPENDIX C

RS 232c INTERFACE SYSTEM

This page intentionally left blank.

APPENDIX C

RS232C INTERFACE SYSTEM

The RS232C Interface System consists of the following I/O driver subroutines and Interface Training System board connections.

Software is used to convert parallel data, transmitted by the MTS, into a serial stream of bits consisting of one low start bit, eight bits of data, and one high stop bit. A Software delay subroutine is used to produce a baud rate of 300 bits/second or 30 characters/second. This delay program may be modified to allow transmission rates of up to 4800 baud (480 characters/second). The ITS hardware option and connections of Section C.1 are required to convert the digital data up to the standard +/- 12 volt EIA-RS232 connector in order to communicate with a terminal.

APPENDIX C

C.1 ICS RS232 SOFTWARE

The RS232 Software includes the following subroutines:

* IMSG - Input a line of text from the terminal, terminated by a Carraige Return, via ICHR.
  - Entry Point 8297

* ICHR - Input a character from the terminal into register E via Port 1A0.
  - Entry Point 82B0

* OCHR - Output a character from register E to the terminal via Port 1C1.
  - Entry Point 82D0.

* OMSG - Output a block of characters, terminated with a CNTL-C (03H), to the terminal via OCHR.
  - Entry Point 82F6.

Specifications, flowcharts and source code for the above programs are given in Section C.3.

C.2 REQUIRED CONNECTIONS FOR RS 232 INTERFACE

C.2.1 Parts List

QTY	Description
1	16-pin DIP plug
7	12" lengths of ©22 gauge wire jumpers
1	25 pin CANNON DB-25s-5 connector
1	1 K ohm resistor - 1/4 watt
5	spade clips (optional)
*	-12 volt power supply (most terminals)

C.2.2 Fabrication Procedure

DIP plug connector

1. Solder a jumper wire to pin 2 of DIP plug. (Fasten spade clip to other end of wire.)
2. Solder a jumper wire to pin 16 of DIP plug. (Fasten spade clip to other end of wire.)

CANNON connector

3. Connect pins 5, 6 and 8 together.
4. Solder one end of a jumper wire to this connection (pin 5, 6 or 8).
5. Solder other end of jumper to the 1K resistor (insulate solder joint with heat shrink tubing or elctrical tape).
6. Connect pins 1 and 7 together.
7. Solder a jumper wire to pin 1 (or 7). (Fasten spade clip to other end of jumper.)

8. Solder a jumper wire to pin 2. (Fasten spade clip to other end of jumper.)
9. Solder a jumper wire to pin 3. (Fasten a spade clip to other end of jumper.)

C.2.3 ITS Board Connections

10. CANNON pin 1 (or 7) to ITS ground (TIE Block GND or CASSETTE GND).
11. CANNON pin 2 to ITS terminal strip pin 4 (RS 232 REC).
12. CANNON pin 3 to terminal strip pins 2 and 3 (TTY SEND and TTY RET).
13. CANNON pin 5 (or 6 or 8) jumper with 1K resistor to TIE Block +12v

C.2.4 -12 Volt Power Supply

14. Connect the -12 volt power supply lead and its ground to the ITS TIE Block -12 and GND tie points, respectively.

* NOTE - While the RS 232 standards specify a high (or 1) signal level of +12 volts and a low (or 0) signal of -12 volts, some terminals (such as the Lear-Seigler ADM-3) will recognize a signal level of 0 volts or less as a low. For these devices the -12v supply is not necessary.

RS 232 - ITS Connections

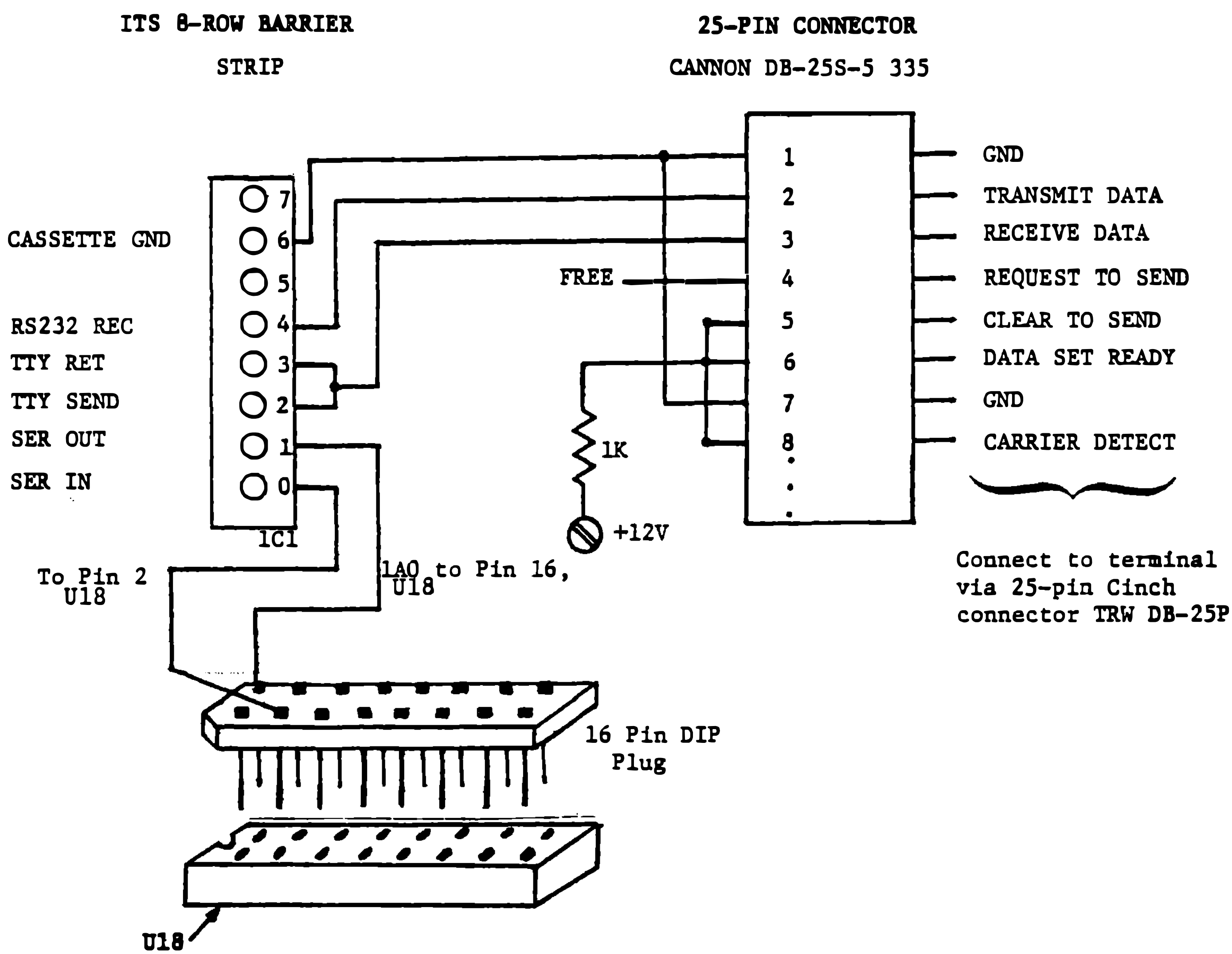


Figure C-1

TERMINAL SIGNALS	25-PIN CANNON CONNECTOR	ITS CONNECTIONS	NOTES
GND	1	GND	Screw terminal #7
TRANSMIT DATA	2	RS232 REC	Screw Terminal #4
RECEIVE DATA	3	"TTY SEND"/ "TTY RET"	Screw Terminal #2&3
REQUEST TO SEND	4	unconnected	
CLEAR TO SEND	5	Pulled up to +12V through 1K resistor	
DATA SET READY	6	Pulled up to +12V through 1K resistor	
GND	7	GND	
CARRIER DETECT	8	Pulled up to +12V through 1K resistor	

SUMMARY OF SIGNAL CONNECTIONS

Figure C-2a

BIT 8 = 0

Parity = INH

STOP = 1

Data = 8

Parity = Don't Care

RS232

FDX

BAUD RATE = 300

ADM3A SWITCH SETTINGS

Figure C-2b

C.3 ICS RS232 SOFTWARE SPECIFICATIONS

The following pages contain the formal specifications, flowcharts, and source code listings for the RS232 Software. The student is advised to note the format used in this documentation for his/her own efforts.

C.3.1 Subroutine IMSG

This subroutine inputs a string of characters from the terminal and echos the characters back to the terminal until it encounters a Carraige Return character (ODH). The routine stores the input string beginning at the address specified in the HL register pair. The parity bit (high order bit 7) is masked out before the ASCII character is stored. The last character in the buffer is always ETX (03H or CNTL-C). Register C contains the maximum input string length acceptable (input buffer size).

The PMPT entry point outputs a question mark ('?') as a prompt character prior to invoking IMSG. All arguments are the same.

* Entry Point 8297

* Arguments

  - Upon entry

    - H,L register pair contains the input buffer start address
    - C contains the maximum input buffer length minus 1

* Upon return

  - C contains the number of input buffer bytes remaining

- A contains 03H

- B and D contain 00H

- E contains last input character

- H,L register pair contain the address of the last input buffer entry (the CNTL-C, 03H, or ETX)

- The Input Buffer contains the inputted character string where the last character in the buffer is an ETX (03H or CNTL-C) character. The Carraige Return character is not stored.

- The Stack has been used and restored.

* Subroutine used

  - OCHR at 82D0H

  - ICHR at 82B0H

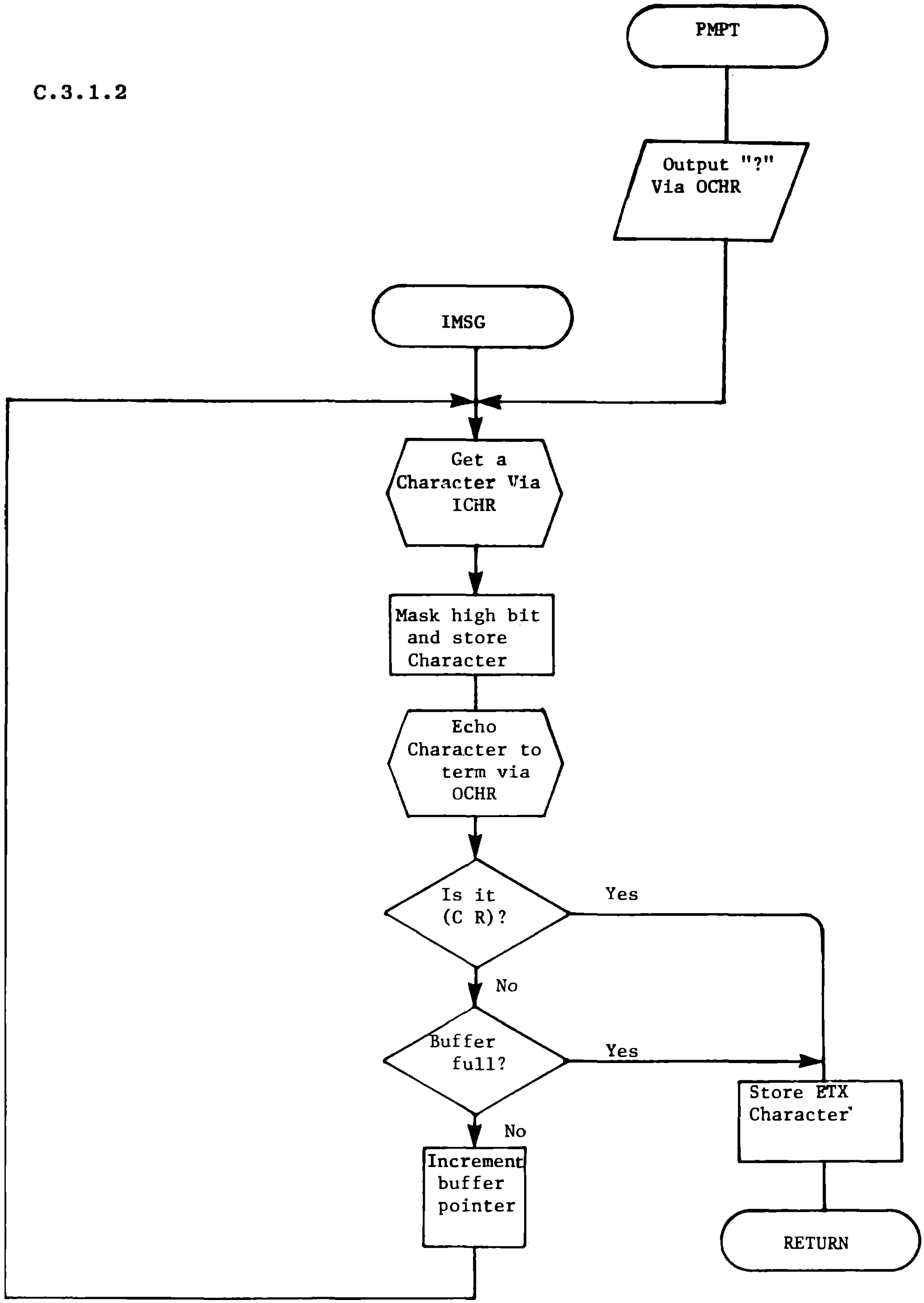
* Other entry points

  - PMPT - Output a prompt character ('?') before invoking IMSG. Entry point 8290H

* Sample usage

```
21 00 83        LXI   H,INBF     Load input buffer address
0E 1F           MVI   C,BFLEN    Load input buffer length
CD 90 82        CALL  PMPT       Prompt with '?' and input string
```

C.3.1.2



IMSG Flowchart

Figure C-3

IMSG Source Code

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		Label	Mnemonic	Operand	Comment
829	0	C	5	PMPT	PUSH	B	Prompt:
	1	1	E		MVI	E,"?"	Output '?' as a
	2	3	F				prompt character
	3	C	D		CALL	OCHR	
	4	D	0				
	5	8	2				
	6	C	1		POP	B	
829	7	C	5	IMSG	PUSH	B	IMSG Entry
	8	C	D		CALL	ICHR	Get a character from
	9	B	0				the keyboard
	A	8	2				
	B	E	6		ANI	7F	Mask-out the Parity Bit,
	C	7	F				bit 7 (MSB)
	D	7	7		MOV	M,A	
	E	C	D		CALL	OCHR	Echo the inputted
	F	D	0				character back to CRT
82A	0	8	2				
	1	C	1		POP	B	
	2	7	B		MOV	A,E	
	3	F	E		CPI	CR	Is it a Carriage Return?
	4	0	D				
	5	C	A		JZ	DONE	Yes; finish up.
	6	A	D				
	7	8	2				
	8	0	D		DCR	C	No;
	9	2	3		INX	H	Update buffer pointer
	A	C	2		JNZ	IMSG	and character counter
	B	9	7				Is buffer full?
	C	8	2				No; ⇒ Get next character
82A	D	3	6	DONE	MVI	M,03	Yes; Put EXT character
	E	0	3				into buffer and
	F	C	9		RET		return.
8	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						

Figure C-4

C.3.2 Subroutine ICHR

This subroutine inputs a character from the terminal keyboard through Port 1A0, with no translation, into the A and E registers.

* Entry point 82B0
* Arguments
    - No entry arguments
- Upon return
    - A contains the last character inputted
    - E contains a copy of A (input character)
    - B and C are 00H
    - D, H and L are unaffected
* Subroutines used
    - DLY of OCHR (at 82E9)
    - DLY1 of OCHR (at 82EB)
* Sample usage

```
CD B0 82     CALL  ICHR       Input a character
```

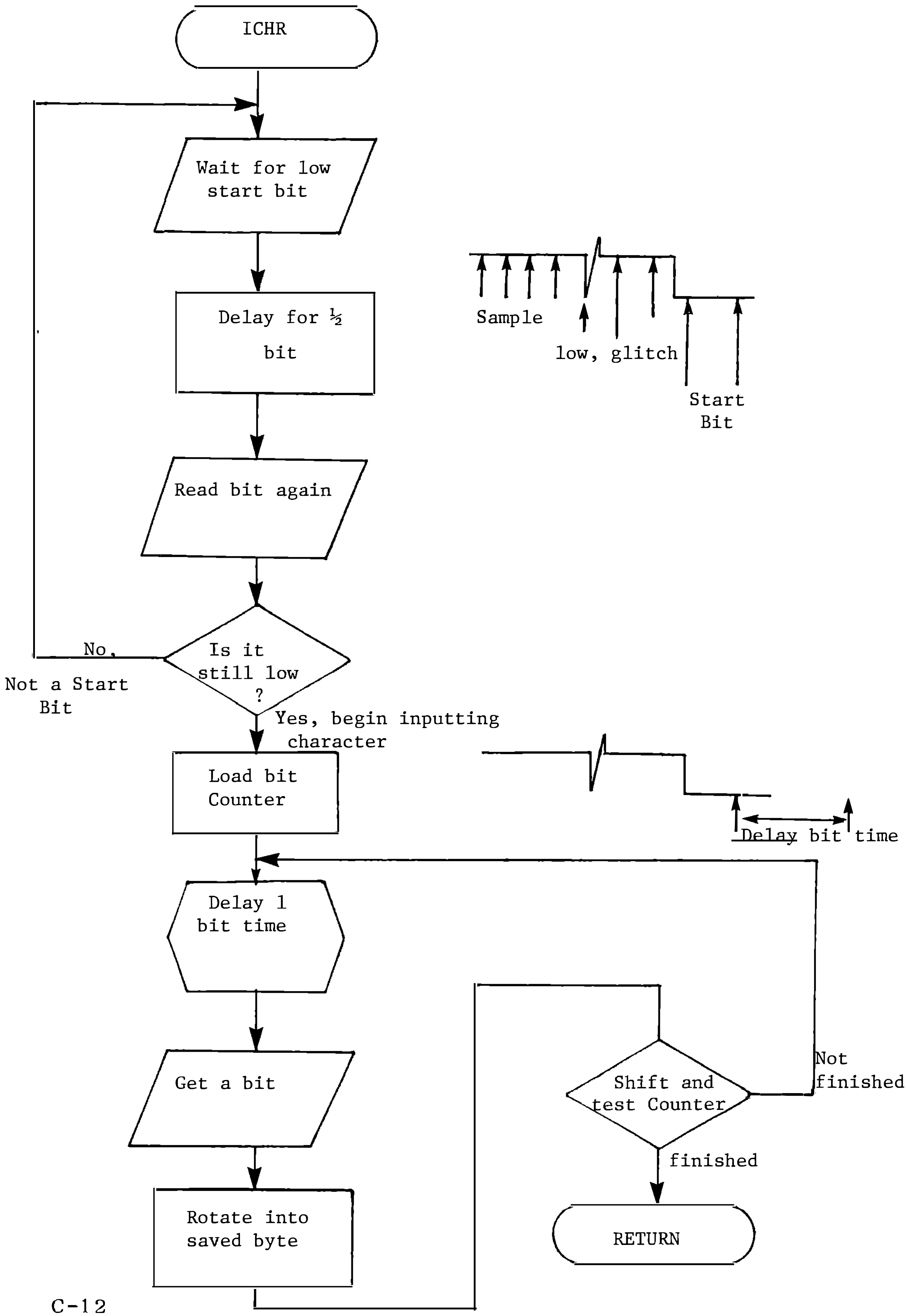

ICHR
Wait for low start bit
Delay for ½ bit
Read bit again
Is it still low ?
No, Not a Start Bit
Yes, begin inputting character
Load bit Counter
Delay 1 bit time
Get a bit
Rotate into saved byte
Shift and test Counter
Not finished
finished
RETURN
Sample
low, glitch
Start Bit
Delay bit time

Figure C-5

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

; GET K/B CHAR INTO REG A

ADDR		CODE		Label	Instruction	Comment
8 2B	0	D	B	ICHR	IN PORT1A	Get a bit
	1	0	4			
	2	1	F		RAR	
	3	D	A		JC ICHR	Wait for low
	4	B	0			start bit
	5	8	2			
	6	0	E		MVI C, 2	Delay for 1/2 bit
	7	0	2			time
	8	C	D		CALL DLY1	
	9	E	B			
	A	8	2			
	B	D	B		IN PORT1A	Read bit again
	C	0	4			
	D	1	F		RAR	Is it still low?
	E	D	A		JC ICHR	False alarm
	F	B	0			
8 2C	0	8	2			
	1	1	E		MVI E, 80	Bit Counter
	2	8	0			
82C	3	C	D	ILP	CALL DLY	Delay full bit
	4	E	9			time
	5	8	2			
	6	D	B		IN PORT1A	Get a bit
	7	0	4			
	8	1	F		RAR	CY ← (A_0)
	9	7	B		MOV A, E	Get previous bits
	A	1	F		RAR	A_7 ← (CY)
	B	5	F		MOV E, A	Save new byte.
	C	D	2		JNC ILP	Get next bit
	D	C	3			unless CY shows
	E	8	2			we've got all 8
	F	C	9		RET	
8 2D	0					
	1					Returns with last
	2					keyboard character
	3					in Regs A and E
	4					
	5					
	6					
	7					
	8					Figure C-6

C.3.3 Subroutine OCHR

This subroutine outputs the character in the E register to the terminal via Port 1C1 at 300 baud (30 chars/sec). The baud rate may be changed by changing the programmed delay in the DLY subroutine at address 82ECH.

Other entry points are DLY and DLY1 for the appropriate delays for the 300 baud (or higher) data rates.

* Entry point(s)
  - 82D0 for OCHR
  - 82E9 for DLY
  - 82ED for DLY1
* Arguments
  - Upon entry
    - E contains the character to be outputted
  - Upon return
    - E contains the outputted character
    - A contains 03H (or ETX)
    - B, C and D contain 00H
    - H and L are unaffected

* Sample usage

06 21	MVI	E,"A"	Output an 'A' to the CRT
CD D0 82	CALL	OCHR	terminal.

* Other entry points

- DLY at 82E9

- Causes the appropriate delay for a 300 baud data rate. The baud rate may be changed up to 4800 baud by recoding address 82EC as follows:

82EC	baud rate	chars/sec
70H	300	30
38H	600	60
0BH	1200	120
04H	4800	480

- DLY1 at 82EB

- Alternate delay entry allows 1/2 bit time delay (used by ICHR). Also allows 110 baud rate delay. User must load register C with appropriate counter (see source code for OCHR).

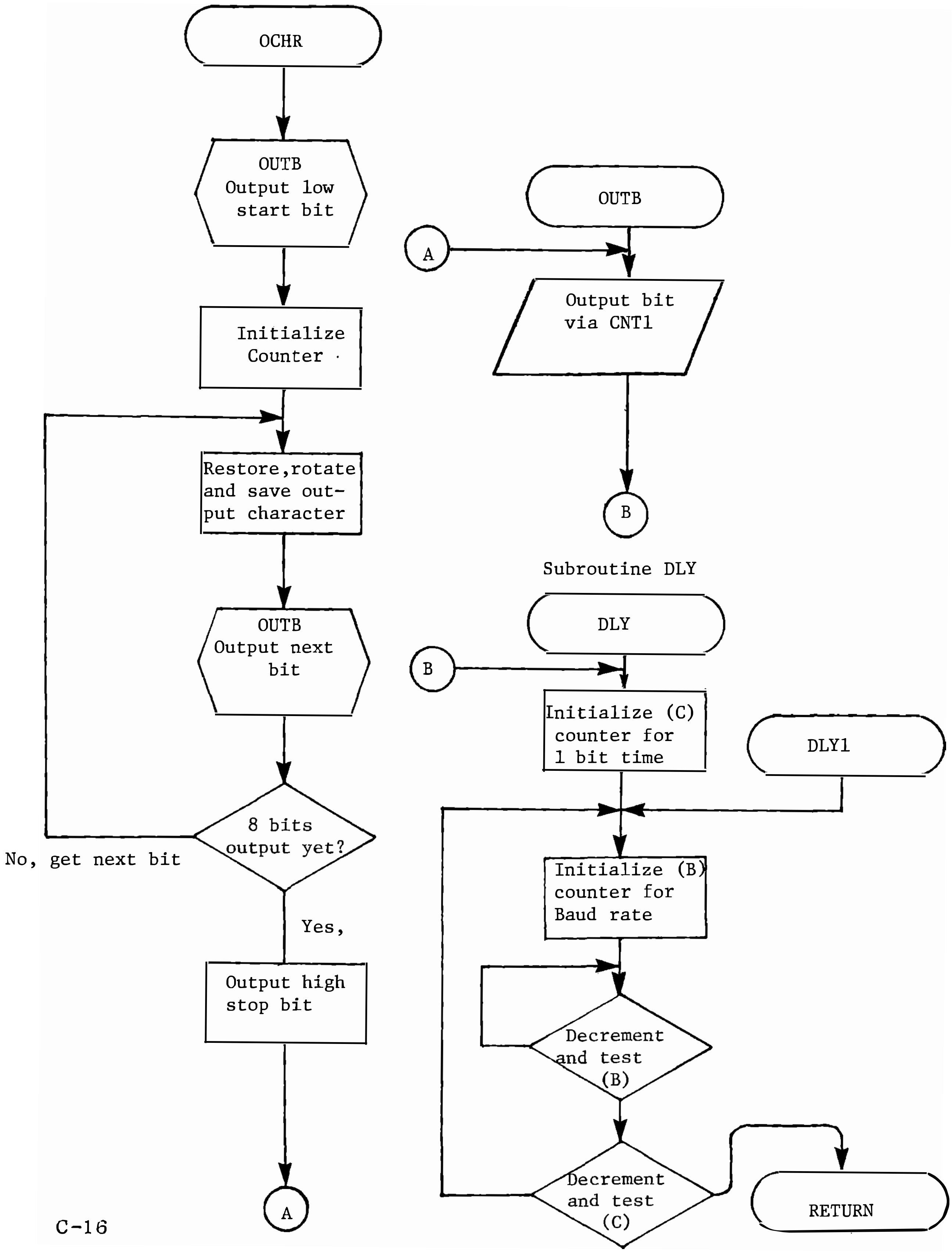

OCHR
OUTB
Output low
start bit
Initialize
Counter
Restore,rotate
and save out-
put character
OUTB
Output next
bit
8 bits
output yet?
No, get next bit
Yes,
Output high
stop bit
A
OUTB
A
Output bit
via CNT1
B
Subroutine DLY
DLY
B
Initialize (C)
counter for
1 bit time
DLY1
Initialize (B)
counter for
Baud rate
Decrement
and test
(B)
Decrement
and test
(C)
RETURN

Figure C-7

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

OUTPUT CONTENTS OF E TO RS232 DEVICE AT 300 BAUD

ADDR		CODE		LABEL	OPCODE	OPERAND	COMMENTS
82D	0	3	E	OCHR	MVI	A,02	Output a low
	1	0	2				start bit
	2	C	D		CALL	OUTB	
	3	E	7				
	4	8	2				
	5	A	F		XRA	A	Clear carry
	6	1	6		MVI	D,8	Initialize bit counter
	7	0	8				
82D	8	7	B	OLP	MOV	A,E	Get output character
	9	0	F		RRC		
	A	5	F		MOV	E,A	Save rotated character
	B	3	E		MVI	A,01	If CY then CNT1←03
	C	0	1				else CNT1←02
	D	8	F		ADC	A	
	E	C	D		CALL	OUTB	Output bit
	F	E	7				
82E	0	8	2				
	1	1	5		DCR	D	Count bits until
	2	C	2		JNZ	OLP	all 8 are done
	3	D	8				
	4	8	2				
	5	3	E		MVI	A,03	Finished, output high
	6	0	3				stop bit
82E	7	D	3	OUTB	OUT	CNT1	Output the bit
	8	0	7				
82E	9	0	E	DLY	MVI	C,03	Load counter for 1
	A	0	3				bit time
82E	B	0	6	DLY1	MVI	B,94	Load timer for baud
	C	7	0				rate: 94 = 300 baud
82E	D	0	5	DLY2	DCR	B	49 = 600 baud
	E	C	2		JNZ	DLY2	23 = 1200 baud
	F	E	D				10 = 2400 baud
82F	0	8	2				06 = 4800 baud
	1	0	D		DCR	C	
	2	C	2		JNZ	DLY1	Loop 'till done.
	3	E	B				
	4	8	2				
	5	C	9		RET		
	6						
	7						
	8						Figure C-8

C.3.4 Subroutine OMSG

This subroutine will output a block of characters, whose starting address is specified in the HL register pair, via OCHR until an ETX (03H or CNTL-C) character is encountered. (02H, 01H and 00H also terminate transmission.)

* Entry point 82F6

* Arguments

  - Upon entry

    - HL register pair contain the output buffer start address

  - Upon return

    - A contains 03H (ETX character)

    - B, C and D contain 00H

    - E contains the last character output

    - HL contain the address of the last character output +1 (The address of the ETX byte.)

* Subroutines used

  - OCHR at 82D0

* Sample usage

```
21 00 83    LXI    H,OBUF    Load ouput buffer
                              address
CD F6 82    CALL   OMSG       and output it
```

C.3.4.2 Subroutine OMSG Flowchart

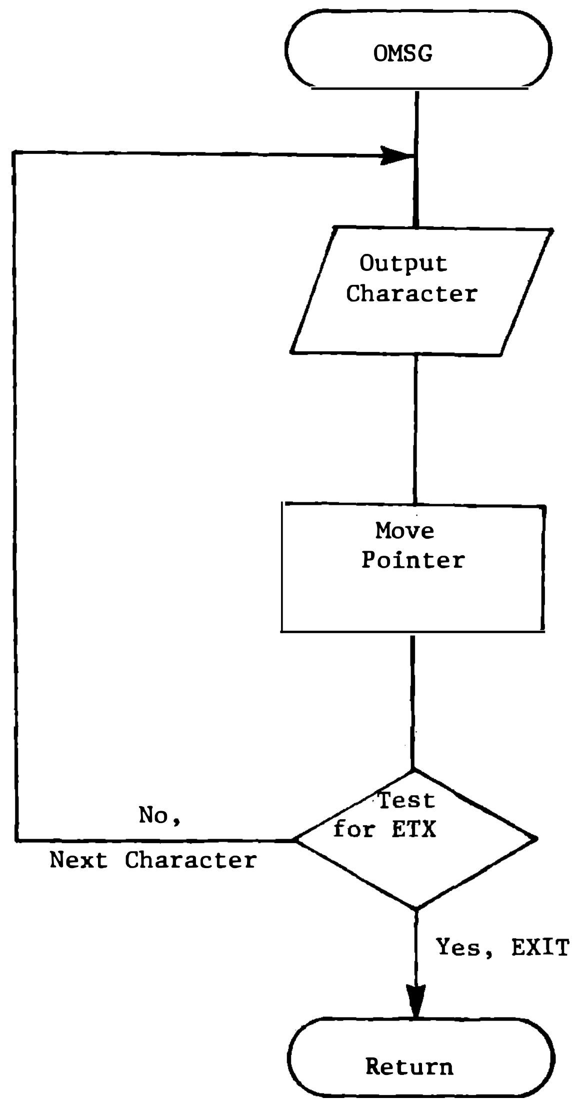


Figure C-9

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

OMSG

ADDR		CODE		Label	Instruction	Comments
8	0					Output the buffer
	1					whose address is
	2					in H, L pair to
	3					terminal until an
	4					ETX character is
	5					encountered
82F	6	5	E	OMSG	MOV E, M	Get next character
	7	C	D		CALL OCHR	
	8	D	0			
	9	8	2			
	A	2	3		INX H	Move pointer
	B	9	3		SUB E	Compare to 03
	C	F	0		RP	Exit if 00, 01, 02, or 03
	D	C	3		JMP OMSG	else go for next
	E	F	6			character
82F	F	8	2			
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	A					
	B					
	C					
	D					
	E					
	F					
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					

C.3.5 Main Calling Program

This routine exercises the RS 232 system by calling the appopriate subroutine modules to output a message to the terminal, accept an input message (with echo), and output the inputted string. The program then repeats the procedure.

* Entry point 8200H

* Arguments, none

* Subroutines used

  - IMSG at 8290 to prompt and then input a message

  - OMSG at 82F6 to output a character string message

* Sample usage procedure

  1. Load the RS232 System from the Cassette Library

  2. Verify memory using the enclosed listings

  3. Press RST, RUN

  4. The display should go blank, and the terminal should display:

     What is your name?

  5. The system is now awaiting keyboard input. Type in your response followed by a Carraige Return character (Press the <u>RETURN</u> key).

6. The system will respond with

Hello <your response>

Hi There. I am ... etc.

7. The system will repeat the sequence.

* Data Tables

- An output buffer is provided at addresses 8300H-8362H with the "Hi There ..." message

- An output buffer is provided at addresses 83A0H - 83A8H with the "Hello " character string.

- An input buffer is provided at addresses 8370H-838FH for the response character string.

NOTE: You may wish to experiment with your own messages and input and output sequences. There is ample space in the 512 bytes of RAM to code your own MAIN Calling program with your own messages. However, note that OMSG expects an ETX (03H) character as the terminating character in the output buffer.

If you have the 1K RAM option, you may wish to use the alternate IMSG routine provided at address 8000H with basic editing capabilities (Underscore for delete character, CTRL-X for delete line). The alternate IMSG is provided in Section C.4.

C.3.5.2 Main Flowchart

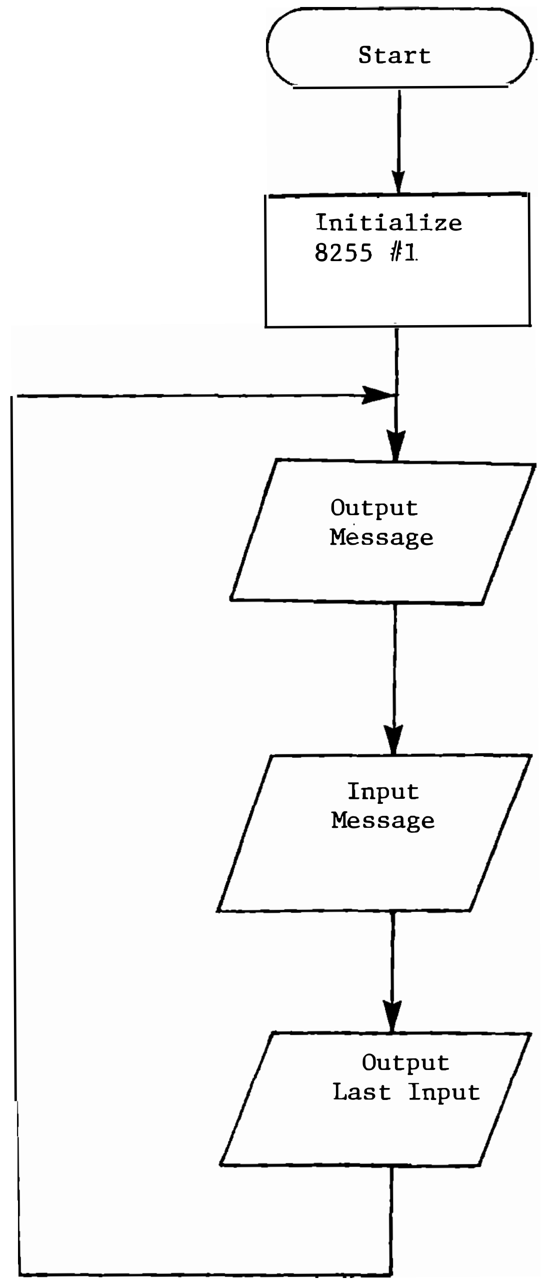


Figure C-11

RS232 INTERFACE EXERCISE PROGRAM - MAIN CALLING PROGRAM

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		LABEL	MNEMONIC	OPERAND	COMMENTS
820	0	3	E	MAIN	MVI	A, 92H	Initialize 8255 #1
	1	9	2				
	2	D	3		OUT	CNT1	
	3	0	7				
820	4	2	1	LOOP	LXI	H, 8300	Output buffer address
	5	0	0				
	6	8	3				
	7	C	D		CALL	OMSG	
	8	F	6				
	9	8	2				
	A	2	E		MVI	L, 70H	Input buffer address
	B	7	0				
	C	0	E		MVI	C, 1F	Input buffer maximum length -1
	D	1	F				
	E	C	D		CALL	PMPT	Prompt with "?" and input a message
	F	9	0				
821	0	8	2				
	1	2	E		MVI	L, A0H	"Hello" address
	2	A	0				
	3	C	D		CALL	OMSG	
	4	F	6				
	5	8	2				
	6	2	E		MVI	L, 70H	Name buffer address
	7	7	0				
	8	C	D		CALL	OMSG	
	9	F	6				
	A	8	2				
	B	C	3		JMP	LOOP	
	C	0	4				
	D	8	2				
	E						
	F						
8	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						Figure C-12

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A D D	R	CODE									
8 30	0	0	A	OBUF	D	B			L	F	Line Feed
	1	0	D		D	B			C	R	Carriage Return
	2	4	8		D	B			H		
	3	6	9		D	B			I		
	4	2	0								
	5	5	4						T		
	6	6	8						H		
	7	6	5						E		
	8	7	2						R		
	9	6	5						E		
	A	2	E						.		
	B	2	0								
	C	2	0								
	D	4	9						I		
	E	2	0								
	F	6	1						A		
8 31	0	6	D						M		
	1	2	0								
	2	7	9						Y		
	3	6	F						O		
	4	7	5						U		
	5	7	2						R		
	6	2	0								
	7	6	6						F		
	8	7	2						R		
	9	6	9						I		
	A	6	5						E		
	B	6	E						N		
	C	6	4						D		
	D	6	C						L		
	E	7	9		D	B			Y		
831	F	2	0		D	B					
8	0										
	1										
	2										
	3										
	4										
	5										
	6										
	7										
	8										Figure C-13a

OUTPUT BUFFER (continued)

ADDR	R	CODE				
8 32	0	6	3	D	B	C
	1	6	F	D	B	O
	2	6	D			M
	3	7	0			P
	4	7	5			U
	5	7	4			T
	6	6	5			E
	7	7	2			R
	8	2	0			
	9	7	2			R
	A	7	5			U
	B	6	E			N
	C	6	E			N
	D	6	9			I
	E	6	E			N
	F	6	7			G
8 33	0	2	0			
	1	6	1			A
	2	6	E			N
	3	2	0			
	4	2	0			
	5	5	2			R
	6	5	3			S
	7	3	2			2
	8	3	3			3
	9	3	2			2
	A	4	3			C
	B	2	0			
	C	4	9			I
	D	6	E			N
	E	7	4			T
833	F	6	5			E
8 34	0	7	2			R
	1	6	6			F
	2	6	7			A
	3	6	3			C
	4	6	5	D	B	E
	5	2	0			
	6					
	7					
	8					

Figure C-13b

INTEGRATED COMPUTER SYSTEMS MICROCOMPUTER TRAINING SYSTEM CODING SHEET

OUTPUT BUFFER (continued)

ADDR		CODE		LABEL	OP	OPERAND	COMMENTS
8	0						
	1						
	2						
	3						
	4						
	5						
834	6	5	0		DB	P	
	7	7	2			R	
	8	6	F			O	
	9	6	7			G	
	A	7	2			R	
	B	6	1			A	
	C	6	D			M	
	D	2	E			.	
	E	0	A			LF	Line Feed
	F	0	D			CR	Carriage Return
835	0	5	7			W	
	1	6	8			H	
	2	6	1			A	
	3	7	4			T	
	4	2	0				
	5	6	9			I	
	6	7	3			S	
	7	2	0				
	8	7	9			Y	
	9	6	F			O	
	A	7	5			U	
	B	7	2			R	
	C	2	0				
	D	6	E			N	
	E	6	1			A	
	F	6	D			M	
836	0	6	5			E	
	1	2	0				
	2	0	3		DB	ETX	End of buffer character
	3						
	4						
	5						
	6						
	7						
	8						Figure C-13c

HELLO OUTPUT BUFFER, INPUT BUFFER

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		Label	Instruction	Operand	Comment
8 3A	0	0	A	HELLO	DB	LF	HELLO message
	1	0	D		DB	CR	Linefeed, carriage return
	2	4	8		DB	"H"	
	3	6	5		DB	"E"	
	4	6	C		DB	"L"	
	5	6	C		DB	"L"	
	6	6	F		DB	"O"	
	7	2	0		DB	SPACE	
	8	0	3		DB	ETX	End of buffer
	9						
	A						
	B						
	C						
	D						
	E						
	F						
8 37	0			IBUF	DS	32H	Reserve 32 bytes
8390							of RAM for 31 input
	2						characters + EXT → 32
	3						bytes
	4						
	5						
	6						
	7						
	8						
	9						
	A						
	B						
	C						
	D						
	E						
	F						
8	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						

Figure C-13d

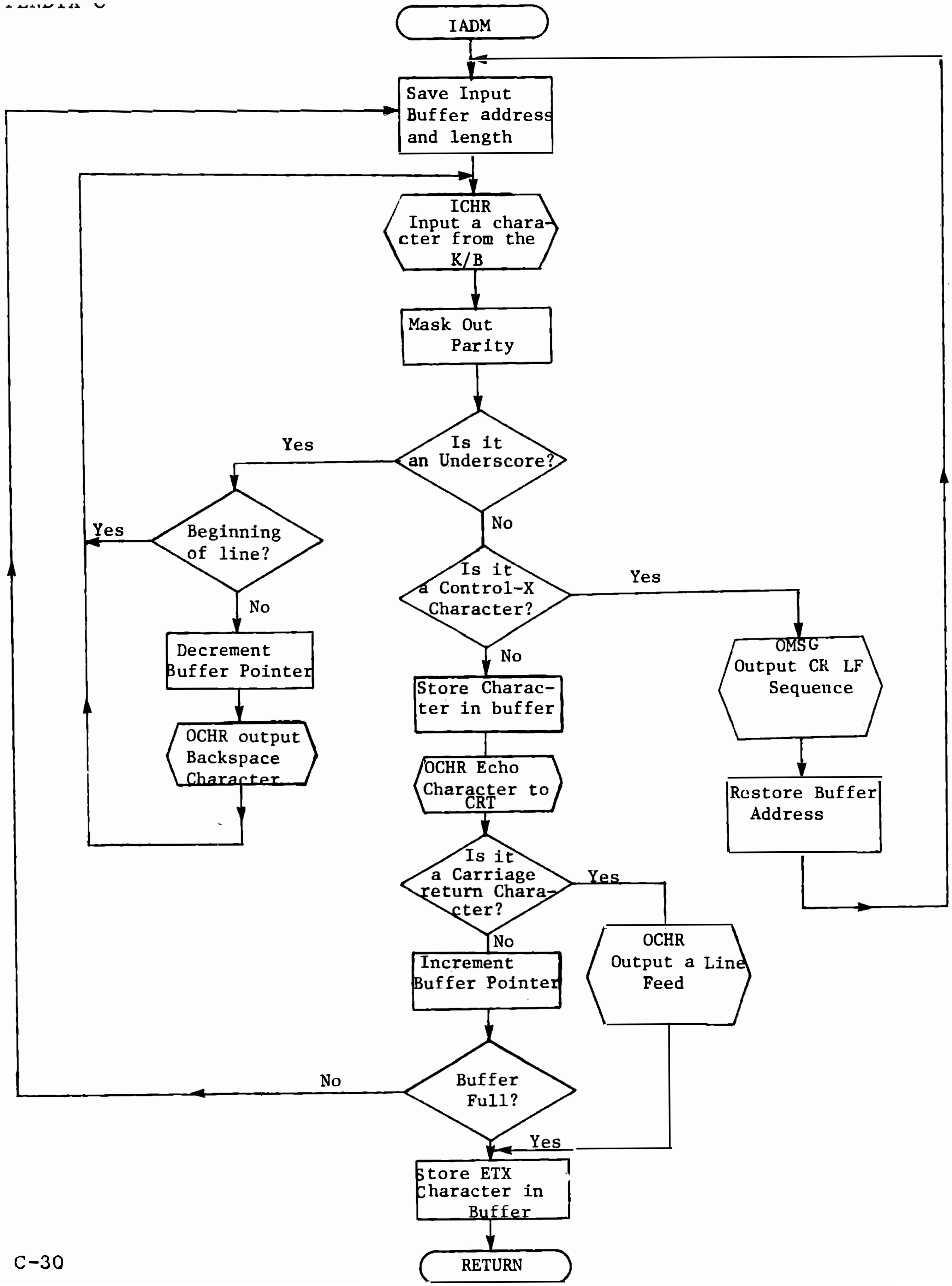
C.4 ALTERNATE IMSG PROGRAM

This subroutine is identical to the IMSG routine with the following exceptions:

1. Entry point is 8000H
2. There is no PMPT entry

character entered

4. A CTRL-X (18H) character will delete the entire input line. The flowchart is shown in Figure C-14 and the code is in Figure C-15.



Alternate IMSG, Flowchart

Figure C-14

ALTERNATE IMSG SUBROUTINE

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE											
8	0												Input a string from
	1												ADM terminals
	2												honoring underscore
	3												as a backspace
	4												character with CTL-X
	5												as a delete line
	6												character. Input
	7												buffer address
	8												H, L pair and
	9												maximum buffer
	A												length in C.
	B												
	C												
	D												
	E												
	F												
8 00	0	E	5	IADM	P	U	S	H		H			Save buffer address
	1	4	1		M	O	V		B	,	C		Save buffer length
800	2	C	5	LP1	P	U	S	H		B			
800	3	C	D	LP2	C	A	L	L		I	C	H R	Get a character
	4	B	0										
	5	8	2										
	6	E	6		A	N	I		7	F			Mask out parity bit
	7	7	F										
	8	F	E		C	P	I		"	_	"		Test for underscore
	9	5	F										
	A	C	A		J	Z				B	K	S P	Go backspace
	B	2	C										
	C	8	0										
	D	F	E		C	P	I		C	A	N		^X: Test for CNTL-X
	E	1	8										
	F	C	A		J	N	Z			D	E	L N	Yes; go delete line
8 01	0	4	4										
	1	8	0										
	2												
	3												
	4												
	5												
	6												
	7												
	8												Figure C-15a

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR		CODE					COMMENTS
8	0						
	1						
801	2	7	7		MOV	M,A	Store the character
	3	5	F		MOV	E,A	in the input buffer
	4	C	D		CALL	OCHR	Echo character
	5	D	0				to CRT
	6	8	2				
	7	C	1		POP	B	
	8	7	B		MOV	A,E	
801	9	F	E		CPI	CR	Is character a
	A	0	D				carriage return?
	B	0	6		MVI	E,LF	
	C	0	A				
	D	C	C		CZ	OCHR	Yes; add line feed
	E	D	0				character and
	F	8	2				
8 02	0	C	A		JZ	EXIT	then exit
	1	2	8				
	2	8	0				
	3	2	3		INX	H	Move buffer pointer
	4	0	D		DCR	C	Decrement character counter
	5	C	2		JNZ	LP1	and go get next
	6	0	2				character if buffer
	7	8	0				not full.
802	8	3	6	EXIT	MVI	M,03	Store EXT (03H)
	9	0	3				character at end of
	A	D	2		POP	D	msg. and return
	B	C	9		RET		* * *
802	C	C	1	BKSP	POP	B	Backspace 1 character:
	D	D	1		POP	D	Retrieve buffer address
	E	D	5		PUSH	D	
	F	7	B		MOV	A,E	
8 03	0	B	D		CMP	L	Already front line;
	1	C	A		JZ	LP1	ignore backspace
	2	0	2				
	3	8	0				
	4						
	5						
	6						
	7						
	8						Figure C-15b

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE				
8	0					
	1					
	2					
	3					
803	4	2	B		DCX H	Move buffer pointer
	5	0	C		INR C	back one character
	6	C	5		PUSH B	
	7	1	E		MVI E, "^"	Output a "move
	8	0	8			cursor left"
803	9	C	D		CALL OCHR	ADM-3 character
	A	D	0			
	B	8	2			
	C	0	0		NOP	These NOP's leave room
	D	0	0		NOP	to include code to
	E	0	0		NOP	output a backslash \
	F	0	0		NOP	delete character \
8 04	0	0	0		NOP	sequence (like D.E.C.)
	1	C	3		JMP LP2	
	2	0	3			Go get next character.
	3	8	0			
804	4	2	1	DELN	LXI H, CRLF	Delete entire line;
	5	5	0			
	6	8	0			
	7	C	D		CALL OMSG	Output Carriage Return -
	8	F	6			Line Feed sequence.
	9	8	2			
	A	C	1		POP B	Restore buffer address
	B	4	8		MOV C, B	and character
	C	E	1		POP H	counter.
	D	C	3		JMP IADM	Go back to beginning
	E	0	0			
	F	8	0			
8 05	0	0	D	CRLF	DS CR	CR, LF buffer.
	1	0	A		DS LF	"
	2	0	3		DS ETX	
	3					
	4					
	5					
	6					
	7					
	8					

Figure C-15c

This page intentionally left blank.

MICROCOMPUTER INTERFACING WORKBOOK

APPENDIX D

TELETYPE INTERFACE SYSTEM

This page intentionally left blank.

APPENDIX D

TELETYPE INTERFACE SYSTEM

This Appendix describes a procedure for interfacing a teletype to the ITS utilizing an 11 bit 110 baud code.

D.1 SPECIAL EQUIPMENT REQUIRED:

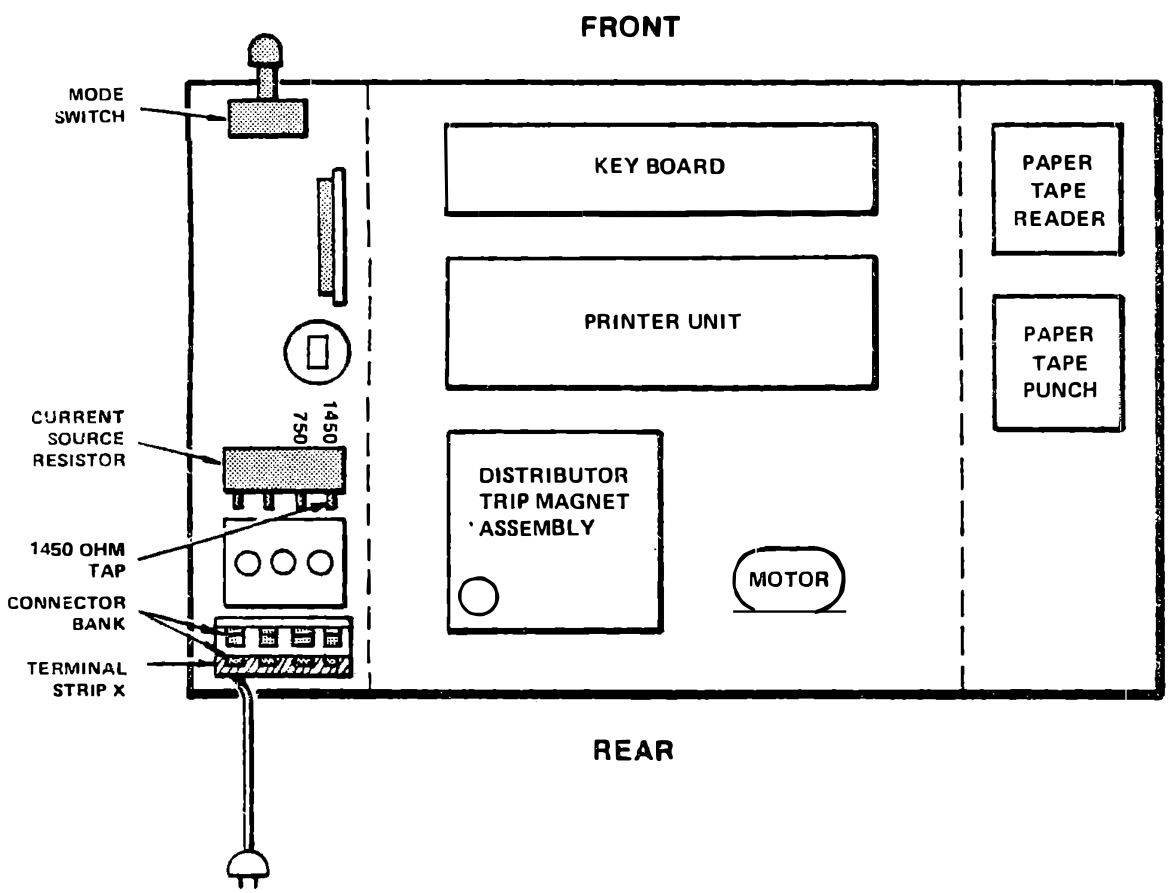
A source of -12 volts 1 volt @ 50 mA. (power supply or batteries)

D.2 MODIFICATIONS TO TELETYPE:

Inerfacing the teletype requires the following modifications to the teletype unit itself:

1. 20 milliamp current loop option
2. Half duplex option

The 20 milliamp current loop and the half-duplex connections are options available on most teletype units. Check your unit before proceeding.



Top View - Teletype with Housing Removed

Figure D-1

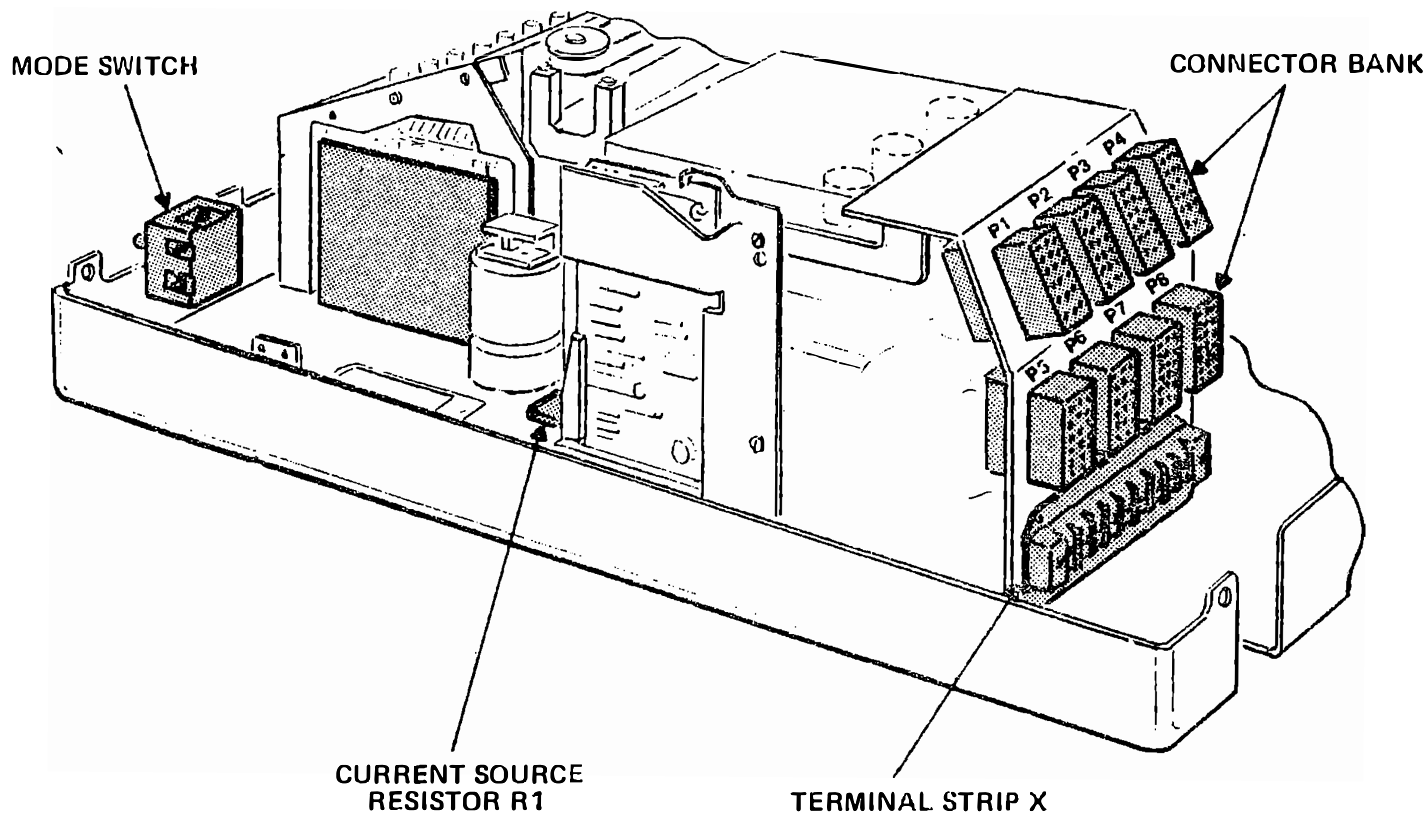
D.2.1 Removing the TTY Housing

It is necessary to remove the TTY housing to inspect or modify the TTY options.

1. Unplug the TTY from the power source.
2. Remove the roll of TTY printer paper from its cradle.
3. Remove the manual paper feed knob by pulling firmly.
4. Remove the mode select knob located on the right front by pulling firmly.
5. Remove the metal trim panel behind the mode select knob by prying downward.
6. Remove the 4 screws under the metal trim panel.
7. Remove the screw on the left side of the paper tape reader housing.
8. Remove the four knurled knobs along the lower rear edge of the housing.
9. Lift upward on the housing to remove, being careful of the controls on the paper tape reader as they clear their openings in the housing.

D.2.2 Locations of Modifications

See Figures D-1 and D-2 to locate the terminal strip X. Terminal strip X is located at the bottom of the rear of the teletype.



Side View - Teletype with Housing Removed

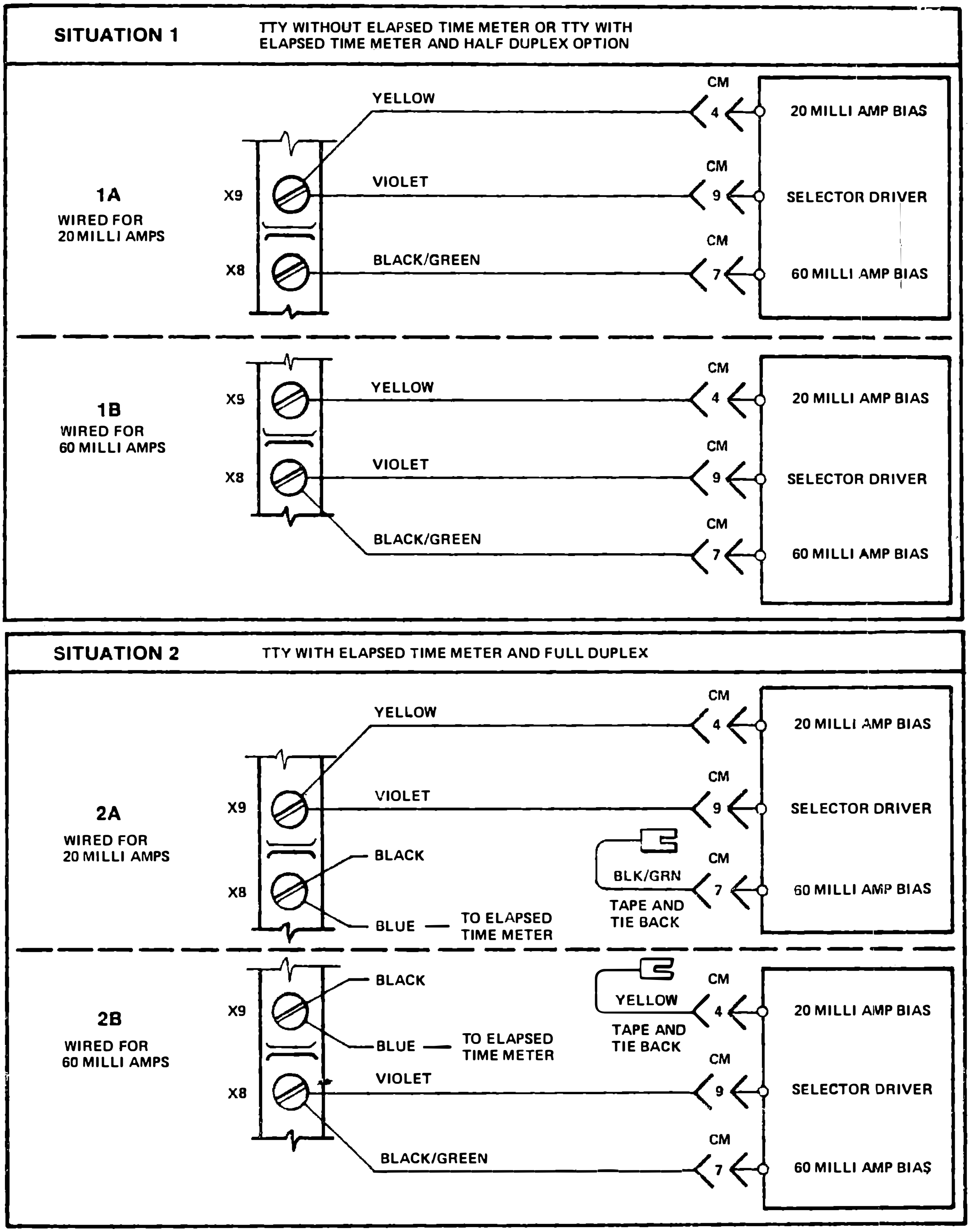
Figure D-2

D.2.3 Current Loop Option

The TTY send and receive current loop can be optionaly selected to work from either 20 milliamp or 60 milliamp. When the selection is made both the internal current source and the selector drive current bias must be modified to be compatible.

D.2.3.1 Internal Current Source

The internal current source is set to 20 milliamp by putting the blue wire on the 1450 ohm tap of power resistor R1 located on the right side of the TTY.



Current Loop Option

Figure D-3

D.2.3.2 Selector Drive Current Bias

The selector drive current bias is set to 20 milliamp by optional wiring on terminal strip X located below the connector bank in the right rear corner of the TTY. In making this change various wiring configurations may be encountered as shown in Figure D-3, depending on whether the unit has an elapsed time meter.

TTY Without Elapsed Time Meter

A TTY without an elapsed time meter may be wired either as 1A or 1B of Figure D-3. To modify for 20 milliamp:

If wired as 1A:

Do nothing; this is the correct connection for 20 milliamp without an elapsed time meter.

If wired as 1B:

Remove the violet wire from terminal X8 and move it to terminal X9 with the yellow wire.

TTY with Elapsed Time Meter

A TTY with an elapsed time meter may be wired as 1A, 1B, 2A, or 2B To modify for 20 milliamp:

If wired as 1A:

Remove the black/green wire from X8, tape the exposed end and tie-back into the wire bundle. Locate a black wire and a blue wire on teminal X5. Move both wires from X5 to terminal X8.

If wired as 1B:

Remove the violet wire from X8 and move it to X9. Remove the black/green wire from X8; tape the exposed end and tie-back into the wire bundle. Locate a black wire and a blue wire connected on terminal X5. Move both wires from X5 to X8.

If wired as 2A:

Do nothing; this is correct connection for 20 milliamp with an elapsed time meter.

If wired as 2B:

Remove the black wire and blue wire from X9. Remove the violet wire and black/green wire from X8. Connect the black wire and blue wire to X8. Connect the violet wire to X9. Locate the yellow wire taped back in the wire bundle. Connect the yellow wire to X9. Tape the exposed end of the black/green wire and tie-back into wire bundle.

D.2.4 Full Duplex Option

The full duplex option is wired into the TTY on terminal strip X located below the connector bank in the right rear corner of the unit.

If the TTY is wired for full-duplex, terminal strip X should appear as in Figure D-4.

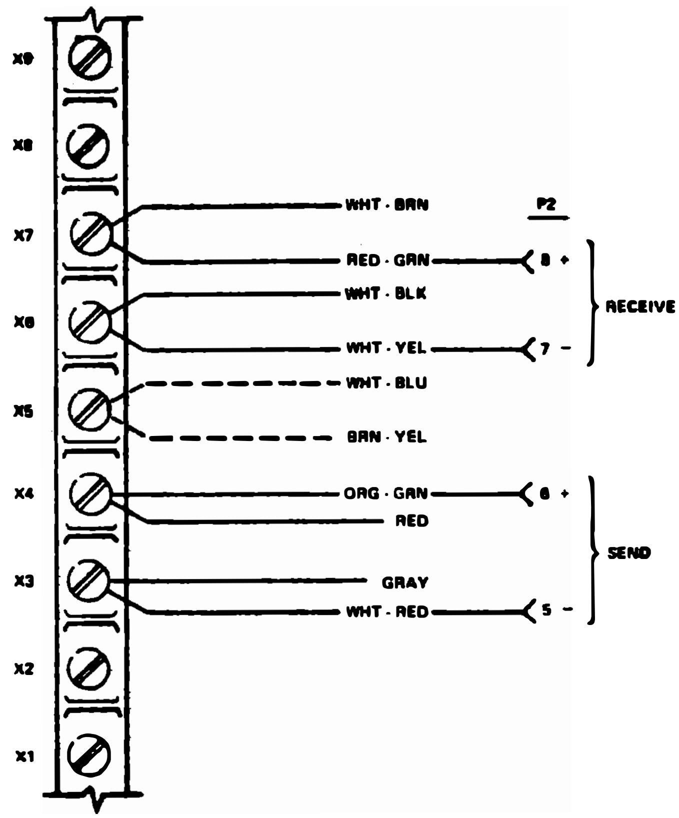
If the TTY is wired for half-duplex, terminal strip X should appear as in Figure D-5.

D.2.4.1 To Convert from Full-Duplex to Half-Duplex:

1. Confirm that screw lug X5 has a white/blue wire and a brown-yellow wire connected. If there is a BLACK wire and a BLUE wire on X5 an elapsed time meter is installed. Refer to the current LOOP option for instructions on moving the black wire and blue wire from X5 to X8.

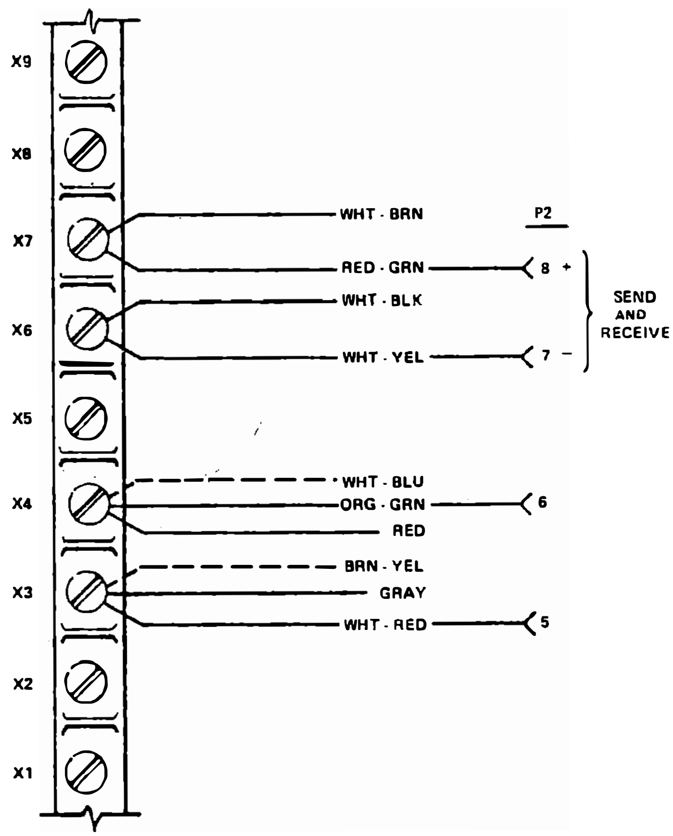
2. Move the white-blue wire from screw lug X5 to X4.

3. Move the brown-yellow wire from screw lug X5 to X3.



TTY Full-Duplex Option

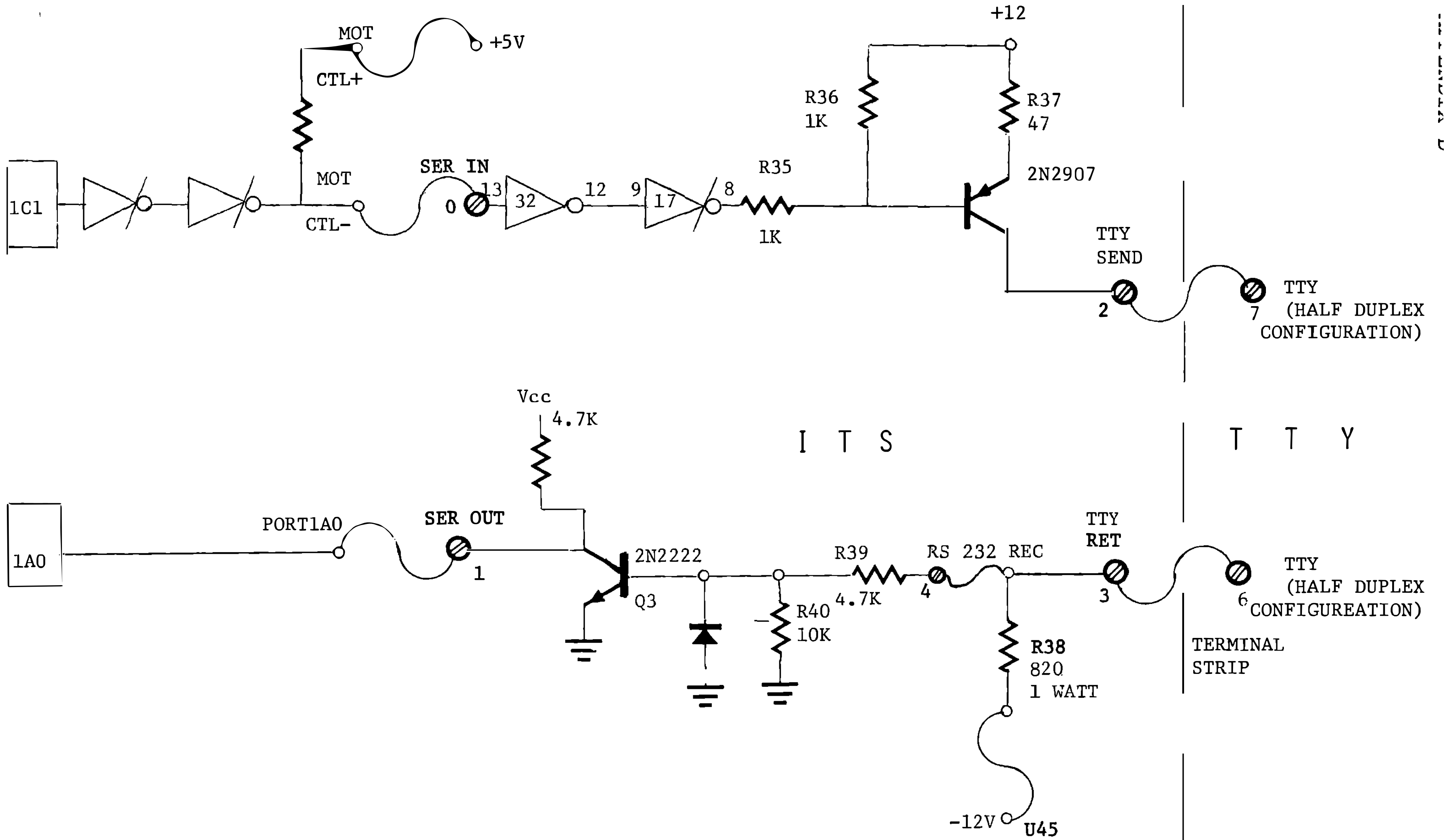
Figure D-4



TTY Half-Duplex Option

Figure D-5

This page intentionally left blank.



ITS/TTY Interface Schetmatic

Figure D-6

D.3 CONNECTING THE HALF-DUPLEX TELETYPE TO THE ITS

Figure D-6 shows the schematic of the connection between the 8255 ports on the ITS and the teletype. The following wiring will implement this schematic:

1. Connect a wire between terminal 7 on the TTY rear terminal strip and "TTY SEND" (screw #2).

2. Connect a wire between terminal 6 on the TTY rear terminal strip and "TTY RET" (screw #3).

3. Connect a jumper wire between "TTY RET" (screw #3) and "RS 232 REC" (screw #4).

4. Connect a wire between "SER OUT" (screw #1) and Port 1A0 (pin 16 on location U18).

5. Connect a wire between "SER IN" (screw #0) and "MOT CTL-" on the left side of the board.

6. Connect a wire between "MOT CTL+" and 5V on the left side of the board.

7. Finally, connect -12 volts (from pin 45 of the ribbon cable connection) to R38.

D.4 POWERING UP THE SYSTEM

Power up the MTS/ITS first. Depress "RST" on the MTS. Turn on the teletype by switching the front swich to "Line".

D.5 RUNNING THE TELETYPE PROGRAM

Enter the program of Figure D-7. Start the program by typing "RST", "RUN". A message should be typed on the teletype printer. Respond to the question by typing your name followed by the "RETURN" key. It will respond by typing a second mesage followed by the response you typed in.

D.5.1 Description of Program Modules

The program listing contans the following modules:

START:	Main routine (8200 - 822A)
OMSG:	TTY output message routine (8250 - 825B)
ICHR:	TTY character input routine (82B0 - 82CF)
OCHR:	TTY character output routine (82D0 - 82FF)
MESSAGES:	Buffers for output and input (8300 - 837A)

TELETYPE PROGRAM - MAIN ROUTINE

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

ADDR		CODE	LABEL	INSTRUCTION	COMMENT
8 20	0	3E	START	MVI A, 92	Initialize 8255 #1
	1	92			A in, C out
	2	D3		OUT 07	
	3	07			
	4	21	LOOP1	LXI H, 8300	1st message pointer
	5	00			
	6	83			
	7	CD		CALL OMSG	
	8	50			
	9	82			
	A	21		LXI H, 8354	Input buffer pointer
	B	54			
	C	83			
	D	CD	LOOP	CALL ICHR	Input character
	E	B0			
	F	82			
8 21	0	FE		CPI 8DH	Is it CR?
	1	8D			
	2	CA		JZ SKIP	Yes; go on.
	3	1A			
	4	82			
	5	77		MOV M, A	No; store character
	6	23		INX H	Increment buffer pointer
	7	C3		JMP LOOP	Get next character
	8	0D			
	9	82			
	A	AF	SKIP	XRA A	Terminate input
	B	77		MOV M, A	buffer
	C	21		LXI H, 8346	2nd message pointer
	D	46			
	E	83			
	F	CD		CALL OMSG	Output 2nd message
8 22	0	50			
	1	82			
	2				
	3				
	4				
	5				
	6				
	7				
	8				

Figure D-7a

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

TTY OUTPUT MESSAGE ROUTINE

ADDR		CODE		LABEL	INSTRUCTION	COMMENT
8	0					
	1					
8222	2	2	1		LXI H, 8370	3rd message pointer
	3	7	0			
	4	8	3			
	5	C	D		CALL OMSG	Output 3rd message
	6	5	0			
	7	8	2			
	8	C	3		JMP LOOP1	
	9	0	4			
		8	2			
					OUTPUT MESSAGE SUBROUTINE	
825	0	7	E		MOV A, M	Get character
	1	F	E		CPI 00	Is it zero?
	2	0	0			
	3	C	8		RZ	Exit if end
	4	5	F		MOV E, A	No, send character
	5	C	D		CALL OCHR	
	6	D	0			
	7	8	2			
	8	C	3		JMP OMSG	Loop
	9	5	0			
	A	8	2			
	B					
	C					
	D					
	E					
	F					
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					Figure D-7b

TELETYPE CHARACTER INPUT ROUTINE, BYTE IS PLACED IN REGISTERS A AND E. REQUIRES OCHR ROUTINE

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		Label	Instruction	Operand	Comments
8 2B	0	C	D	ICHR	CALL	TBIT	Input Port 1A0
	1	F	B				
	2	8	2				Start bit detected?
	3	D	A		JC	ICHR	No; keep trying
	4	B	0				
	5	8	2				
	6	0	E		MVI	C, 04	Yes; wait 1/2 bit
	7	0	4				
	8	C	D		CALL	DLY1	
	9	F	0				
	A	8	2				
	B	C	D		CALL	TBIT	Start bit still there?
	C	F	B				
	D	8	2				
	E	D	A		JC	ICHR	No; try again
	F	F	B				
8 2C	0	8	2				
	1	1	E		MVI	E, 80	Initialize
	2	8	0				bit counter
	3	C	D	ILP	CALL	DLY	Wait for next bit
	4	E	E				
	5	8	2				
	6	C	D		CALL	TBIT	Input Port 1A0
	7	F	B				
	8	8	2				
	9	7	B		MOV	A, E	Get previous bits
	A	1	F		RAR		MOV carry into bit 7
	B	5	F		MOV	E, A	Save present bits
	C	D	2		JNC	ILP	Get next bit until
	D	C	3				CY shows we got
	E	8	2				all 8
	F	C	9		RET		
8 2D	0						
	1						Returns with
	2						Last keyboard
	3						character in
	4						regs A and E
	5						
	6						
	7						
	8						Figure D-7c

OUTPUT CONTENTS OF REG E TO ASR 33 TELETYPE

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A D D R		CODE					
8 2D	0	3E	OCHR	MVI	A,03		Output start bit
	1	03					
	2	CD		CALL	OUTB		
	3	EC					
	4	82					
	5	AF		XRA	A		Clear carry
	6	16		MVI	D,08		Initialize Counter
	7	08					
	8	7B	OLP	MOV	A,E		Get character
	9	2F		CMA			Invert it
	A	0F		RRC			Shift into carry
	B	2F		CMA			Invert it back
	C	5F		MOV	E,A		Save it
	D	3E		MVI	A,01		If CY = 0, then A→02
	E	01					If CY = 1, then A→03
	F	8F		ADC	A		
8 2E	0	CD		CALL	OUTB		Output to Port KC1
	1	EC					
	2	82					
	3	15		DCR	D		Done 8 bits?
	4	C2		JNZ	OLP		Not yet
	5	D8					
	6	82					
	7	CD		CALL	STOPB		Yes; output 1st stop bit
	8	EA					
	9	82					Fall into 2nd stop bit
	A	3E	STOPB	MVI	A,02		Stop bit routine
	B	02					
	C	D3	OUTB	OUT	CNTL		Bit output routine
	D	07					
	E	0E	DLY	MVI	C,09		Delay 1 baud routine
	F	09					
8	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						Figure D-7d

TELETYPE PROGRAM INPUT AND OUTPUT ROUTINES (continued)

CODING SHEET — MICROCOMPUTER TRAINING SYSTEM — INTEGRATED COMPUTER SYSTEMS

A D D R		CODE		Label	Instruction	Operand	Comment
8 2F	0	0	6	DLY1	MVI	B, 88	Delay 1 millisecond
	1	8	8				
	2	0	5	DLY2	DCR	B	
	3	C	2		JNZ	DLY2	
	4	F	2				
	5	8	2				
82F	6	0	D		DCR	C	
	7	C	2		JNZ	DLY1	
	8	F	0				
	9	8	2				
	A	C	9		RET		
82F	B	D	B	TBIT	IN	PORT1A	Get bit
	C	0	4				
	D	2	F		CMA		Invert bit
	E	1	F		RAR		Bit → CY
	F	C	9		RET		
8	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						
	9						
	A						
	B						
	C						
	D						
	E						
	F						
8	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						Figure D-7e

TELETYPE PROGRAM MESSAGES

A D D	R	CODE		
8 30	0	8	D	CR
	1	8	D	CR
	2	8	A	LF
	3	C	8	H
	4	C	5	E
	5	C	C	L
	6	C	C	L
	7	C	F	O
	8	A	C	,
	9	A	0	
	A	C	D	M
	B	D	9	Y
	C	A	0	
	D	C	E	N
	E	C	1	A
	F	C	D	M
8 31	0	C	5	E
	1	A	0	
	2	C	9	I
	3	D	3	S
	4	A	0	
	5	D	4	T
	6	C	8	H
	7	C	5	E
	8	A	0	
	9	C	D	M
	A	D	4	T
	B	D	3	S
	C	A	F	/
	D	D	4	T
	E	D	4	T
	F	D	9	Y
8 32	0	A	0	
	1	D	4	T
	2	C	5	E
	3	D	3	S
	4	D	4	T
	5	A	0	
	6			
	7			
	8			Figure D-7f

TELETYPE PROGRAM MESSAGES (continued)

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADD	R	CODE		
8 32	0			
	1			
	2			
	3			
	4			
	5			
	6	D	0	P
	7	D	2	R
	8	C	F	O
	9	C	7	G
	A	D	2	R
	B	C	1	A
	C	C	D	M
	D	A	E	.
	E	8	D	CR
	F	8	D	CR
8 33	0	8	A	LF
	1	D	7	W
	2	C	8	H
	3	C	1	A
	4	D	4	T
	5	A	0	
	6	C	9	I
	7	D	3	S
	8	A	0	
	9	D	9	Y
	A	C	F	O
	B	D	5	U
	C	D	2	R
	D	A	0	
	E	C	E	N
	F	C	1	A
8 34	0	C	D	M
	1	C	5	E
	2	B	F	?
	3	A	0	
	4	0	0	EOF
	5			
	6			
	7			
	8			Figure D-7g

TELETYPE PROGRAM MESSAGES (continued)

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ADDR		CODE		
8	0			
	1			
	2			
	3			
	4			
	5			
834	6	8	D	CR
	7	8	D	CR
	8	8	A	LF
	9	D	4	T
	A	C	8	H
	B	C	1	A
	C	C	E	N
	D	C	B	K
	E	A	0	
	F	D	9	Y
8 35	0	C	F	O
	1	D	5	U
	2	A	C	,
	3	A	0	
	4	0	0	EOF
	5	0	0	
	6	0	0	
	7	0	0	
	8	0	0	
	9	0	0	
	A	0	0	
	B	0	0	
	C	0	0	
	D	0	0	
	E	0	0	
	F	0	0	
8 36	0	0	0	
	1	0	0	
	2	0	0	
	3	0	0	
	4	0	0	
	5	0	0	
	6	0	0	
	7	0	0	
	8	0	0	EOF

Figure D-7h

TELETYPE PROGRAM MESSAGES (continued)

A D D	R	CODE		
8	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
836	9	0	0	EOF
	A	0	0	
	B	0	0	
	C	0	0	
	D	0	0	
	E	0	0	
	F	0	0	EOF
837	0	8	D	CR
	1	8	D	CR
	2	8	A	LF
	3	8	A	LF
	4	8	A	LF
	5	8	A	LF
	6	8	A	LF
	7	8	7	BELL
	8	8	7	BELL
	9	8	7	BELL
	A	0	0	EOF
	B			
	C			
	D			
	E			
	F			
8	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			Figure D-7i

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

This page intentionally left blank.

INTEGRATED COMPUTER SYSTEMS

EDUCATION IS OUR BUSINESS™

NORTH AMERICAN HEADQUARTERS
Integrated Computer Systems
3304 Pico Boulevard
P.O. Box 5339
Santa Monica, California 90405 USA
Telephone: (213) 450-2060
TWX: 910-343-6965

NORTH AMERICA – EASTERN REGION
Integrated Computer Systems
300 North Washington Street
Suite 103
Alexandria, Virginia 22314 USA
Telephone: (703) 548-1333
TWX: 710-832-0045

EUROPEAN HEADQUARTERS
ICSP – U.K.
Pebblecoombe, Tadworth
Surrey KT20 7PA
England
Telephone: Leatherhead (03723) 79211
Telex: 915133

FRANCE
ICS France
90 Ave Albert 1er
92500 Rueil-Malmaison
France
Telephone: (01) 749 40 37
Telex: 204593

GERMANY
ICSD GmbH
Leonrodstrabe 54
8000 Munich 19
West Germany
Telephone: (089) 19 80 66
Telex: 5215508

SCANDINAVIA
ICSP Inc. – Scandinavia
Utbildningshuset AB
Box 1719
S-221 01 Lund, Sweden
Telephone: (046) 30 70 70
Telex: 33345

06/8/0