

H A - 8 9 - 3

C o l o r G r a p h i c s

B o a r d

D o c u m e n t a t i o n

NEW ORLEANS GENERAL DATA SERVICES, INC.
7230 CHADBOURNE DRIVE
NEW ORLEANS, LOUISIANA
(504) 241-9495

Parts of this document contain material copyrighted by other manufacturers. This material is reprinted with the permission of the copyright holders. No material in this document may be reproduced without the expressed written consent of the copyright holder.

(c) copyright 1982 by: New Orleans General Data Services, Inc.
7230 Chadbourne Drive
New Orleans, Louisiana 70126
(504) 241-9495

All rights reserved.

Table of Contents

Introduction	1
Specification	2
Installation	3
Power Modification	3.1
Interfacing and Initial Adjustments	3.2
Interfacing Information	4
Connector Pin Numbering	4.1
Joystick Interfacing Information	4.2
Composite Video Interface	4.3
Audio and DAC Interface	4.4
Counter Timer Interface	4.5
Distribution Diskette	5
Diagnostic Software (H893DIAG)	5.1
Colors	5.2
Color Bars	5.3
Adjustments	6
Adjusting the Composite Video Output	6.1
Mix Level	6.2
Board Address	6.3
Interrupts	6.4
Programming the HA-89-3	7
The TMS-9918A Video Display Processor (VDP)	7.1
The AY-3-8910 Programmable Sound Generator (PSG)	7.2
The ADC0808/ADC0809 Analog to Digital Converter (A/D)	7.3
The 8253 Counter/Timer Chip (CTC)	7.4
The 8259A Programmable Interrupt Controller (PIC)	7.5
Theory of Operation	8
H89 Bus Interface	8.1
Device Selection	8.2
Video Display Processor	8.3
Programmable Sound Generator	8.4
Analog to Digital Converter	8.5
Counter/Timer Chip	8.6
Programmable Interrupt Controller	8.7
Warranty and Service	9

1 Introduction

The HA-89-3 Color Graphics Board uses the TMS-9918A Color Video Display Generator to add color graphics to the H/Z-89 computer. Also included is a Programmable Sound Generator (PSG) for generating sound effects. An 8-input analog-to-digital converter allows for 4 X-Y joystick consoles (not included). Two 8-bit parallel I/O ports are provided which may be used for controlling lights and switches. In addition it includes a three channel counter/timer (8253) and a priority interrupt controller (8259). Three options are also available. These include an arithmetic processor (9511A or 9512), a pair of 12 bit analog to digital converters and a Votrax™ phoneme speech synthesizer.

The board comes with a shunt to select which I/O select is to be used and is supplied with driver routines, diagnostic software, and demonstration programs.

2 SPECIFICATIONS

Power Requirements for HA-89-3 board with all three options

+5 Volt Supply	1000 ma (typ)	1300 ma (max)
+12 Volt Supply	100 ma (typ)	150 ma (max)
-12 Volt Supply	60 ma (typ)	100 ma (max)

Note: Power requirements without Am9511A/8231A

+12 Volt Supply	50 ma (typ)	75 ma (max)
-----------------	-------------	-------------

Temperature

Operating Temperature	0 C (min)	40 C (max)
-----------------------	-----------	------------

Interfaces

H89 Bus Data Bus Transceivers	74LS245
H89 Bus Other Inputs	2 LS TTL loads or less
H89 Bus Open Collector Driver	7406

Video Amplifier Output Impedance	75 Ohms
Video Output	Adjustable to NTSC composite video

Audio Output Impedance	1K Ohms (min)
Audio Output (PSG and PSS)	1 Volt peak-to-peak (typ)
Audio output (Mixer)	1 Volt peak-to-peak (typ)
Note: Mixer output nominally 1 Volt, adjustable from 0 to 2 Volts	

A/D Input Impedance	10K Ohms (typ)
---------------------	----------------

D/A Output Impedance	10K Ohms
----------------------	----------

Parallel Inputs	Internal Pullups (From AY-3-8910)
-----------------	--------------------------------------

Parallel Output	1 TTL Load
-----------------	------------

NOGDS HA-89-3 Color Graphics Board

VDP Section

Video Display Processor	TMS-9918A
RAM	16K X 8
Resolution	256 X 192 pixels
Colors	15 + Transparent
Number of X,Y Movable Sprites	32
Text Mode	24 X 40 characters

PSG Section

Programmable Sound Generator	AY-3-8910
Tone Channels	3
Noise Channels	1
Envelope Generator	1
Parallel I/O Ports	2 - 8 bit
Output Impedance	1K Ohms (min)

ADC Section

Analog Multiplexer/Converter	ADC0809
Analog Input Channels	8
A/D Resolution	8 bits, binary, unipolar
A/D Accuracy	+/-1 LSB
A/D Convert Time	83 us (max)
Analog Mux Delay Time	2.5 us (max)

CTC Section

Counter/Timer Chip	8253
Counter Resolution	16 bits
Count Rate	2 MHz (max)
Number of Counters	3
Counter 0 input	1.79 MHz
Counters 1 and 2 input	external

PIC Section

Programmable Interrupt Controller	8259A
Number of Priority Interrupts supported	7
Number of Interrupts Maskable	All
CPU Interrupt Vector	Selectable

NOGDS HA-89-3 Color Graphics Board

PSS Section (option 1)

Phoneme Speech Synthesizer	Votrax SC-01
Number of Phonemes	64
Master Clock Rate	Adjustable
Software setable Master Clock Rates	2
Software setable Inflection levels	4
Output Impedance	1K ohms (min)

APU Section (option 2)

Arithmetic Processing Unit	Am9511A/8231A
Fixed Point Operations	16 and 32 bit
Floating Point Operations	32 bit
Arithmetic Operations	Add, Subtract, Multiply and Divide
Other Operations	Trig, Inv-Trig, Sqrt, Log, Exp, Float-fix conv
Floating point Divide, 32 bit with 3.58MHz Clock	43 to 51 us
Cosine, 32 bit Floating Point with 3.58MHz Clock	1150 us

DAC Section (option 4)

Digital to Analog Converter	AD7542
Number of DAC Converters	2
Resolution	12 bits
Accuracy (Tmin to Tmax)	+/- 1 LSB
Reference Voltage Source (5.0 +/- 15mv)	AD584J
Output Impedance	10K Ohms (min)

3 Installation

Installation is broken into two major categories: Power Modifications and Interfacing and Initial Adjustments. This section is concerned only with those aspects concerning installation and initial set-up. For more detailed information on adjustments refer to Section 6. While the HA-89-3 board will not be inserted until you reach section 3.2 below, this warning is placed so that you are sure to see it.

*** WARNING ***

Be very careful that you insert the board correctly. The components must be facing the middle of the computer (CRT neck) and care must be taken that it is correctly inserted in the CPU logic board. Misalignment with the pins on the CPU logic board is almost certain to severely damage the HA-89-3 board because of the power connections from the H/Z-89. A board which has been misaligned will not be eligible for warranty repair. Do not insert the board until you reach section 3.2 below.

3.1 Power Modifications

There are two modifications that may need to be made to the H/Z-89. One involves the upgrade of the +5V regulator and the other consists of the addition of a ground strap to the CPU board. A heat sink and a regulator with improved specifications are shipped with the HA-89-3 board. Newer versions of the H/Z-89 are already equipped with an improved regulator and heat sink, while older models are not. Thus it will not be necessary to replace the +5V regulator on newer systems unless low voltage problems (less than +4.75V) occur.

Unless otherwise stated, assume that you are facing the CRT screen when directions are given such as left, right, front or rear. It will be necessary to open the cover of the computer before proceeding with the following steps. A small screwdriver should be used to release the latches on each side of the computer. The latch should be visible about four inches back from the CRT faceplate between the bottom chassis and cover. Insert the screwdriver in the notch and move the latch forward. Apply a small upward pressure with the latch on the left side in the forward position, and the cover on that side should be released. Do the same to the latch on the right side and the cover should now be free and open on its hinges. Swing the cover all the way back until completely opened. Now proceed with the following steps.

First you must locate U101, the +5V regulator in your system.

With power off and the line cord disconnected, open the cabinet cover as described in your manual.

Unplug the fan and remove the top cover.

The power supply is located in the far (rear) right-hand side. Face the right side and you will find an aluminum plate near the back with two regulators mounted near the top. The one on the right top of this plate is U101 (closest to the back of the computer). It has a red wire connected to its socket over to the P516 plug on the CPU logic board.

If your system does not have a black heat sink on U101, then the IC regulator U101 should be discarded and replaced with the 78H05A regulator and heat sink, then proceed to 3.1.1 below.

If your system does not have a 78H05 for U101, then it should be replaced with the 78H05A +5V regulator supplied, so proceed to section 3.1.1.

If your system does have a heat sink on U101 and a 78H05 regulator, then section 3.1.1 may be skipped unless low voltage (less than +4.75) problems exist on the +5V supply. If low voltage problems occur then perform section 3.1.1 since the regulator supplied has better specifications than the one supplied by Heath/Zenith.

3.1.1 Replacement of +5V Regulator

The following is a step by step procedure to remove the old regulator and replace it with the 78H05A supplied with the board along with the heat sink, if necessary. To the rear of the computer there are two large circuit boards of approximately 10" by 11" in size. The one closest to the front (exposed) of the computer is the CPU logic board. It should be recognizable by the cable plugged into P516 on the top center right side of the board. It also has the I/O boards plugged into it.

Disconnect the four cables (P101, P102, P103, P104) from the power supply circuit board, which is located next to the aluminum plate found in the previous section. Also disconnect the cable to the CPU logic board at P516 (five total).

Remove the four corner screws and set the circuit board aside.

Remove the four hex spacers and lift the power supply heat sink assembly from the computer. Be sure not to disconnect any of the soldered wires that are connected to the heat sink assembly.

Remove the two screws that hold the socket of U101 to the heat

sink assembly. You may now discard the old regulator but keep the screws and socket.

Remove the Thermalloy™ Insulcote spacer from its plastic enclosure. Next, insert it on the bottom of the new regulator being careful that the spacer lies flat. Care should be taken to clean your hands after this step so as not to spread the thermal heat sink compound on other components.

Install the new regulator on the new black heat sink provided with the HA-89-3, then place it in the TO-3 socket in the same manner as the old one so that the screw holes line up with the plate, heat sink and socket. Once properly seated in its mounting holes install the two screws that were previously removed and saved.

Remount the power supply heat sink assembly with the four hex spacers previously removed.

Remount the power supply circuit board with the four 6-32 screws that were previously removed. If there was a lockwasher on the far left hex spacer, replace it.

Replug cables P101 and P103 to the power supply circuit board. Do not replug P102 or 104.

Connect a voltmeter with the negative (common) lead to the common point near the middle of the aluminum plate of the heat sink assembly. It should have four black wires attached to it and be in electrical contact with the aluminum plate. Connect the other lead to the right solder tab of socket U101.

Plug the line cord in and turn on the power while watching the voltmeter. The voltmeter should read between +4.75 and +5.25 volts DC. If it does not, power off immediately and contact NOGDS or an authorized Heath/Zenith service center.

If the +5V test was successful, turn off the H/Z-89 and unplug the line cord.

Disconnect the voltmeter and place it out of the way. Replug the remaining two cables (P102 and P104) to the power supply board and the one to the CPU logic board at P516.

3.1.2 Grounding

The ground return of many H/Z-89 systems seems to need a lower resistance path than that provided when high current cards such as the HA-89-3 are used. An additional ground connection assembly is provided to eliminate this problem. This assembly consists of a wire with a female "disconnect" and a male "tab disconnect".

If undesirable noise appears on the color display, then it may be necessary to add this extra ground strap provided with the board. In general, less picture interference occurs if this ground connection is included, so it is recommended that it always be installed. If it is decided not to include it, skip to section 3.2 below.

Turn power off and unplug the line cord.

For the following step it will be necessary to slide the CPU board high enough to gain access to the back of a screw on the top aluminum plate. The two end screws securing the card via this plate must be temporarily removed. Carefully push the wires, at the top right of the board, to the side and lift the CPU board about an inch and a half.

Looking at the CPU logic board from the front of the system there is an aluminum strap at the top of the board upon which regulators U567 and U568 are mounted. Mount the male tab disconnect and lockwasher provided with the screw located 2-5/8 inches from the right side of this strap. Be careful that the nut is not dropped into the computer. Do not use the screws at the upper edge for they are required for mounting the card cage. Be sure that it is mounted so that the tab is turned away from the circuit card when plugged in.

The CPU board should now be repositioned back down to its normal place. It should be noted that there are some plastic card guides and the board should be within them.

If the wire with the female "disconnect" does not have 1/4" insulation removed from the other end then do so with wire strippers. The wire should now be soldered to one of the ground lugs on the heat sink assembly which is located with the power supply in the right rear of the H/Z-89. This is the same point that was used as the connection for the negative (common) lead of the voltmeter in the above test in 3.1.1. The ground lugs are located close to the middle of the aluminum plate (not the one on the CPU logic card) which makes up the heat sink assembly and has four black wires connected to them. The lugs are mounted to and electrically connected to the heat sink assembly plate. This is the power supply ground point.

Connect the negative (common) lead of a voltmeter to the aluminum strap across the top of the CPU logic board. The other lead should be connected to the female disconnect of the wire just soldered. With the voltmeter on a 20 or 25 volt DC scale, plug the computer in and turn on.

If a voltage in excess of 1 volt DC is present, then the connection to the power supply lug is incorrect. Recheck the solder connection for correct location and re-test as above. If a voltage below 1 volt DC cannot be achieved, contact NOGDS.

If a voltage below 1 volt DC (usually less than 0.15 VDC) is

present, then plug the male tab disconnect and the female disconnect (at the end of the wire just soldered) together creating the additional ground path.

At this point the power modifications are complete and you may proceed to section 3.2.

3.2 Interfacing and Initial Adjustments

Turn off your H/Z-89 system and disconnect the line cord. While facing the screen, locate the three circuit board slots on the right side of the CPU logic board. The HA-89-3 will be placed in one of those right-hand slots and must not be placed in one of the three left-hand slots. The HA-88-3 three port serial I/O card, if present, will be one of those on the right-hand side. If your system has an HA-88-3 card, it should be removed and placed on the side for the initial checkout under the diagnostic program.

Remove the two screws holding the mounting bracket for the three I/O cards on the right side and put the bracket on the side. The HA-89-3 may be placed in either the left or middle position of the three. The board should be inserted in one of those two slots keeping in mind the previous warning given above about misalignment. Components should face to the middle of the H/Z-89 (toward the neck of the CRT). Be absolutely sure that the pins on the CPU logic board match the sockets of the HA-89-3 and that it is not plugged in one pin too high or low (misalignment warning above). Only after you are certain that it is plugged in correctly, plug in the line cord and turn on power.

Now insert the distribution diskette along with a bootable HDOS disk, bring up the system and run H893DIAG. An initial test of the bus interface, PSG and VDP RAM interface will be made. If no errors are reported proceed with the display testing; otherwise, contact NOGDS.

Under the assumption that you have one disk controller card and a serial I/O card, the HA-89-3 can become the third card. The HA-89-3 is shipped with J9 set in the "S0" position which corresponds to I/O address 320Q (Hex D0). This implies that no other device in the system can respond to that address. If the HA-88-3 three port serial card is installed, the middle 8250 will respond to address 320Q also. There are two solutions for this problem. First, the HA-88-3 serial I/O card can be removed when the HA-89-3 is used. This implies that the HA-88-3 will be installed and removed fairly frequently. If the HA-89-3 is swapped with the HA-88-3, the probably of a card being inserted improperly is increased. The only time this method should be used is in systems which have all three I/O slots in use on the H/Z-89 CPU logic board. Alternately, the 8250 (U603) 40-pin chip, located in the middle of the serial I/O board, may be removed and placed in protective foam. Remember to used standard procedures for handling MOS chips since they are sensitive to static charges. The serial I/O board may then be placed back into the H/Z-89 along with the HA-89-3. Since U603

has been removed, the HA-88-3 will not respond to address 320Q.

The HA-89-3 can be jumpered to respond to other addresses. The other positions and the corresponding I/O addresses are listed below:

Jumper Position	I/O address	HA-88-3 Chip
S0	320Q	U603 (center)
S1	330Q	U604 (lower)
LP	340Q	U602 (upper)
CA	do not use	do not use

The above I/O addresses are only for standard I/O decoder ROMs. Since the HA-89-3 requires a group of eight contiguous I/O addresses, the CA position cannot be used with the standard I/O decoder ROMs. However, with non-standard ROMs, this position may be useable. Normally the disk controllers or the cassette board obtain device select from the CA position. Should it be necessary to make the HA-89-3 respond to "S1" or "LP", the same procedure may be used except that the corresponding 8250 (U604 or U602) should be removed instead. Also, the device support routines will have to be modified to respond to addresses other than 320Q. This procedure is documented in section 6.3 (Board Address).

The mounting bracket may now be replaced with the two screws previously placed on the side. Be sure that the cards are not misaligned with respect to the pin connections and that they are correctly in the card guides.

The HA-89-3 comes adjusted to NTSC standards and should be usable with a color monitor without needing adjustment. If adjustments are needed see section 6.2 for procedures. Two types of signal cables are provided with the board, one for video and one for audio and other outputs. The cable with a male phono plug is the video cable. The end of this cable with the three pin connector plugs into the board at the VID connector. The other end of this cable connects to the VIDEO IN of a Heath GDZ-1320 (or equivalent) color monitor. Many RF modulators and video tape recorders also use this type of composite video interface connector.

The cable with the three pin connector is the video cable. The connector is not polarized so it does not matter in which direction it is plugged into the VID connector. The cable with the ribbon cable connector is the audio cable. The end of this cable with the connector plugs into the board at the AUD connector. The other end of this cable must be soldered to a miniature male phone plug and connected to the AUDIO IN if used with the GDZ-1320 (or equivalent) color monitor. Many video tape recorders also use this type of audio interface connector.

If the cables supplied with the HA-89-3 will not interface, refer to section 4.7² for information on how to interface to the VID or section 4.7¹ for the AUD connector.

Refer to section 5.1 and now run the diagnostic program to completion in order to check out the HA-89-3 completely. If any adjustments are required, refer to section 6 for more information.

4 Interfacing Information

This section describes the interfacing requirement of the joystick, counter/timer, composite video output, and audio output connectors. For each of the following connectors, the cables can be lead out of the H/Z-89 via the opening for an IEEE-488 interface in the rear of the computer.

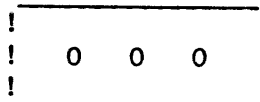
4.1 Connector Pin Numbering

There are two basic types of interface connectors, a three pin VID connector for video output and a 10 pin type used for all other interfaces. Below is a table specifying the interface connector name, jack number and number of pins.

<u>NAME</u>	<u>JACK</u>	<u>PINS</u>
JOY1	J1	10
JOY2	J2	10
JOY3	J3	10
JOY4	J4	10
AUD	J5	10
CTC	J6	10
VID	J7	3

The three pin VID connector is shown below with the pins labeled according to their function.

VID Connector

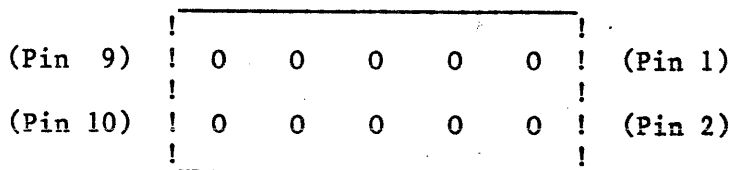


G V G
n i n
d d d
e
o

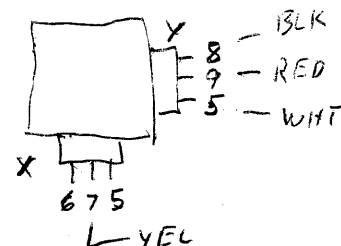
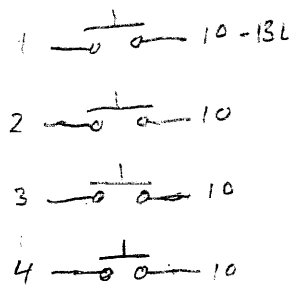
Ten pin connectors are used for all other connections listed above. These are intended to be used with ribbon cable connectors. The pin numbers for these are as shown below:

10 Pin Connector

(Top, component side)



(View from rear of the board)



4.2 Joystick Interfacing Information

This section describes how to interface joysticks to the HA-89-3. Each joystick connects to a 10 pin male connector. The following is a detailed table of the pins on the joystick connector:

	<u>Joy 1</u>	<u>Joy 2</u>	<u>Joy 3</u>	<u>Joy 4</u>
Pin 1	0	2	4	6 (P/P A)
Pin 2	1	3	5	7 (P/P A)
Pin 3	0	2	4	6 (P/P B)
Pin 4	1	3	5	7 (P/P B)
Pin 5	Vref	Vref	Vref	Vref
Pin 6	Gnd	Gnd	Gnd	Gnd
Pin 7	0 X-AXIS	2	4	6 (A/D In)
Pin 8	Gnd	Gnd	Gnd	Gnd
Pin 9	1 Y-AXIS	3	5	7 (A/D In)
Pin 10	Gnd	Gnd	Gnd	Gnd

In the above "P/P" stands for parallel port and "A/D In" stands for analog to digital input. The interfacing information is broken into two parts: Interfacing to the Parallel Ports, and Interfacing to the Analog Ports.

4.2.1 Interfacing to the Parallel Ports

Note that two bits from each of the two parallel ports are brought out to each of the joystick connectors. If one port is enabled for input and one for output (see 7.2.1.5), each joystick connector will have 2 bits of parallel input for push button switches, etc., and 2-bits of parallel output for LED's, etc. When one port is enabled for output and the other for input, it is recommended that port B be used for output and port A be used for input. This is suggested as a programming standard and there is no hardware restriction for this. It is possible to program both ports for input or both ports for output, if desired.

4.2.1.1 Parallel Input Interface

The PSG contains internal pull-up resistors on the parallel port inputs. Therefore the recommended method of interfacing to an input port is to use a normally open push button switch. Connect the input bit to one lead and ground to the other. Ground is available at pins 6,8 and 10 of the connector.

4.2.1.2 Parallel Output Interface

The output drive of a parallel port is one TTL load. This is not enough to directly drive a relay or LED, etc. The parallel outputs must be buffered before they can be used to drive an external device. Gnd is available on the connector at pins 6,8 and 10 but Vcc must be supplied externally. Do not use Vref to power external devices.

4.2.2 Interfacing to the Analog Input Ports

The analog input voltage must be between Vref and Gnd. To accomplish this a pot (potentiometer) may be used as a voltage divider. One end of the pot should be connected to Vref and the other to Gnd. The wiper (center lead) of the pot should be connected to one of the analog channels. A 10K ohm pot with a linear taper is recommended for this application.

A joystick requires two pots - One for the X-axis, and one for the Y-axis. When a joystick is used, the following configuration is recommended:

- 1) The X-axis pot should be connected to the even numbered analog channel.
- 2) The Y-axis pot should be connected to the odd numbered analog channel.
- 3) The X-axis pot should be wired so that as the joystick is moved right, the wiper of the pot approaches Vref. Conversely, as the joystick is moved to the left, the wiper of the pot approaches Gnd.
- 4) The Y-axis pot should be wired so that as the joystick is moved up, the wiper of the pot approaches Gnd. Conversely, as the joystick is moved down, the wiper of the pot approaches Vref.

4.3 Composite Video Interface

The 3 pin connector labeled VID is the composite video output of the HA-89-3. The output impedance of the video amplifier is 75 Ohms. The two outer pins on the VID connector are both ground. The middle connector pin is the composite video output signal. Because

the outer pins are both ground, the connector is not polarized and hence may be plugged in either way.

If the cable supplied with the HA-89-3 does not match the video input connector of the display device, an adapter cable with a female phono socket on one end and the appropriate connector for the display device on the other end should be obtained.

4.4 Audio and DAC Interface

The 10 pin connector labeled AUD supplies the audio and DAC outputs of the HA-89-3. The audio output from the PSG (Programmable Sound Generator), PSS (Phoneme Speech Synthesizer) and mixer are each a 1 Volt peak-to-peak AC coupled signal. The mixer output is of course an audio mixing of both the PSG and PSS outputs. Should it be required to connect the PSG or PSS independently, those outputs are also available on the AUD connector see the table of assignments below.

The two digital to analog converters (DACs) will each provide an output range of ground to +5.00 volts. The following is a list of the pin assignments for the AUD connector:

<u>AUD Connector</u>	
Pin 1	DAC 1 Output
Pin 2	Gnd
Pin 3	DAC 0 Output
Pin 4	Gnd
Pin 5	PSG Output
Pin 6	Gnd
Pin 7	Mixer Output
Pin 8	Gnd
Pin 9	PSS Output
Pin 10	Gnd

4.5 Counter Timer Interface

There is a 10 pin connector on the rear of the HA-89-3 board marked CTC. This is the I/O interface connector for the Counter Timer Chip (CTC). Clock, Gate, Out, Vcc and Gnd are provided on this connector with the exception of the Gate input for counter 0. Also NOTE that CLK 0 pin 1 of the CTC connector is an output set at 1.79MHz, not an input. This is because counter 0 has been set-up as a timer with a clock rate of 1.7898MHz and may not be used with an external clock input. The other two 16 bit counters are available to be used with an external input in either the CLK or GATE inputs. It should be noted that the GATE inputs have pull-up resistors so a ground connection will apply a low and no connection will assert a high. The pin assignments for this connector are shown below:

CTC Connector

Pin 1	1.79MHz Output
Pin 2	OUT 0
Pin 3	CLK 2
Pin 4	Vcc
Pin 5	GATE 2
Pin 6	OUT 2
Pin 7	CLK 1
Pin 8	Gnd
Pin 9	GATE 1
Pin 10	OUT 1

5 Distribution Diskette

The distribution software for the HA-89-3 is on a 5¼" or 8" floppy diskette. In addition to the software support routines documented in section 4, this diskette has both diagnostic and demonstration programs for the HA-89-3.

5.1 Diagnostic Software (H893DIAG)

The diagnostic program is H893DIAG. Both the source and the ABS files have been included. H893DIAG performs the following tests:

- 1) The first test performed by H893DIAG consists of a memory test on the VRAM. This test requires about one minute and, if no errors are detected, it will print the message "No VRAM failures detected.". If the test fails, then it will list the board location of the failing VRAMs by "Fn" designation. Memory chips are identified by location F1 thru F8. The chip at the top of the board (near resistor R28) is F1 and the one at the bottom (near capacitor C40) is F8.
- 2) After the memory diagnostic, H893DIAG paints color bars on the color display. The order of these color bars is:
 - 1) White
 - 2) Gray
 - 3) Dark Blue
 - 4) Light Blue
 - 5) Cyan
 - 6) Dark Green
 - 7) Medium Green
 - 8) Light Green
 - 9) Dark Yellow
 - 10) Light Yellow
 - 11) Dark Red
 - 12) Medium Red
 - 13) Light Red
 - 14) Magenta

The display should be examined for the proper color sequence.

- 3) After the color bars are displayed, H893DIAG produces four sound effects:
 - A) The first sound effect is a laser. The sound is a frequency sweep effect on channel C of the PSG (Programmable Sound Generator).
 - B) The second sound effect is a whistling bomb. The whistling effect is a frequency sweep effect on channel B. The

explosion is a noise effect under the control of the envelope generator. All three channels participate in the explosion effect.

- C) The third sound effect is a wolf whistle. The whistling is produced on channel A. Channel B is used in this effect to produce white noise associated air movement through the lips during the whistle.
- D) The fourth sound effect is a racing car changing gears. The effect of increasing engine RPM is produced on channel A, while a constant, high frequency engine whine effect is produced on channel B.

The sound effects should be listened to carefully for defects.

After the sound effects are produced, H893DIAG will display 16 squares on the top of the display. The top row of squares are the bits from parallel port A, and the bottom row are from parallel port B. The leftmost square in each row corresponds to the high order bit. A white square indicates that the corresponding bit is ON (a high input), and a black square indicates that it is OFF (a low input). Since the PSG has pullup resistors on the parallel ports, the dots should be displayed as white. The numbers 1 through 4 are also displayed. The position of these numbers is determined by the analog inputs on each joystick connector (JOY1 thru JOY4). The "X" position is determined by the voltage on the even numbered analog channel, and the "Y" position is determined by the voltage on the odd numbered analog channel. If no joysticks are available, the numbers can be moved randomly by touching the joystick connectors. The electrical noise produced by touching the connectors should be enough to randomly move the numbers on the screen.

To end the diagnostic, type two control Z's (CTRL + Z). This will terminate execution and return to command level.

5.2 Colors

There are 15 programs on the distribution diskette to paint the face of the color display a solid color. The following is a table of colors and programs:

<u>Color</u>	<u>Program</u>	<u>Source</u>
White	WHITE.ABS	WHITE.ASM
Gray	GRAY.ABS	GRAY.ASM
Black	BLACK.ABS	BLACK.ASM
Dark Blue	DBLUE.ABS	DBLUE.ASM
Light Blue	LBLUE.ABS	LBLUE.ASM
Cyan	CYAN.ABS	CYAN.ASM
Dark Green	DGREEN.ABS	DGREEN.ASM
Medium Green	MGREEN.ABS	MGREEN.ASM
Light Green	LGREEN.ABS	LGREEN.ASM
Dark Yellow	DYELLOW.ABS	DYELLOW.ASM
Light Yellow	LYELLOW.ABS	LYELLOW.ASM
Dark Red	DRED.ABS	DRED.ASM
Medium Red	MRED.ABS	MRED.ASM
Light Red	LRED.ABS	LRED.ASM
Magenta	MAGENTA.ABS	MAGENTA.ASM

To paint the face of the color display to one of the above colors, execute the program whose name appears to the right of the color.

5.3 Color Bars

There are two programs on the distribution diskette to paint color bars on the color display. The program NTSCBARS paints color bars in the NTSC order. The program BARS paints color bars so that the color phase angle difference between colors is minimized.

5.3.1 NTSCBARS (NTSC Order of Color Bars)

This program paints color bars in the NTSC order. The order of the color bars is given in the following table:

- 1) White
- 2) Gray
- 3) Dark Yellow
- 4) Light Yellow
- 5) Cyan
- 6) Dark Green
- 7) Medium Green
- 8) Light Green
- 9) Magenta
- 10) Dark Red
- 11) Medium Red
- 12) Light Red
- 13) Dark Blue
- 14) Light Blue

5.3.2 BARS (Phase Order of Color Bars)

This program paints colors bars on the color display so that the phase angle difference between colors is minimized. The order of the color bars is given in the following table:

- 1) White
- 2) Gray
- 3) Dark Blue
- 4) Light Blue
- 5) Cyan
- 6) Dark Green
- 7) Medium Green
- 8) Light Green
- 9) Dark Yellow
- 10) Light Yellow
- 11) Dark Red
- 12) Medium Red
- 13) Light Red
- 14) Magenta

6 Adjustments

Adjustments to the HA-89-3 are described in this section. All boards are shipped fully tested and preadjusted. Should it be necessary to make any adjustments to the board, follow the recommendations in this section.

6.1 Adjusting the Composite Video Output

The video output of the HA-89-3 has been preadjusted to NTSC standards and should be compatible with such inputs. This section covers alteration of these settings. There are two adjustments for the composite video output. Both of the adjustments are described below:

6.1.1 Video Offset

Trim pot P1 is labeled VIDEO OFFSET in the logic diagrams. This trim pot controls the DC component of the composite video signal. Most composite video inputs (including the GDZ-1320 color monitor) consist of 75 ohm AC coupled inputs. For these applications this trim pot should be adjusted so that the blanking level corresponds to zero volts. This corresponds, in most cases, to setting P1 fully clockwise, then counterclockwise 12 turns.

6.1.2 Video Level

Trim pot P2 is labeled VIDEO LEVEL in the logic diagram. In most applications (including the GDZ-1320 monitor) this trim pot should be adjusted for 1 volt peak to peak. Fine adjustments should be made to P1 for the best overall picture as P2 interacts with it. This corresponds, in most cases, to setting P2 fully counterclockwise, then clockwise 5 turns.

6.2 Mix Level

Trim pot P3 is labeled MIX LEVEL in the logic diagram. This pot adjusts the gain of the mixer which combines the output of the PSG and the PSS. In most instances this pot should be adjusted to the desired level. This corresponds to setting the pot fully clockwise, then counterclockwise 8 turns.

6.3 Board Address

The board address is determined by the placement of the shunt on J9. The set of 8 pins at J9 is termed BOARD ADR in the logic diagram. Within this set there are four pairs of pins labeled S0, S1, LP, and CA. Jumpering one and only one of these pairs together

determines the address that the board will respond to. The pins should be jumpered together via the shunt that has been provided with the board. The pair should be used which corresponds to a device that is not in use. The software is shipped expecting the board to respond to 320Q (J9 set to S0, see table in section 3.2), but it can be reassembled to use another choice. To change the software routines to respond to an address other than 320Q, only one line in the file HA893DEF.ACM need be changed. Modify the equate on the line labeled HA893L to the desired I/O address. There are restrictions to the use of "CA", see section 3.2.

6.4 Interrupts

The HA-89-3 provides for interrupts from the PIC (Programable Interrupt Controller). The board is shipped with none of the interrupt vectors (J10) jumpered. A shunt may be placed across one of the pairs marked I3, I4, or I5 depending on the interrupt vector desired. It must be remembered that, if this is done, when interrupts occur, the software must be prepared to handle them.

7 Programming the HA-89-3

This section describes how to program the major components of the HA-89-3 using the support software on the distribution diskette. The routine HA893DEF.ACM must be included when any of the routines, discussed below, are assembled. The data sheets for each of the major components of the HA-89-3 are located in the appendices of this manual. Before reading how the support software works for a given component, it is advisable to read the data sheet for that device.

7.1 The TMS-9918A Video Display Processor (VDP)

The basic support software for the VDP is contained in two files on the distribution diskette: VDPDEF.ACM and VDPIO.ACM. The file VDPDEF.ACM contains the definitions of the bits in the registers of the VDP. The file VDPIO.ACM contains I/O routines to maintain the VDP's registers.

There are three other files on the distribution diskette which contain software aids for programming the VDP. The file MCXYSUBS.ACM contains software support for using the multicolor mode to produce "X-Y" plots. The file SPRITE.ACM contains software routines for sprite handling. The file VDPMOVES.ACM contains routines to move blocks of data to and from VRAM.

Unless otherwise noted, values requiring 8 bits or less are passed in register A. Values requiring more than 8 bits are passed in register HL. All registers are preserved except for those used to return data. Since the VDP has 16K of VRAM, only the lower 14 bits are used when passing VRAM addresses.

7.1.1 Primitive VDP I/O Routines

The file VDPIO.ACM contains the primitive functions for doing I/O on the VDP. Each routine is described below:

7.1.1.1 VP.SRA (Set VDP VRAM Read Address)

This is the primitive routine used to set the VDP's VRAM Read Address Register. The 14-bit VRAM read address is placed in register HL, and then VP.SRA is called. Subsequent calls to VP.RDV (Read VRAM) will start at the address passed in register HL.

To set the VRAM Read Address Register to location 8190, the following call to VP.SRA could be used:

```
LXI    HL,8190
CALL   VP.SRA
```

7.1.1.2 VP.RDV (Read VRAM)

This routine reads data from the VRAM at the location in the Read Address Register (see 7.1.1.1 VP.SRA to set the Read Address Register). The Read Address Register is autoincremented by the read. The VRAM data is returned in register A.

The following code could be used to place the contents of VRAM location 2000 into register B and the contents of VRAM location 2001 into register C:

```
LXI    HL,2000
CALL   VP.SRA
CALL   VP.RDV
MOV    B,A
CALL   VP.RDV
MOV    C,A
```

7.1.1.3 VP.RVD (Read VRAM Direct)

This routine is a combination of VP.SRA and VP.RDV. The 14-bit VRAM read address is placed in register HL and the data from VRAM location HL is returned in register A. The Read Address Register is autoincremented.

The following code can also be used to place the contents of VRAM location 2000 in register B, and the contents of location 2001 in register C:

```
LXI    HL,2000
CALL   VP.RVD
MOV    B,A
CALL   VP.RDV
MOV    C,A
```

7.1.1.4 VP.SWA (Set VDP VRAM Write Address)

This routine sets the VDP VRAM Write Address Register. The 14-bit read address is passed in register HL. Subsequent calls to VP.WRV (Write VRAM) will start at the address in register HL.

To set the VRAM Write Address Register to location 8190, the following call to VP.SWA could be used:

```
LXI    HL,8190
CALL   VP.SWA
```

7.1.1.5 VP.WRV (Write VRAM)

This routine writes the data in register A to the VRAM location in the Write Address Register (see 7.1.1.4 VP.SWA to set the Write Address Register). The Write Address Register is autoincremented by the write.

The following code could be used to write the contents of register B to VRAM location 2000 and the contents of register C to VRAM location 2001:

```
LXI    HL,2000
CALL   VP.SWA
MOV    A,B
CALL   VP.WRV
MOV    A,C
CALL   VP.WRV
```

7.1.1.6 VP.WVD (Write VRAM Direct)

This routine is a combination of VP.SWA and VP.WRV. The VRAM address to be written is placed in register HL and the data to be written is placed in register A. The Write Address Register is autoincremented by this operation.

The following code can also be used to write the contents of register B to VRAM location 2000 and the contents of register C to VRAM location 2001:

```
LXI    HL,2000
MOV    A,B
CALL   VP.WVD
MOV    A,C
CALL   VP.WRV
```

7.1.1.7 VP.WRR (Write VDP Register)

This routine is used to write the write only VDP registers. The VDP register number to be written is passed in register L, and the value to be written is passed in register A. This is an internal routine and should not be used. The routines described in 7.1.2 should be used to maintain the VDP registers.

7.1.2 VDP Register Maintenance Routines

The file VDPIO.ACM contains the software routines to maintain the nine VDP registers, and routines to read and write the 16K of VDP RAM (VRAM). These routines use the register pair HL to pass values requiring more than 8 bits and register A to pass 8 bit values.

The file VDPDEF.ACM contains the bit names for the bits in the registers. Where applicable, the bit name from VDPDEF.ACM is given. In order to be consistent with the data sheet for the VDP, the description of the registers uses bit 7 for a low order bit, and bit 0 for a high order bit. Each of the VDP registers, along with the software support for the register, is described below:

7.1.2.1 Registers 0 and 1 (Option Control)

Registers 0 and 1 contain the option bits used to enable and disable various VDP features and modes. A description of each of the option bits follows:

Register 0:

BIT 7 (VP.NEV) OFF: Disable external video
(VP.EV) ON: Enable external video (not supported on HA-89-3)

Register 1:

BIT 7 (VP.M0) OFF: Magnification 0 sprites
(VP.M1) ON: Magnification 1 sprites

BIT 6 (VP.S0) OFF: Size 0 sprites
(VP.S1) ON: Size 1 sprites

BIT 4 (VP.MCM) OFF: Multicolor mode not selected
ON: Enable multicolor mode

BIT 3 (VP.TM) OFF: Text mode not selected
ON: Enable text mode

BIT 2 (VP.DI) OFF: Disable interrupts
(VP.EI) ON: Enable interrupts

BIT 1 (VP.DDP) OFF: Disable (blank out) display
(VP.EDP) ON: Enable display

BIT 0 (VP.4K) OFF: 4K dynamic VRAMs (do not use this)
(VP.16K) ON: 16K dynamic VRAMs (use this)

Note that it is not legal to select both the multicolor mode and the text mode together (VP.MCM+VP.TM). When neither the multicolor mode nor text mode is selected, the pattern mode (VP.PM) is selected.

To load the option control registers, place the 16-bit option mask in register HL and call the routine VP.SOP. For example, to select the multicolor mode with size 0, magnification 0 sprites with the display active, the following code could be used:

```

LXI    HL,VP.NEV+VP.16K+VP.EDP+VP.DI+VP.MCM+VP.SO+VP.MO
CALL   VP.SOP

```

The VDP register maintenance routine VP.SOP saves the most recently set options in the variable VP.OPT.

7.1.2.2 Register 2 (Pattern Name Table Base Address)

The register maintenance routine for the Pattern Name Table Register is VP.SPN. The 14-bit VRAM address of the Pattern Name Table is placed in register HL. The Pattern Name Table address is stored in the variable VP.PNT. (VP.PNT should contain the most recent Pattern Name Table address.) The upper 4 bits of the 14-bit Pattern Name Table Address are then stored in the Pattern Name Table Register of the VDP. Since the VDP does not store the low order 10 bits of the Pattern Name Table address, this address must be evenly divisible by 1024 (the bottom ten bits must all be 0's). VP.SPN does not verify this.

To set the VRAM address of the Pattern Name Table to 2048, the following calling sequence could be used:

```

LXI    HL,2048
CALL   VP.SPN

```

After this call, the variable VP.PNT will contain the value 2048, and the value 2 (2048/1024) has been stored in the VDP's Pattern Name Table Register.

7.1.2.3 Register 3 (Color Generator Table Base Address)

The register maintenance routine for the Color Generator Table Register is VP.SCG. The 14-bit VRAM address of the Color Generator Table is placed in register HL. The Color Generator Table Address is stored in the variable VP.CGT. VP.PNT should contain the most recent Color Generator Table address. The upper 8 bits of the 14-bit Color Generator Table address are then stored in the Color Generator Table Address Register of the VDP. Since the VDP does not store the low order 6 bits of the Color Generator Table, this address must be evenly divisible by 64 (the bottom 6 bits must all be 0's). VP.SCG does not verify this.

To set the VRAM address of the Color Generator Table to be 256, the following calling sequence could be used:

```
LXI    HL,256
CALL   VP.SCG
```

After this call, the variable VP.CGT will contain the value 256, and the value 4 (256/64) has been stored in the VDP's Color Generator Table Register.

7.1.2.4 Register 4 (Pattern Generator Table Base Address)

The register maintenance routine for the Pattern Generator Table Register is VP.SPG. The 14-bit VRAM address of the Pattern Generator Table is placed in register HL. The Pattern Generator Table address is stored in the variable VP.PGT. (VP.PGT should contain the most recent Pattern Generator Table address.) The upper 3 bits of the 14-bit Pattern Generator Table Address are then stored in the Pattern Generator Table Register of the VDP. Since the VDP does not store the low order 11 bits of the Pattern Generator Table address, this address must be evenly divisible by 2048 (the bottom 11 bits must all be 0's). VP.SPG does not verify this.

To set the VRAM address of the Pattern Generator Table to be 6144, the following calling sequence could be used:

```
LXI    HL,6144
CALL   VP.SPG
```

After this call, the variable VP.PGT will contain the value 6144, and the value 3 (6144/2048) has been stored in the VDP's Pattern Generator Table Register.

7.1.2.5 Register 5 (Sprite Name Table Base Address)

The register maintenance routine for the Sprite Name Table Register is VP.SSN. The 14-bit VRAM address of the Sprite Name Table is placed in register HL. The Sprite Name Table address is stored in the variable VP.SNT. (VP.SNT should contain the most recent Sprite Name Table address.) The upper 7 bits of the 14-bit Sprite Name Table Address are then stored in the Sprite Name Table Register of the VDP. Since the VDP does not store the low order 7 bits of the Sprite Name Table address, this address must be evenly divisible by 128 (the bottom 7 bits must all be 0's). VP.SSN does not verify this.

To set the VRAM address of the Sprite Name Table to be 128, the following calling sequence could be used:

```
LXI    HL,128
CALL   VP.SSN
```

After this call, the variable VP.SNT will contain the value 128, and the value 1 (128/128) has been stored in the VDP's Sprite Name Table Register.

7.1.2.6 Register 6 (Sprite Pattern Generator Table Base Address)

The register maintenance routine for the Sprite Pattern Generator Table Register is VP.SSG. The 14-bit VRAM address of the Sprite Pattern Generator Table is placed in register HL. The Sprite Pattern Generator Table address is stored in the variable VP.SGT. VP.SGT should contain the most recent Sprite Pattern Generator Table address. The upper 3 bits of the 14-bit Sprite Pattern Generator Table Address are then stored in the Sprite Pattern Generator Table Register of the VDP. Since the VDP does not store the low order 11 bits of the Sprite Pattern Generator Table address, this address must be evenly divisible by 2048 (the bottom 11 bits must all be 0's). VP.SSG does not verify this.

To set the VRAM address of the Sprite Pattern Generator Table to be 4096, the following calling sequence could be used:

```
LXI    HL,4096
CALL   VP.SSG
```

After this call, the variable VP.SGT will contain the value 4096, and the value 2 (4096/2048) has been stored in the VDP's Sprite Pattern Generator Table Register.

7.1.2.7 Register 7 (Text/Backdrop Color)

The register maintenance routine for the Text/Backdrop Register is VP.STB. The contents of this register depend on the operating mode of the VDP. If the VDP is in the text mode, then the left nibble of register 7 contains the color of ON pattern bits, and the right nibble contains the color of OFF pattern bits. In all other modes, the left nibble is not used, and the right nibble contains the color of the backdrop.

The following is a table of colors and their names from VDPDEF.ACM:

Color	Name	Value
Transparent	VC.CLR	0
Black	VC.BLK	1
Medium Green	VC.MGR	2
Light Green	VC.LGR	3
Dark Blue	VC.DBL	4
Light Blue	VC.LBL	5
Dark Red	VC.DRD	6
Cyan	VC.CYN	7
Medium Red	VC.MRD	8
Light Red	VC.LRD	9
Dark Yellow	VC.DYL	10
Light Yellow	VC.LYL	11
Dark Green	VC.DGR	12
Magenta	VC.MAG	13
Gray	VC.GRY	14
White	VC.WHT	15

To place a color from the above table in the left nibble, multiply the name from the above table by VC.LFT. In the text mode, the ON bits could be set to black, and the OFF bits could be set to light blue by making the following call to VP.STB:

```
MVI    A,VC.BLK*VC.LFT+VC.LBL
CALL   VP.STB
```

The backdrop color could be set to dark green by making the following call to VP.STB:

```
MVI    A,VC.DGR
CALL   VP.STB
```

VP.STB maintains the most recent text/backdrop color in the variable VP.TBC.

7.1.2.8 Status Register (Read Only)

The VDP contains one read only status register. To read the status register, use the routine VP.RDR. The contents of the status register are returned in register A. The contents of this register are:

7.1.2.8.1 Interrupt Flag (Bit 0)

This bit is set if the VDP is requesting an interrupt. This bit is automatically reset by reading the status register. The mask value from VDPDEF.ACM is VP.IF.

7.1.2.8.2 Fifth Sprite Flag (Bit 1)

The VDP cannot display more than four sprites on any one raster scan line. If this rule is violated, this bit is set and the fifth sprite number is also placed in bits 3-7. The mask value from VDPDEF.ACM is VP.5S.

7.1.2.8.3 Coincidence Flag (Bit 2)

When two or more sprites have one or more overlapping pixels, this bit is turned on. The mask value from VDPDEF.ACM is VP.C.

7.1.2.8.4 Fifth Sprite Number (Bits 3-7)

When the Fifth Sprite Flag is set, these bits will contain the number of the first sprite not displayed (i.e., the fifth sprite on the same horizontal raster scan). The mask value from VDPDEF.ACM is VP.FSN.

7.1.3 "X-Y" Routines for the Multicolor Mode

The file MCXYSUBS.ACM contains two routines to aid in producing "X-Y" plots. The "X-Y" routines divide the screen into 48 rows by 64 columns of display. Each of these 3072 points can be any of the 16 colors. Transparent points will appear to be the same color as the backdrop color. Before these routines can be used the following registers must have been set:

- 1) The multicolor mode must be enabled using the routine VP.SOP and the option VP.MCM (see 7.1.2.1).
- 2) The pattern name table base address register must have been loaded using the routine VP.SPN (see 7.1.2.2).
- 3) The pattern generator table base address register must have been loaded using the routine VP.SPG (see 7.1.2.4).

MC.XYI and MC.XY make use of the Pattern Name Table pointed to by VP.PNT and the Pattern Generator Table address pointed to by VP.PGT. VP.SPN (see 7.1.2.2) will set VP.PNT to the Pattern Name Table address most recently loaded to the VDP Pattern Name Table Register, and VP.SPG (see 7.1.2.4) will set VP.PGT to the Pattern Generator Table address most recently loaded to the VDP Pattern Generator Table Register.

MC.XYI and MC.XY are described below:

7.1.3.1 MC.XYI (Initialize "X-Y" Plotting)

This routine initializes the Pattern Name Table and the Pattern Generator Table for "X-Y" plotting. The screen is initialized to black. The routine uses a standard 768 Pattern Name Table and a 1536 byte Pattern Generator Table. This routine requires no data to be passed in the registers.

7.1.3.2 MC.XY (Plot an "X-Y" Point)

This routine changes the dot at the (X,Y) point specified in register (H,L) to the color specified by register A. The "X" value is passed in register H. The value contained in register H must be between 0 and 47 inclusive. The "Y" value is passed in register L. The value contained in register L must be between 0 and 63 inclusive. The color of the point is passed in the low order nibble of register A. The value passed must be one of the colors described in 7.1.2.7.

7.1.4 Sprite Processing Routines

The routines described in this section aid the user in programming sprites. The routines are found in the file SPRITE.ACM. These routines provide an easy to use mechanism to read and write the sprite attributes. The programmer need not be concerned with the location of the sprite in VRAM. All references to the sprites are via the sprite number. The routines will automatically convert the sprite number to a VRAM address and read or write the sprite attributes.

The following routines make use of the Sprite Name Table pointed to by VP.SNT, and the Sprite Pattern Generator Table pointed to by VP.SGT. VP.SSN (see 7.1.2.5) will set VP.SNT to point to the Sprite Name Table most recently loaded to the VDP Sprite Name Table Register, and VP.SSG (see 7.1.2.6) will set VP.SGT to point to the Sprite Pattern Generator Table most recently loaded to the VDP Sprite Pattern Generator Table Register.

A sprite attribute consists of four bytes. The first byte is the vertical ("Y") position of the sprite. The second byte is the horizontal ("X") position of the sprite. The third byte is the sprite name. For size 1 sprites, the sprite name must be evenly divisible by four. The last byte contains the sprite color in the low order four bits (see color codes in 7.1.2.7), and an early clock bit in the high order bit. When the early clock bit is set, the horizontal ("X") value is offset 32 pixels to the left to allow the sprite to bleed in from the left edge of the display.

7.1.4.1 VP.SAA (Get Sprite Attribute Address)

This routine will return the base VRAM address of the sprite attribute block for the sprite number passed in register A. The base address of the sprite attribute block is returned in register HL.

7.1.4.2 VP.SPA (Get Sprite Pattern Address)

This routine will return the base VRAM address of the sprite pattern generator block for the sprite number passed in register A. The base address of the sprite pattern generator block is returned in register HL.

7.1.4.3 VP.GSC (Get Sprite Coordinates)

This routine will return the coordinates of the sprite number passed in register A. The horizontal ("X") value is returned in register H, and the vertical ("Y") value is returned in register L.

7.1.4.4 VP.SSC (Set Sprite Coordinates)

This routine will set the coordinates of the sprite number passed in register A. The horizontal ("X") value is passed in register H, and the vertical ("Y") value is passed in register L.

7.1.4.5 VP.GSA (Get Sprite Attributes)

This routine will return all the attributes of the sprite number passed in register A. The horizontal ("X") value is returned in register H, the vertical ("Y") value is returned in register L, the sprite name is returned in register D, and the sprite color/early clock bit is returned in register E.

7.1.4.6 VP.SSA (Set Sprite Attributes)

This routine will set the attributes of the sprite number passed in register A. This horizontal ("X") value is passed in register H, the vertical ("Y") value is passed in register L, the sprite name is passed in register D, and the sprite color/early clock bit is passed in register E.

7.1.5 VRAM Maintenance Routines

The file VDPMOVES.ACM contains two routines to facilitate moving blocks of data between VRAM and memory. The routine VP.MTV is used to move data to VRAM, and the routine VP.VTM is used to transfer data from VRAM to memory. The most common use of these routines is to block load VRAM table from memory.

7.1.5.1 VP.MTV (Move from Memory to VRAM)

This routine is used to move data from memory to VRAM. The starting VRAM address is passed in register HL. The starting memory address is passed in register DE. The number of bytes to be transferred is passed in register BC.

7.1.5.2 VP.VTM (Move from VRAM to memory)

This routine is used to move data from VRAM to memory. The starting VRAM address is passed in register HL. The starting memory address is passed in register DE. The number of bytes to be transferred is passed in register BC.

7.2 The AY-3-8910 Programmable Sound Generator (PSG)

This section describes the support software for the PSG. The files PSGDEF.ACM, PSGIO.ACM, LASER.ACM, WBOMB.ACM, WOLF.ACM, and RACECAR.ACM contain the software support for the PSG.

Unless otherwise noted, values requiring 8 bits or less are passed in register A. Values requiring more than 8 bits are passed in register HL. All registers are preserved unless needed to return data.

7.2.1 PSG Register Maintenance Routines

The PSG contains registers to control the tone and amplitude for three channels, a white noise generator, an envelope generator, and two parallel input/output ports. Each of these registers is described below.

7.2.1.1 Tone Control Registers

The PSG has three independent tone channels (channels A, B and C). The frequency for each channel is controlled by writing a 12-bit tone period value to registers in the PSG. The tone period is inversely related to the tone frequency (pitch) by the following formulae:

$$\text{Frequency} = 111861 / \text{Tone Period} \quad \text{equ. 7.2.1.1-1}$$

$$\text{Tone Period} = 111861 / \text{Frequency} \quad \text{equ. 7.2.1.1-2}$$

The second equation above will be the more useful. For example, suppose the PSG is to sound the frequency 261.624 hertz (middle C). Since the frequency is known and a tone period is needed for the PSG register, equation 7.2.1.1-2 is applied:

$$\text{Tone Period} = 111861 / 261.624$$

$$\text{Tone Period} = 427.56$$

$$\text{Tone Period} = 428 \quad (\text{rounded})$$

Note the final result must be rounded because an integral tone period value is written to the PSG. Since the result was rounded, a slight frequency error will occur. The exact frequency produced can now be computed using equation 7.2.1.1-1. In this instance the tone period is known, and the exact frequency is not known.

$$\text{Frequency} = 111861 / 428$$

$$\text{Frequency} = 261.357$$

A tone period value of 0 is used to turn off a tone channel.

The value 111861 is dependent on the PSG clock. The 3.579545MHz color burst clock from VDP pin 38 is divided by two yielding a 1.7897725MHz PSG clock. The PSG internally divides this clock by 16 yielding a final value of 111860.78 which is rounded to 111861.

The PSG tone period registers are 12 bit registers. The tone period values are passed in register HL. No checking is done to insure the values passed do not exceed 4095 (12 bits). Each of the PSG tone period register maintenance routines are described below:

7.2.1.1.1 Channel A Tone Period Registers

The PSG register maintenance routines for the Channel A Tone Period Registers are PS.WTA (Write Tone period A) and PS.RTA (Read Tone period A).

7.2.1.1.1.1 PS.WTA

PS.WTA will write the 12-bit tone period value passed in register HL to PSG register 0 (low order or fine value) and register 1 (high order or coarse value).

To cause PSG tone channel A to sound Middle C, the following code could be used:

```
LXI    HL,428           ;11861/261.624
CALL   PS.WTA
```

After this call, PSG tone channel A will be set to the frequency 261.357. No sound will be produced until the channel is enabled for output (see 7.2.1.6) and the proper amplitude is set (see 7.2.1.2.1).

7.2.1.1.1.2 PS.RTA

PS.RTA returns the current tone period value for channel A in register HL. The fine value (contents of PSG register 0) is placed in register L, and the 4-bit coarse value (contents of PSG register 1) is placed in register H.

To read current tone period for channel A, the following code could be used:

```
CALL   PS.RTA
```

After this call register HL will contain the current tone period value for channel A.

7.2.1.1.2 Channel B Tone Period Registers

The PSG register maintenance routines for the Channel B Tone Period Registers are PS.WTB (Write Tone period B) and PS.RTB (Read Tone period B).

7.2.1.1.2.1 PS.WTB

PS.WTB will write the 12-bit tone period value passed in register HL to PSG register 2 (low order or fine value) and register 2 (high order or coarse value).

To cause PSG tone channel B to sound A-440 (440 hertz), the following code could be used:

```
LXI    HL,254          ;11861/440
CALL   PS.WTB
```

After this call, PSG tone channel B will be set to the frequency 440.398. No sound will be produced until the channel is enabled for output (see 7.2.1.6) and the proper amplitude is set (see 7.2.1.2.2).

7.2.1.1.2.2 PS.RTB

PS.RTB returns the current tone period value for channel B in register HL. The fine value (contents of PSG register 2) is placed in register L, and the 4-bit coarse value (contents of PSG register 2) is placed in register H.

To read current tone period for channel B, the following code could be used:

```
CALL   PS.RTB
```

After this call register HL will contain the current tone period value for channel B.

7.2.1.1.3 Channel C Tone Period Registers

The PSG register maintenance routines for the Channel C Tone Period Registers are PS.WTC (Write Tone period C) and PS.RTC (Read Tone period C).

7.2.1.1.3.1 PS.WTC

PS.WTC will write the 12-bit tone period value passed in register HL to PSG register 4 (low order or fine value) and register 5 (high order or coarse value).

To cause PSG tone channel C to sound Middle C, the following code could be used:

```
LXI    HL,428           ;11861/261.624
CALL   PS.WTC
```

After this call, PSG tone channel C will be set to the frequency 261.357. No sound will be produced until the channel is enabled for output (see 7.2.1.6) and the proper amplitude is set (see 7.2.1.2.3).

7.2.1.1.3.2 PS.RTC

PS.RTC returns the current tone period value for channel C in register HL. The fine value (contents of PSG register 4) is placed in register L, and the 4-bit coarse value (contents of PSG register 5) is placed in register H.

To read current tone period for channel C, the following code could be used:

```
CALL   PS.RTC
```

After this call register HL will contain the current tone period value for channel C.

7.2.1.2 Amplitude Control Registers

The amplitude for each of the three tone channels is controlled by writing an amplitude control mask to the Amplitude Control Register for that channel. The amplitude control mask allows for two modes of operation. A channel can either have a fixed amplitude value, or be placed under control of the envelope generator. Bit 4 of the amplitude control mask selects the mode of operation. If bit 4 is ON (PS.VLA in PSGDEF.ACM), the channel amplitude is placed under the control of the envelope generator (see 7.2.1.4). All other bits in the amplitude control mask are ignored in this mode. If bit 4 is OFF (PS.FLA in PSGDEF.ACM), then the amplitude level is set by the low order 4 bits of the amplitude control mask. This mode allows for 16 logarithmically related amplitude steps. Since the human ear responds logarithmically to amplitude changes, these steps will be heard as 16 equally stepped changes in amplitude.

The routines to maintain the Amplitude Control Register for each of the channels is described below:

7.2.1.2.1 Channel A Amplitude Control Register

The PSG register maintenance routines for the Channel A Amplitude Control Register are PS.WAA (Write channel A Amplitude control register) and PS.RAA (Read channel A Amplitude control register).

7.2.1.2.1.1 PS.WAA

PS.WAA will write the amplitude control mask passed in register A to the PSG Channel A Amplitude Register (register 8).

To set channel A's amplitude at maximum value, the following call could be made:

```
MVI    A,PS.FLA+15
CALL   PS.WAA
```

After this call, the channel A amplitude will be at its maximum level and not under the control of the envelope generator.

7.2.1.2.1.2 PS.RAA

PS,RAA returns the current amplitude control mask for channel A in register A.

To read the current amplitude control mask for channel A, the following call could be made:

```
CALL   PS.RAA
```

After this call, register A will contain the current amplitude control mask for channel A.

7.2.1.2.2 Channel B Amplitude Control Register

The PSG register maintenance routines for the Channel B Amplitude Control Register are PS.WBA (Write channel B Amplitude control register) and PS.RBA (Read channel B Amplitude control register).

7.2.1.2.2.1 PS.WBA

PS.WBA will write the amplitude control mask passed in register A to the PSG Channel B Amplitude Register (register 9).

To set channel B's amplitude under the control of the envelope generator, the following call could be made:

```
MVI    A,PS.VLA
CALL   PS.WBA
```

After this call, channel B's amplitude will be under the control of the envelope generator.

7.2.1.2.2.2 PS.RBA

PS.RBA returns the current amplitude control mask for channel B in register A.

To read the current amplitude control mask for channel B, the following call could be made:

```
CALL   PS.RBA
```

After this call, register A will contain the current amplitude control mask for channel B.

7.2.1.2.3 Channel C Amplitude Control Register

The PSG register maintenance routines for the Channel C Amplitude Control Register are PS.WCA (Write channel C Amplitude control register) and PS.RCA (Read channel C Amplitude control register).

7.2.1.2.3.1 PS.WCA

PS.WCA will write the amplitude control mask passed in register A to the PSG Channel C Amplitude Register (register 10).

To set channel C's amplitude at maximum value, the following call could be made:

```
MVI    A,PS.FLA+15
CALL   PS.WCA
```

After this call, the channel A amplitude will be at its maximum level and not under the control of the envelope generator.

7.2.1.2.3.2 PS.RCA

PS.RCA returns the current amplitude control mask for channel C in register A.

To read the current amplitude control mask for channel C, the following call could be made:

```
CALL    PS.RCA
```

After this call, register A will contain the current amplitude control mask for channel C.

7.2.1.3 Noise Generator Register

The PSG has a white noise generator. The noise frequency is controlled by writing a 5-bit noise period to the Noise Period Register. Equations 7.2.1.1-1 and 7.2.1.1-2 apply to frequency and period calculations for the noise generator. The 5-bit noise period is passed in register A. The Noise Period Register may be read or written with the routines described below:

7.2.1.3.1 PS.WNP

PS.WNP will write the 5-bit noise period passed in register A to PSG register 6 (Noise Period Register).

To set the noise frequency to 10 Khz, the following call could be made:

```
MVI    A,11    ;111861/10000  
CALL   PS.WNP
```

After this call, the noise frequency will be set to 10.169 Khz. Before noise will be heard, the noise generator must be enabled for each channel where noise is to be heard (see 7.2.1.6.).

7.2.1.3.2 PS.RNP

PS.RNP returns the current noise period value in register A.

To read the current noise period, the following call could be made:

CALL PS.RNP

After this call, register A will contain the noise period.

7.2.1.4 Envelope Generator

The envelope generator consists of two parts: an envelope period value and an envelope shape/cycle control mask.

The envelope period value controls the duration of the envelope, and the envelope shape/cycle control mask is used to define the envelope's shape and whether or not the envelope repeats. A complete description of the Envelope Period Register and the Envelope Shape/Cycle Register follows:

7.2.1.4.1 Envelope Period Registers

The PSG register maintenance routines for the Envelope Period Registers are PS.WEP (Write Envelope Period registers) and PS.REP (Read Envelope Period registers).

The envelope period value is a 16-bit value. The envelope period is inversely related to the envelope frequency by the following formulae:

$$\text{Frequency} = 6991 / \text{Envelope Period} \quad \text{equ. 7.2.1.4-1}$$

$$\text{Envelope Period} = 6991 / \text{Frequency} \quad \text{equ. 7.2.1.4-2}$$

$$\text{Envelope Period} = 6991 * \text{Cycle Time} \quad \text{equ. 7.2.1.4-3}$$

Equation 7.2.1.4-3 will be the most useful equation when the envelope is used to control a decaying sound.

7.2.1.4.1.1 PS.WEP

PS.WEP writes the 16-bit tone period value in register HL to PSG register 11 (low order or fine value) and register 12 (high order or coarse value).

To write the envelope period value for a decaying sound which will last 2 seconds the following call could be made:

```
LXI    HL,13982      ;6991*2
CALL   PS.WEP
```

After this call, the Envelope Period Registers will be set for a 2 second period.

7.2.1.4.1.2 PS.REP

PS.REP returns the current envelope period value in register HL. The fine value (contents of PSG register 11) is placed in register L, and the coarse value (contents of PSG register 12) is placed into register H.

To read the current value of the Envelope Period Registers, the following call could be made:

```
CALL PS.REP
```

After this call register HL will contain the current envelope period value.

7.2.1.4.2 Envelope Shape/Cycle Control Register

The Envelope Shape/Cycle Control Register contains a 4-bit mask which controls the shape and cycling of the envelope. The function of each of the bits in the mask is described below (the bit name from PSGDEF.ACM is enclosed in parenthesis):

BIT 0 (PS.HLD)	ON: Limit envelope to one cycle. OFF: Do not limit envelope to one cycle.
BIT 1 (PS.ALT)	ON: Reverse counter direction at cycle end. OFF: Do not reverse counter direction.
BIT 2 (PS.ATT)	ON: Start counter direction up. OFF: Start counter direction down.
BIT 3 (PS.CNT)	ON: Start cycle over if bit 0 not ON. OFF: Reset counter to zero after one cycle and hold.

The PSG register maintenance routines for the Envelope Shape/Cycle Control Register are PS.WEC (Write Envelope shape/cycle Control Register) and PS.REC (Read Envelope shape/cycle Control register).

7.2.1.4.2.1 PS.WEP

PS.WEP will write the Envelope Shape/Cycle Control register (register 13) from the value passed in register A.

To set the envelope shape/cycle to a one cycle decay the following call could be made:

```
MVI    A,PS.HLD
CALL   PS.WEP
```

After this call the Envelope Shape/Cycle Control Register will be set to decay for one cycle. This shape/cycle will control the amplitude of all channels which have enabled the envelope generator in their respective Amplitude Control Register (see 7.2.1.2).

7.2.1.5 Parallel Ports

The PSG has two 8-bit parallel ports, each of which are capable of either input or output. Two bits from each port are brought out to each joystick connector as follows:

<u>Bit</u>	<u>Joystick</u>
0	1
1	1
2	2
3	2
4	3
5	3
6	4
7	4

For detailed information on interfacing to the parallel ports refer to section 4.2.1.

Software support for reading and writing the parallel ports is described below:

7.2.1.5.1 Parallel Port A

The PSG register maintenance routines for parallel port A are PS.WPA (Write Parallel port A) and PS.RPA (Read Parallel port A).

7.2.1.5.1.1 PS.WPA

PS.WPA writes the 8-bit value passed in register A to parallel port A. In order for this operation to work properly, parallel port A must have been enabled for output (see 7.2.1.6).

The following call could be made to set all bits on parallel port A:

```
MVI    A,11111111B
CALL   VP.WPA
```

After this call, all bits on parallel port A will be ON.

7.2.1.5.1.2 PS.RPA

PS.RPA returns the data on parallel port A in register A. In order for this operation to work properly, parallel port A must have been enabled for input (see 7.2.1.6).

7.2.1.5.2 Parallel Port B

The PSG register maintenance routines for parallel port B are PS.WPB (Write Parallel port B) and PS.RPB (Read Parallel port B).

7.2.1.5.2.1 PS.WPB

PS.WPB writes the 8-bit value passed in register A to parallel port B. In order for this operation to work properly, parallel port B must have been enabled for output (see 7.2.1.6).

The following call could be made to clear all bits on parallel port B:

```
XRA    A           ;CLEAR REGISTER A
CALL   PS.WPB
```

After this call, all bits on parallel port B will be OFF.

7.2.1.5.2.2 PS.RPB

PS.RPB returns the data on parallel port B in register A. In order for this operation to work properly, parallel port B must have been enabled for input (see 7.2.1.6).

7.2.1.6 PSG Enable Register

The PSG Enable Register controls the direction of the parallel ports and the three channels by enabling noise and tone on the channels. The PSG Enable Register is loaded with an 8-bit mask. The bit meanings of the mask are described below:

BIT 0 (PS.ETA)	ON: Enable tone on channel A.
(PS.DTA)	OFF: Disable tone on channel A.
BIT 1 (PS.ETB)	ON: Enable tone on channel B.
(PS.DTB)	OFF: Disable tone on channel B.
BIT 2 (PS.ETC)	ON: Enable tone on channel C.
(PS.DTC)	OFF: Disable tone on channel C.
BIT 3 (PS.ENA)	ON: Enable noise on channel A.
(PS.DNA)	OFF: Disable noise on channel A.
BIT 4 (PS.ENB)	ON: Enable noise on channel B.
(PS.DNB)	OFF: Disable noise on channel B.
BIT 5 (PS.ENC)	ON: Enable noise on channel C.
(PS.DNC)	OFF: Disable noise on channel C.
BIT 6 (PS.PAI)	ON: Enable parallel port A for input.
(PS.PAO)	OFF: Enable parallel port A for output.
BIT 7 (PA.PBI)	ON: Enable parallel port B for input.
(PA.PBO)	OFF: Enable parallel port B for output.

All other registers are usually initialized before the Enable Register is written. If the parallel ports are not being used, it is advisable to enable them for input. The PSG register maintenance routines for the Enable Register are PS.WER (Write Enable Register) and PS.RER (Read Enable Register).

7.2.1.6.1 PS.WER

PS.WER writes the 8-bit enable mask passed in register A to the PSG Enable Register.

Assuming the parallel ports are not needed, the following call could be made to set both parallel ports for input (allowing for accidental shorting), noise on channel B, and tone on channel A:

```
MVI    A,PS.PAI+PS.PBI+PS.ENB+PS.ETA
CALL   PS.WER
```

After this call both parallel ports will be enabled for input, channel B will be enabled for noise only, and channel A will be enabled for tone. If the other registers have been set properly and the audio cable is connected to an amplifier, the PSG will generate sound.

7.2.2 Sound Effect Examples

There are 4 ACM files which contain sound effect examples which closely follow the examples in chapter 6 of the PSG data sheet. These sound effects can be included in a program by using an XTEXT statement. The sound effects are written as subroutines. For example, the XTEXT statement for the sound effect should not appear in the main line program. A CALL statement for the sound effect should be used to invoke a sound effect subroutine. Each of the sound effects is described below:

7.2.2.1 Laser Sound Effect

This sound effect is in the file LASER.ACM. To use this sound effect, XTEXT the file LASER.ACM outside of the main line code. Whenever this sound effect is desired, CALL the subroutine LASER.

7.2.2.2 Whistling Bomb Effect

This sound effect is in the file WBOMB.ACM. To use this sound effect, XTEXT the file WBOMB.ACM outside of the main line code. Whenever this sound effect is desired, CALL the subroutine WBOMB.

7.2.2.3 Wolf Whistle Sound Effect

This sound effect is in the file WOLF.ACM. To use this sound effect, XTEXT the file WOLF.ACM outside of the main line code. Whenever this sound effect is desired, CALL the subroutine WOLF.

7.2.2.4 Race Car Sound Effect

This sound effect is in the file CAR.ACM. To use this sound effect, XTEXT the file RACECAR.ACM outside of the main line code. Whenever this sound effect is desired, CALL the subroutine CAR.

7.3 The ADC0808/ADC0809 Analog to Digital Converter (A/D)

This section describes the programming of the 8 channel A/D. The software support consists of two files: ADDEF.ACM and ADIO.ACM. The A/D component on the HA-89-3 consists of an eight channel (multiplexed) analog to digital converter. By specifying a channel the corresponding analog voltage can be converted to an 8-bit value, referred to as Number below.

The A/D generates an eight bit unsigned number based on the relationship of its input voltage to a reference voltage. The closer the input voltage to the reference voltage the higher the number generated, until eventually the input voltage equals the reference voltage and the A/D returns a value of 255. The formula which governs the number generated is:

$$\text{Number} = (\text{Vin} / \text{Vref}) * 255 \quad \text{equ. 7.3-1}$$

or simply

$$\text{Number} = (\text{Input Voltage} / 10) * 255$$

Voltages which slightly exceed the reference voltage will convert to 255, voltages which greatly exceed the reference voltage may damage the A/D.

The only routine required to support the A/D is AD.RD.

7.3.1 AD.RD

AD.RD converts the voltage on the analog channel passed in register L to a number and returns the number in register A. The following call could be made to read the voltage on analog channel 3:

```
MVI    L,3
CALL   AD.RD
```

After this call, register A contains the number based on equation 7.3-1 where Vin is the voltage on analog channel 3.

For AD.RD to work, the PIC must have been initialized properly. The routine PI.IPL (section 7.5.3) will initialize the PIC so that AD.RD can inquire when the ADC has completed its conversion.

7.4 The 8253 Counter/Timer Chip (CTC)

This section describes the support software for the CTC. The files CTCDEF.ACM and CTCIO.ACM contain the software support for the CTC.

The CTC contains a control word register and three 16-bit, pre-settable, down counters. Routines are available to read or write each counter and to write the mode word into the control word register.

7.4.1 Control Word Register

The control word register provides for counter select, read/write the high/low byte, mode specification, and BCD/Binary counter format.

CT.WMW (Write Mode Word) writes the contents of register A into the Control Word Register of the CTC. The definition of the bit assignments of the control word may be found in the 8253 data sheet in appendix E. Having loaded register A with the desired control word a call to the routine should be made as follows:

```
CALL CT.WMW
```

7.4.2 Load Counter

To initialize a counter to some value it is necessary to be able to write an arbitrary 16-bit quantity to the CTC. Since only 8-bits can be written at a time, two successive writes are necessary. To accomplish this a control word register write (via CT.WMW) should be done with RL1=1 and RLO=1 along with SC0,1 set to select the appropriate counter. The mode must also be specified by M2, M1 and M0 in the control word. After having called CT.WMW with the control word in register A, see section 7.4.1 above, two successive calls to the appropriate load counter routine must be given. There are three load counter routines, one for each counter. The naming of these are CT.LC0 to load CTC counter 0, CT.LC1 for counter 1 and CT.LC2 for counter 2.

In the example below counter 0 of the CTC will be set up to generate a square wave with a period of 10 ms ($f=100$ Hz). It must be recalled that counter 0 is prewired on the board with a clock input of 1.7898MHz (see section 4.5). If the counter is preloaded with a hex value of 45EA then it will take 10.00015 ms for the count to decrement to zero with the above clock input. Each time the count reaches zero the counter will be re-loaded from an internal register in the CTC with the 45EA value and down counting will resume. Once loaded, no software service is needed until it is decided to change the count period or stop the square wave

generation. OUT0 on the CTC connector would then provide the 100 Hz output. In the example below, CT.CT0 selects counter 0; CT.RBB specifies a load of low byte followed by the high byte; and CT.MD3 specifies MODE 3 (Square Wave Rate Generator) for the selected counter. These parameters are defined in CTCDEF.ACM. The code to set up the 100 Hz output is as follows:

```

MVI    A,CT.CT0+CT.RBB+CT.MD3
CALL   CT.WMW
LXI    H,45EAH           ;Put 45EA in HL
MOV    A,L               ;Get low byte (L) of HL
CALL   CT.LC0
MOV    A,H               ;Get high byte (H) of HL
CALL   CT.LC0

```

Some other useful values for preloading counter 0 to achieve desired time periods are shown in the following table.

CTC Counter 0 Timing Constants		
Clock = 1.7898 MHz		
<u>HEX constant</u>	<u>DEC constant</u>	<u>Period in ms</u>
06FE	1790	1.00011
0DFC	3580	2.00022
22F5	8949	5.00000
45EA	17898	10.00000
8BD4	35796	20.00000
D1BE	53694	30.00000

If it is desired to load only the 8-bit low byte or high byte then the control word must have the appropriate value in its RL 2-bit field. The control word should be set with RL1=0, RL0=1 to load the low byte and RL1=1, RL0=0 to load the high byte. This control word is loaded to register A and a call is then made to CT.WMW. Remember that the mode and select counter along with BCD fields must be set also. A call would then be made to the appropriate load counter routine (CT.LC0, CT.LC1 or CT.LC2) with the byte to be written in register A.

7.4.3 Read Counter

There are three routines to read the content of a given CTC counter. These are named CT.RCn, where n may be 0,1 or 2. Thus CT.RC0 reads CTC counter 0 and places its contents into CPU register A, and likewise CT.RC1 or CT.RC2 put CTC counter 1 or 2 contents into register A. A representative call to read the contents of CTC counter 0 is shown below.

```
CALL   CT.RC0
```

7.5 The 8259A Programmable Interrupt Controller (PIC)

This section describes the support software for the PIC. The files PICDEF.ACM and PICIO.ACM contain the software support for the PIC.

The PIC consists of an Interrupt Request Register (IRR), In Service Register (ISR), Interrupt Mask Register (IMR), and Priority Resolver along with control logic and I/O buffers. There are four interrupt modes in the PIC which are as follows:

- 1) Fully nested mode
- 2) Rotating priority mode
- 3) Special mask mode
- 4) Polled mode

Two major methods are available within the chip for the resolution of the highest priority interrupt request. Automatic Rotation may be chosen for priority determination of requests which are from devices of equal importance, while Specific Rotation may be used when a ranking order of importance is desired for the in-coming requests. The mask register allows certain requests to be ignored if they are not desired at that time. The interrupt vector capability of the 8259A is not supported on the HA-89-3 board.

One should keep in mind that there is no reset pin on the 8259A. A H/Z-89 Reset will not re-initialize a PIC which has been programmed to generate interrupts, hence interrupts will continue to be present. ONLY A POWER-ON RESET OR SOFTWARE INITIALIZATION WILL RESET THE PIC. Thus this condition may prevent the system from coming up. If this happens it may be necessary to power off then back on. When using interrupts, before exiting, the appropriate command words (ICWs and OCWs) should be written so that all inputs are masked and no interrupts are generated.

It should also be stressed that unless interrupt handler software is present the PIC should not be allowed to generate an interrupt to the H/Z-89. This can be prevented in one of two ways. First, by setting up the PIC with all interrupt requests masked so that no request will generate an interrupt. Alternately, jumper J10 should not be strapped to a H/Z-89 interrupt vector pin (see section 6.4) thus preventing a PIC induced interrupt.

7.5.1 Initialization Command Words (ICWs)

There are four initialization command words (ICW1, ICW2, ICW3 and ICW4) whose detail format may be found in the data sheet in Appendix F. There are routines for writing each of these four command words. In each case, register A is loaded with the command word to be written and an appropriate CALL instruction is given. The naming convention for these routines are as follows:

<u>Routine</u>	<u>Action</u>
PI.IW1	Load Initialization Command Word 1
PI.IW2	Load Initialization Command Word 2
PI.IW3	Load Initialization Command Word 3
PI.IW4	Load Initialization Command Word 4

All addressing and board decoding is handled by the routines themselves. A representative call to write initialization command word 1 would appear as follows:

```
MVI    A,PI.NC4+PI.SIN+PI.IN8+PI.ETM
CALL   PI.IW1
```

In the above example all bit values were defined even if there values were zero in order to be explicit. It should also be noted that "PI.ETM" was specified. This implies that the PIC will be edge sensitive and not level sensitive to an interrupt request. Edge sensitivity is preferred for the HA-89-3 board.

The board does not support interrupt vectors to the H/Z-89; however, it is necessary to write the second command word. To accomplish this a dummy call must be made with register A loaded with zeros. Since there is only one 8259A, PI.SIN was specified above and no call to PI.IW3 should be given. The reference to PI.NC4 implies that no call to PI.IW4 should be given. For further info on these last two, see the data sheets on the 8259A (Appendix F) and the file PICDEF.ACM.

7.5.2 Operation Command Words (OCWs)

There are three routines to handle the writing of Operation Control words (OCWs), one for each of the three types (OCW1, OCW2 and OCW3). These command words, in general, control which of the various interrupt modes (see section 7.5 above) are in operation. The first of these, OCW1, sets and clears the mask bits in the Interrupt Mask Register (IMR) of the PIC. OCW2 controls the Rotate and End of Interrupt Modes and also determines the interrupt level acted upon. OCW3 controls Special Mask Mode, Poll Mode and the ability to read the Interrupt Request (IR) and In Service (IS) registers.

The three routines for writing the OCWs all require that register A be loaded with the control word first. Then an appropriate CALL instruction is given within the constraints of the following naming convention:

<u>Routine</u>	<u>Action</u>
PI.OW1	Load Operation Command Word 1
PI.OW2	Load Operation Command Word 2
PI.OW3	Load Operation Command Word 3

Again, all addressing and board decoding is done by the routines. An example of a call to write the interrupt mask is shown below.

```
MVI    A,FFH                ;Mask all requests
CALL   PI.OW1
```

It must be remembered that this should follow the call to the last "Load Initialization Command Word" routine. After the call to the last "Load Operation Command Word" routine has been given, then processor interrupt enable can be issued.

The following is an example of a sequence of both Initialization and Operation Command Word calls including processor interrupt disable and enable.

```
DI
MVI    A,PI.NC4+PI.SIN+PI.IN8+PI.ETM
CALL   PI.IW1
XRA    A                ;Clear Register A
CALL   PI.IW2           ;Write dummy value
MVI    A,FFH           ;Mask all requests
CALL   PI.OW1           ;Write mask word
EI                    ;re-enable interrupts
```

It must be remembered that the correct number of Initialization Command Word calls must be given as a function of SINGL (PI.SIN or PI.CAS) and IC4 (PI.IC4 or PI.NC4) in ICW1. SINGL=0 (PI.CAS, cascaded PICs) should never be specified since there is only one PIC on the board. If IC4=1 (PI.IC4) in ICW1, then a call to PI.IW4 must be placed between that to IW2 and OW1.

7.5.3 PI.IPL - Initialize Poll Mode

PI.IPL should be called in place of both Initialization and Operation Command Word routines. This routine places the PIC in poll mode so that subsequent reads will act as a poll to return the priority interrupt status information. This routine must be executed before any call to the ADC or PSS routines.

7.5.4 Read Status

It will be necessary to be able to determine the status of the PIC at various times. To accomplish this routines have been provided that will return the contents of several registers internal to the PIC. In each case the routine places the PIC

register contents into processor register A.

7.5.4.1 PI.RIR - Read Interrupt Request Register

PI.RIR returns the current contents of the PIC Interrupt Request Register (IRR) to processor register A. This routine is particularly suited for determining the completion status of the ADC or PSS. This is an 8-bit value which contains the levels requesting an interrupt to be acknowledged. Interrupt Request 0 would be the least significant bit (LSB) while Interrupt Request 7 would be the most significant bit (MSB). A standard CALL instruction should be used and register A may be considered to be a copy of the IRR. If a bit is "on" then the device is requesting an interrupt. The following is a list of the bit assignments and the corresponding device (see section 2 for device definitions).

<u>IRR</u>	<u>Device</u>
Bit 0	VDP
Bit 1	CTC 0
Bit 2	CTC 1
Bit 3	CTC 2
Bit 4	ADC
Bit 5	APU
Bit 6	PSS
Bit 7	-

7.5.4.2 PI.RIM - Read Interrupt Mask Register

PI.RIM returns the current contents of the PIC Interrupt Mask Register (IMR) to processor register A. This is an 8-bit value which contains a bit set for each input request line which is currently masked. The LSB corresponds to the interrupt mask bit for request 0, while the MSB corresponds to the interrupt mask bit for request 7. Again a standard CALL instruction should be used and register A will contain the current contents of the IMR.

7.5.4.3 PI.POL - Poll Priority Interrupt Status

PI.POL issues a Poll command to the PIC and returns, in register A, the interrupt status and the value of the highest priority request currently present. The MSB is equal to "1" if there is an interrupt present and "0" if not. If there is an interrupt present, the low-order three bits represent the binary coding (a value of 0-7) of the highest priority request. See section 7.5.4.1 for bit assignments of the devices generating the request. These assignments correspond to the binary coding of the priority request.

8 Theory of Operation

The basic HA-89-3 Color Graphics Board consists of seven major functions:

- 1) H/Z-89 bus interface
- 2) Device select
- 3) Video Display Processor (VDP)
- 4) Programmable Sound Generator (PSG)
- 5) Analog to Digital Converter (A/D)
- 6) Counter/Timer Chip (CTC)
- 7) Programmable Interrupt Controller (PIC)

Optional devices are covered in the corresponding manual for each option. Refer to the logic diagram of the HA-89-3 (appendix A) when reading the following theory of operation. The HA-89-3 board identifies ICs by grided location. Left to right is A to F and top to bottom is 1 to n, where n is the number of ICs vertically. In this section the location of an IC will be placed in square brackets. As an example 74LS138[B2] indicates a 74LS138 IC chip at location [B2].

8.1 H/Z-89 Bus Interface

Only the low order three bits of the H89 address bus (A0, A1 and A2) are available at the card edge connector. These three bits are used as modifiers once the card is selected. For example, A2=0 implies that a board device address is to be latched into the Device Address Register (DEV A, B, and C). If A2=1 then data will be sent to or from a specific device on the board. The address bit A1 has meaning only to the CTC. Address A0 is preprocessed thru the 16L8[A3] and is used by most devices on the board. The other bits are decoded on the H89 processor board and create the device enable lines (S0, S1, LP, and CA) at the bus edge connector.

The data bus (D7 - D0) is buffered by a 74LS245[A5] bidirectional bus transceiver. The direction of this transceiver is controlled by the I/O Read line BRDL. When an I/O read operation is in progress BRDL=L (L means logic low) and the board is selected $\overline{BS}=L$ then the 74LS245[A5] buffers the internal data bus onto the H89 data bus. Otherwise, if $\overline{BS}=L$ and BRDL=H (H means logic high) it buffers the H89 data bus onto the internal

data bus.

There are four signals which are buffered as they come from the H89 edge connector. The first of these is RESET L which is buffered by a 74LS240[B5] to produce RESET on the HA-89-3 board. RESET is further inverted by the same 74LS240 to provide $\overline{\text{RESET}}$. The other three signals $\overline{\text{BS}}$, BWRL ($\overline{\text{W}}$ on board) and A0 are buffered by the 16L8[A3].

The HA-89-3 uses wait states to provide for required set-up and hold times of the various devices on board. The 16L8[A3] and 16R8[A4] together act as a micro sequencer to provide for all of the timing requirements of all devices. Thus they generate the H89 bus signal WAIT L from the WAIT output of the 16L8 via the open collector driver 7406[B3]. This causes a wait state to be generated in the CPU.

Interrupts can be generated on one of three bus lines: INT3L, INT4L or INT5L. The INT output of the PIC (8259A[D3]) can drive one of these three vectored interrupt lines, chosen by jumper J10, via the open collector driver 7406[B3]. This facility is made available in case user written software requires it.

8.2 Device Selection

There are only three address bits available on the H89 bus (A0, A1 and A2). Since there are eight (counting the two D/As as one) major devices, some with multiple addresses, on the board, there is not sufficient external addressing from the bus. For this reason it was decided to provide an "on-board" device register which could be written to. This device register DEV is actually part of the 16R8[A4] and is three bits wide with outputs labeled DEV,A DEV,B and DEV,C. The register DEV will be written to if board select ($\overline{\text{BS}}=\text{L}$) is asserted along with bus write (BWRL=L) and a low on address 2 (A2=L). This register specifies the major device (one of eight) and is modified by the two address lines along with the bus read BRDL and write BWRL. These augmentations allow data vs. control and read vs. write operations for each device using A0 and A1. Once a device has been selected by the above write operation, data and control info can be sent to and from that device freely until an operation is needed on some other device. Sheet 1 of the logic diagrams provides a table called DEVICE OPERATIONS which details the interactions of these signal lines for a given device. Below is a table of the device address assignments used by the device register DEV:

<u>Dev Addr</u>	<u>Device</u>
0	PIC
1	VDP
2	PSG
3	CTC
4	ADC
5	PSS
6	DAC
7	APU

NOTE: For the three character mnemonic definitions see Sec. 2

8.3 Video Display Processor

The major components of the VDP section of the HA-89-3 consist of a TMS-9918A[E2], eight 4116 150ns Dynamic RAMs [F1] - [F8], a video amplifier CA3100[D2] and an output buffer transistor Q1 (2N2222). Timing for the VDP section is derived from a 10.738635 MHz crystal (three times the color burst frequency). The crystal has been set to the correct frequency and should need no adjustments.

Note that in the documentation for the TMS-9918A, the designation for a low order bit is opposite that of all other components on the HA-89-3. A "7" is used to designate the low order bit, while a "0" designates the high order bit.

The "CSW" pin of the TMS-9918A (pin 18) is used to select it for a write operation. The decode logic in SEL,1[B2] detects when the TMS-9918A is selected and an I/O write operation is in progress.

The "CSR" pin of the TMS-9918A (pin 15) is used to select it for a read operation. The decode logic in SEL,0[A2] detects when the TMS-9918A is selected and an I/O read operation is in progress.

The "MODE" pin of the TMS-9918A (pin 13) is used to select a control or data operation. This pin is connected to the buffered low order address bit (A0') generated by the 16L8[A3].

The video amplifier in the VDP section buffers the composite video output of the TMS-9918A (pin 36). In order to interface to as many different devices as possible (color monitors, RF modulators, video tape recorders, etc.) the video amplifier has adjustments for peak to peak output voltage VIDEO LEVEL (P2), and the DC bias voltage VIDEO OFFSET (P1). The output impedance of the video amplifier is 75 ohms.

8.4 Programmable Sound Generator

The PSG section of the HA-89-3 consists of an AY-3-8910[A1] and an input to the mixer [C3]. The AY-3-8910 contains three tone generators, two 8 bit parallel I/O ports, one noise generator, and one envelope generator. The two parallel I/O ports are brought out to the four joystick connectors.

Two bits from each of the two ports are brought out to each joystick. If one port is set up for input and the other for output, then each joystick can have two bits of input for switches, and two bits of output for LEDs.

The clock for the AY-3-8910 (pin 22) is derived from the color burst clock on the TMS-9918A (pin 38). This 3.579545 Mhz clock signal is divided by 16R8[A4] to produce the 1.7897725 Mhz clock for the AY-3-8910. This is the clock frequency used in the examples in the AY-3-8910 data sheet.

Due to the data bus setup and hold requirements of the AY-3-8910, the PSG section requires that wait states be added to I/O operations. This timing is generated by 16R8[A4] and 16L8[A3].

8.5 Analog to Digital Converter

The A/D section of the HA-89-3 consists of a single analog to digital converter chip ADC0809 (or ADC0808). This chip contains both the 8-input multiplexer and converter. It is intended for ratiometric conversion, hence Vref is provided to both the ADC0809 and the joystick connectors (pin 5). The eight analog channels are brought in from the 4 joystick connectors.

The A/D uses a successive approximation technique to convert the analog voltage to a digital number. A conversion requires a maximum of 83 microseconds.

To use the A/D, a 3-bit channel number is first written to the A/D by placing it on D0-D2 (data lines) and performing a write with the ADC preselected and A0=L.

8.6 Counter/Timer Chip

The 8253 contains three independent, programmable, multi-mode 16-bit counter/timers. One of the timers has a fixed input to CLK0 of the color burst frequency/2 or 1.7897725 MHz. This 1.79MHz clock is also available on the CTC connector. The other two have external inputs to CLK and GATE. All three channels have external outputs (see Sec. 4.5). Selection of the counter within the chip is determined by the value of the address lines A1 and A0.

8.7 Programmable Interrupt Controller

The 8259A supports interrupt requests from the VDP, each of the CTC counters, the ADC, the APU and the PSS. Each request is maskable and hence any combination of devices may be ignored. The command select address A0 (pin 27) is tied to the H89 bus address A0 via the 16L8[A3]. Since there is only one 8259A, SP/EN (pin 16) is tied high and CAS 0-2 are not used. The output INT (pin 17) may be tied via 7406[B3] to INT3L, INT4L or INT5L as a function of the jumper placement on J10. This allows a vectored interrupt to the H/Z-89, however the software must be set-up to handle the interrupts.

9 Warranty and Service

The HA-89-3 Color Graphics Board is fully warranted against electrical failure for a period of 90 days after date of purchase. This warranty is limited to electrical failure of the HA-89-3, and does not cover physical damage to the HA-89-3 circuit board, external amplifiers, monitors, RF modulators, video recorders, or other devices attached to the HA-89-3. This warranty does not cover the backplane or any other component of the computer system.

Should the HA-89-3 require service, mail it insured and postage paid to:

New Orleans General Data Services, Inc.
7230 Chadbourne Drive
New Orleans, Louisiana 70126

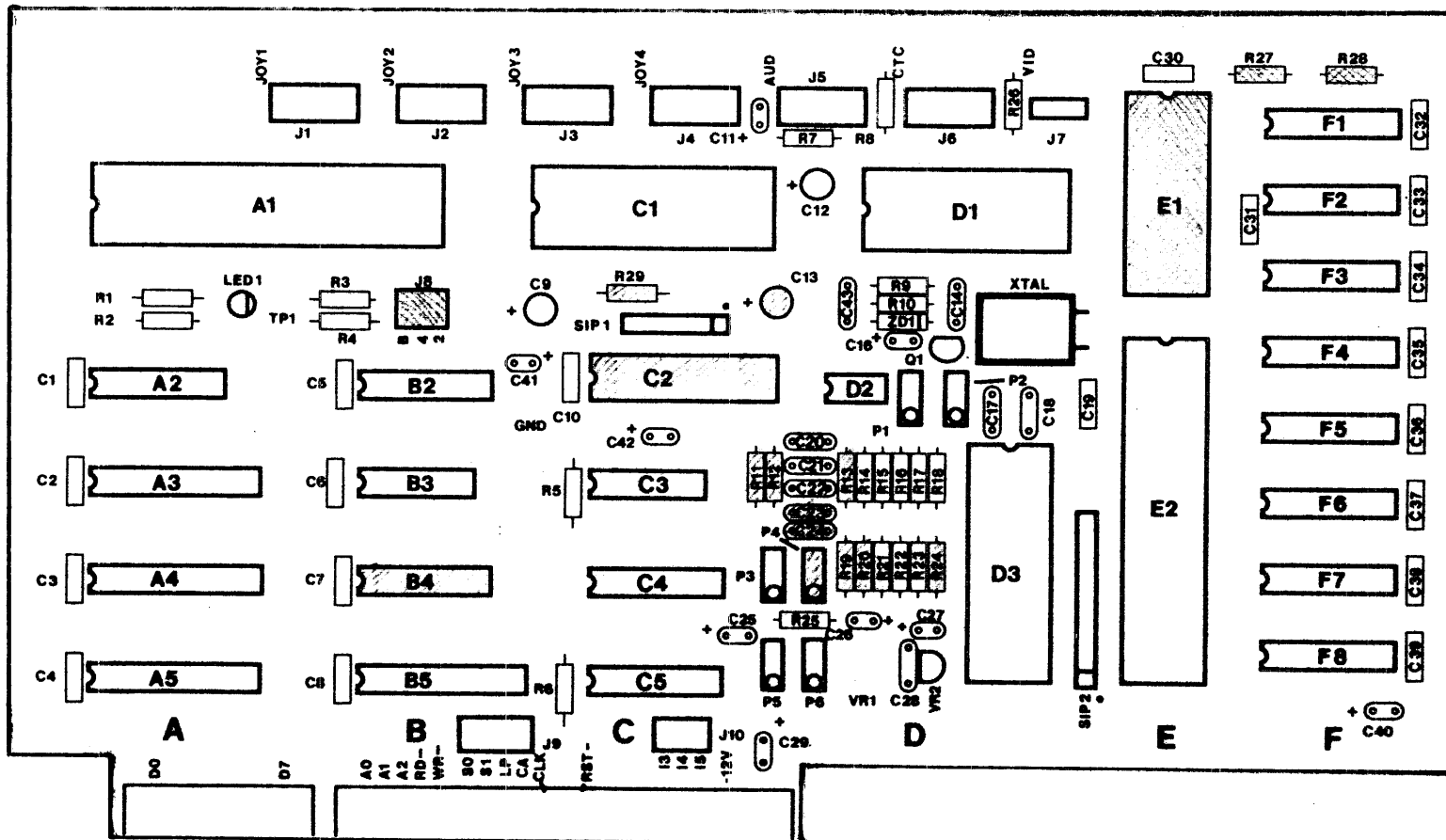
It is recommended that the board be packaged carefully and insured. A description of the problem should be included. If the HA-89-3 is under warranty, include a copy of the sales receipt showing date and place of purchase. Warranty work will not be performed unless a copy of the sales receipt is enclosed.

The HA-89-3 will be repaired and returned via COD insured first class mail or UPS for parts, labor, and postage. Warranty work will not be charged for labor or parts, but will be charged for shipping and insurance. A request for price quotation may be made before sending a board in for repair.

The above repair policy applies only to electrical failures. Boards with physical damage will be repaired or returned without repair at the discretion of New Orleans General Data Services, Inc., depending on the nature of and severity of the physical damage.

Appendix A

HA-89-3 Component Layout and Circuit Diagram



DEVICE OPERATIONS

DEV ADR	A2	A1	A0	R-W	OPERATION
X	0	X	X	W	LATCH DEVICE ADR (D2-D0)
0	1	X	0-1	R-W	PIC (A0 ← A0)
1	1	X	0-1	R-W	VDP (MODE ← A0)
2	1	X	0	W	PSG DATA WRITE
2	1	X	X	R	PSG DATA READ (A0=X)
2	1	X	1	W	PSG LATCH ADR
3	1	0-1	0-1	R-W	CTC (A1 ← A1, A0 ← A0)
4	1	X	0	W	ADC LATCH ADR (D2-0)
4	1	X	1	W	ADC START CONVERT (DATA=X)
4	1	X	X	R	ADC READ DATA
5	1	X	0	W	PSS WRITE PHONEME (D5-D0)
5	1	X	1	W	PSS WRITE INFLECTION (D2-D0)
6	1	X	0	W	DAC 0 WRITE (A1-A0 ← D5-D4, D3-D0 ← D3-D0)
6	1	X	1	W	DAC 1 WRITE (A1-A0 ← D5-D4, D3-D0 ← D3-D0)
7	1	X	0-1	R-W	APU (C/D ← A0)

JUMPERS

PIN NR'S	BOARD ADR J9	INT VECTOR J10	APU CLK J8
1-2	SERL 0	INT 3	BUS CLK
3-4	SERL 1	INT 4	3.58MHZ
5-6	LP	INT 5	1.79MHZ
7-8	CASS		

I/O CONNECTORS

PIN NR	JOY 1 (J1)	JOY 2 (J2)	JOY 3 (J3)	JOY 4 (J4)	AUD (J5)	CTC (J6)	VID (J7)
1 (STRIPE)	IOA0	IOA2	IOA4	IOA6	DAC 1	1.79MHZ (CLK 0)	GND
2	IOA1	IOA3	IOA5	IOA7	GND	OUT 0	VIDEO
3	IOB0	IOB2	IOB4	IOB6	DAC 0	CLK 2	GND
4	IOB1	IOB3	IOB5	IOB7	GND	+5V	
5	VREF	VREF	VREF	VREF	PSG	GATE 2	
6	GND	GND	GND	GND	GND	OUT 2	
7	IN 0	IN 2	IN 4	IN 6	MIX	CLK 1	
8	GND	GND	GND	GND	GND	GND	
9	IN 1	IN 3	IN 5	IN 7	PSS	GATE 1	
10	GND	GND	GND	GND	GND	OUT 1	

ABBREVIATIONS

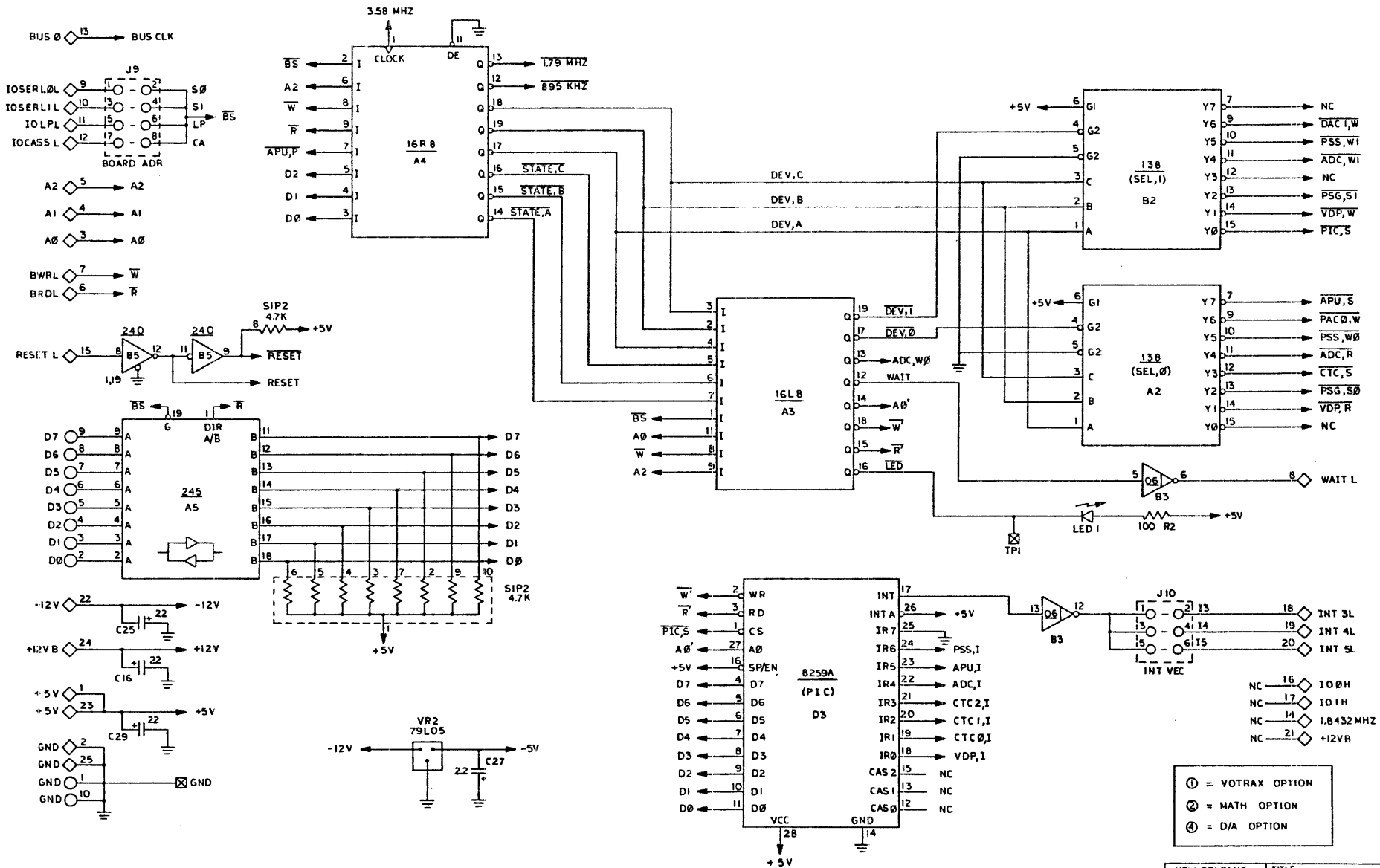
ABBR	IC	NAME
PIC	82595 (D3)	PRIORITY INTERRUPT CONTROLLER
VDP	TMS9918A (E2)	VIDEO DISPLAY GENERATOR
PSG	AY-3-8910 (A1)	PROGRAMMABLE SOUND GENERATOR
CTC	8253 (D1)	COUNTER-TIMER CHIP
ADC	ADC 0809 (C1)	ANALOG - TO - DIGITAL CONVERTER
PSS	SC-01 A (C2)	PROGRAMMABLE SPEECH SYNTHESIZER
DAC	AD7542 (C4,5)	DIGITAL - TO - ANALOG CONVERTER
APU	8231A (E1)	ARITHMETIC PROCESSING UNIT

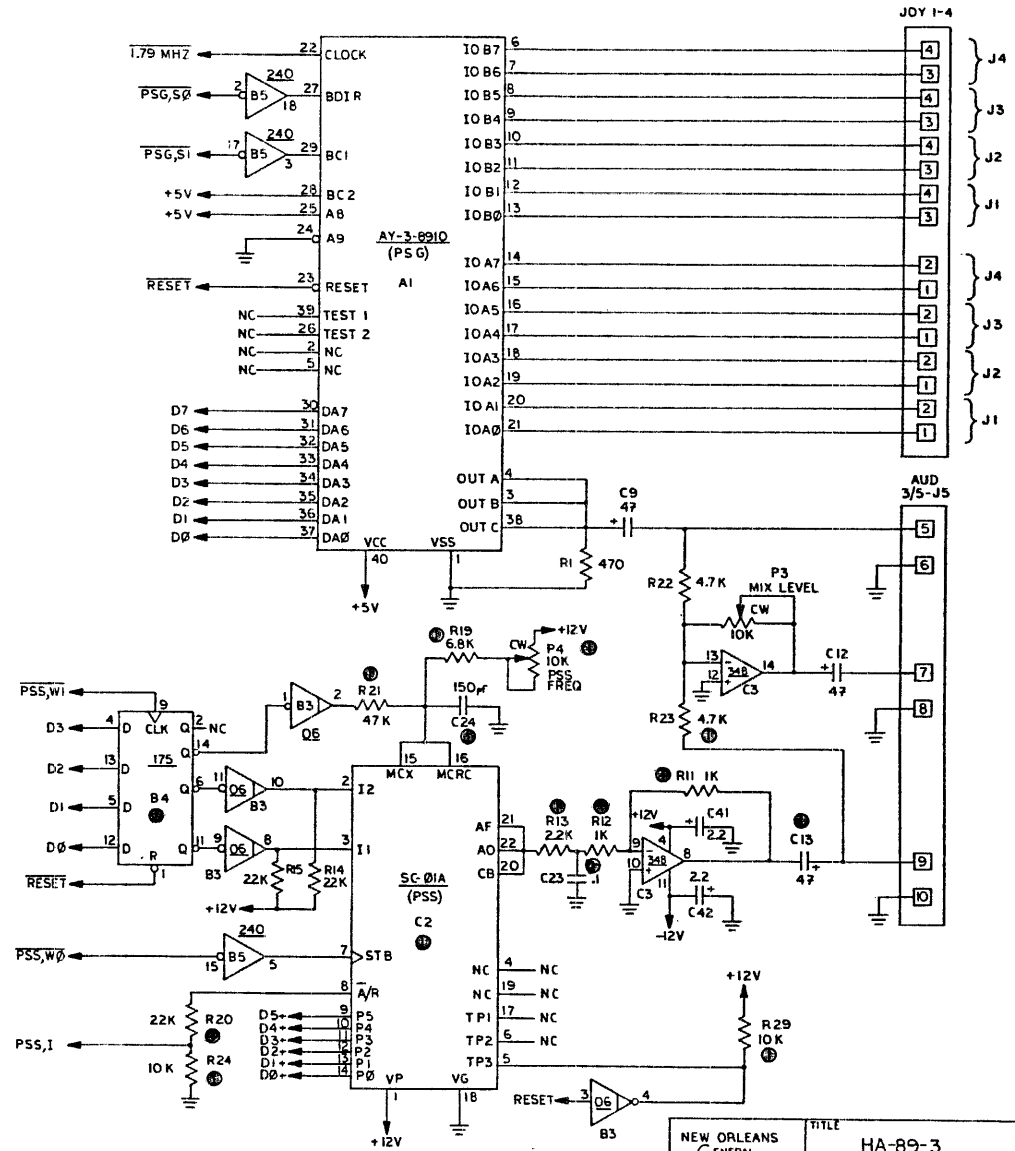
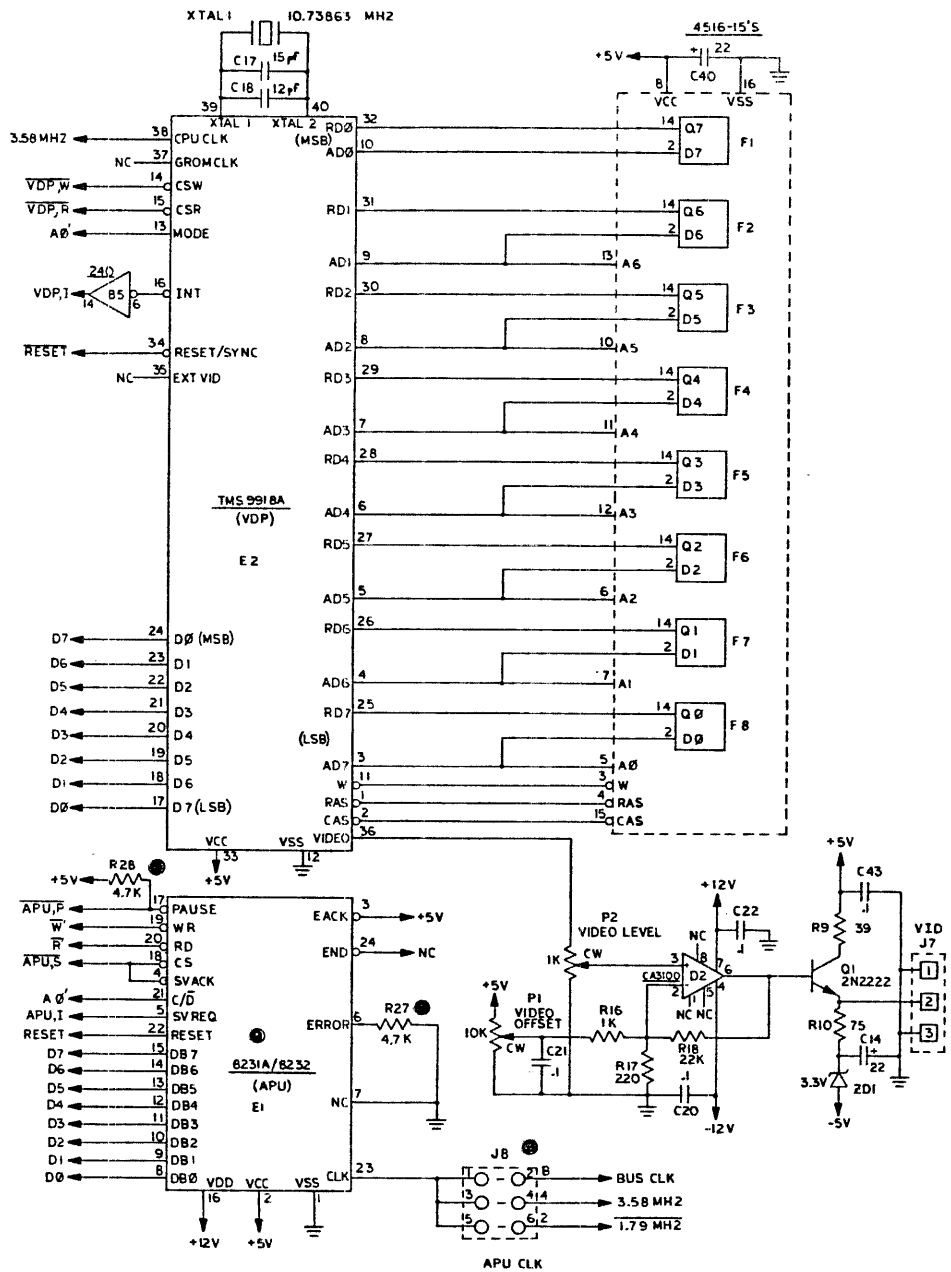
PIC VECTORS

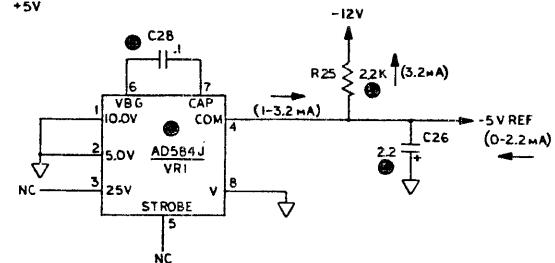
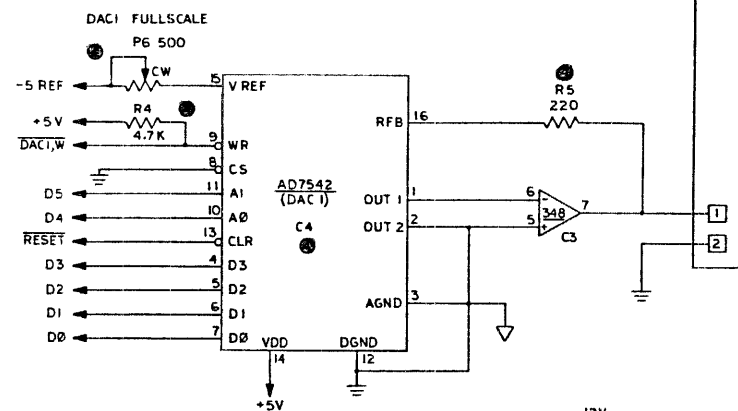
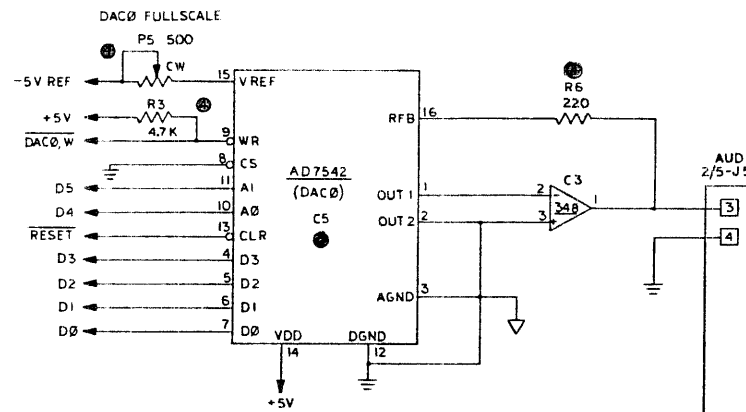
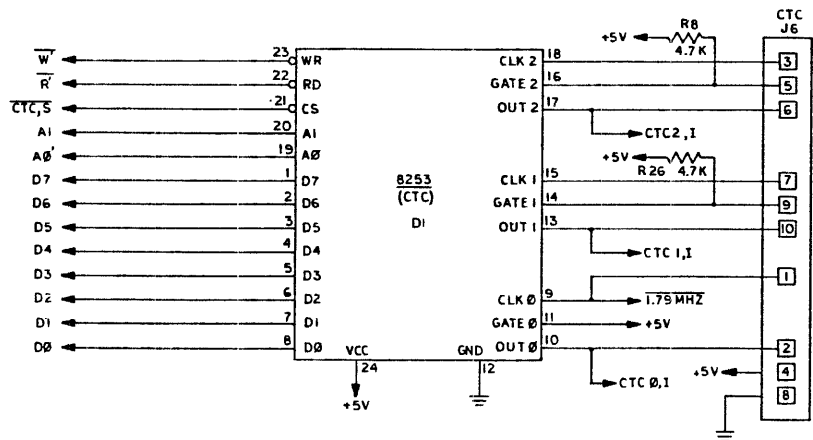
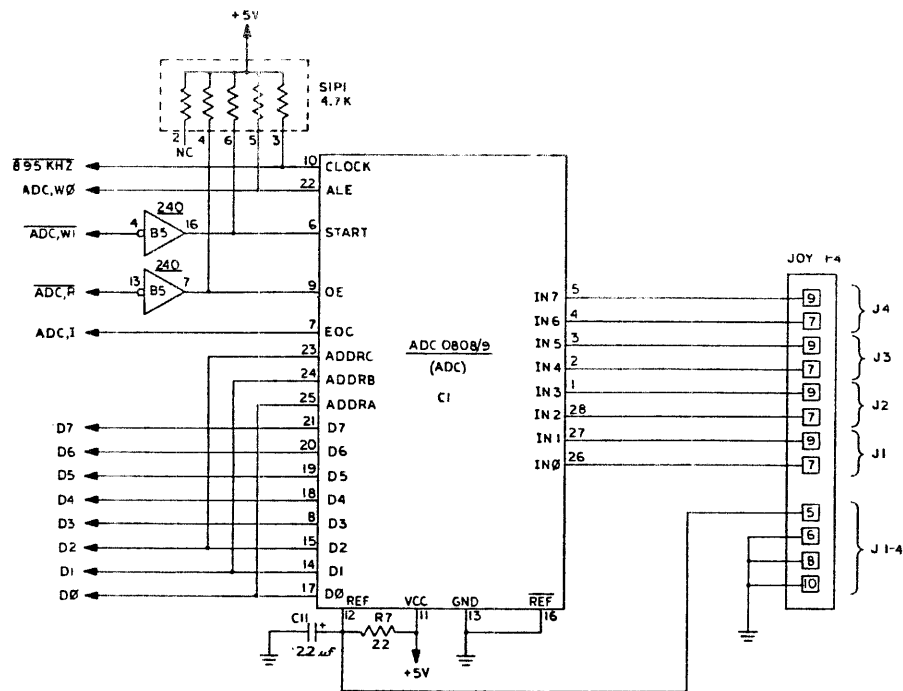
PIC VECTOR	DEVICE
0	VDP
1	CTC 0
2	CTC 1
3	CTC 2
4	ADC
5	APU
6	PSS
7	-

ADJUSTMENTS

POT	ADJ
P1	VIDEO OFFSET
P2	VIDEO LEVEL
P3	MIX LEVEL
P4	PSS FREQ
P5	DAC 0 FULLSCALE
P6	DAC 1 FULLSCALE





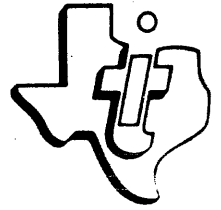


Appendix B

TMS 9918A Data Sheet
(VDP)

The following material is copyrighted by Texas Instruments Incorporated. It is reprinted here with the permission of Texas Instruments. This data sheet may not be reproduced for any purpose in whole or part without the expressed written consent of Texas Instruments.

The Engineering Staff of
TEXAS INSTRUMENTS INCORPORATED
Semiconductor Group



**TMS 9918A
VIDEO DISPLAY
PROCESSOR
DATA MANUAL**

NOVEMBER 1980

TEXAS INSTRUMENTS
INCORPORATED

1. INTRODUCTION

1.1 DESCRIPTION

The TMS 9918A Video Display Processor (VDP) is an N-channel MOS LSI device used in video systems where data display on a raster-scanned home color television set or color monitor is desired. The TMS 9918A generates all necessary video, control, and synchronization signals and also controls the storage, retrieval, and refresh of display data in the dynamic screen refresh memory. The interfaces to the microprocessor, refresh memory, and the TV require a minimum of additional electronics.

The VDP has four video display modes: Graphics I, Graphics II, Multicolor and Text mode. The Text mode provides twenty-four 40-character rows in two colors and is intended to maximize the capacity of the TV screen to display alphanumeric characters. The Multicolor mode provides an unrestricted 64 X 48 color-dot display utilizing 15 colors plus transparent. The Graphics I mode provides a 256 X 192 pixel display for generating pattern graphics in 15 colors plus transparent. The Graphics II mode is an enhancement of Graphics I mode, providing the capability to generate more complex color and pattern displays.

The video display consists of 35 planes, external video, backdrop, pattern plane, and 32 Sprite Planes. The planes are vertically stacked with the external video being the bottom or innermost plane. The backdrop plane is the next plane followed by the pattern plane that contains Graphics I and Graphics II patterns with the 32 Sprite Planes as the top planes. (A sprite is an object-oriented animation pattern that can be moved smoothly across the screen.)

The TMS 9918A VDP utilizes either a 4K, 8K, or 16K-type low-cost dynamic memory (TMS 4027, TMS 4108, TMS 4116) for storage of the display parameters.

1.2 FEATURES

- Single-chip interface to color TV's (excluding RAM and RF modulator).
- 256 X 192 resolution on TV screen
- 15 unique colors plus transparent
- General 8-bit bidirectional interface to CPU
- Direct wiring to 4K, 8K, or 16K dynamic RAM memories
- Automatic and transparent refresh of dynamic RAMs
- External video input capability
- NTSC - standard composite video output
- Unique planar representation for 3D simulation
- Standard 40-pin package

1.3 TYPICAL APPLICATIONS

- Color computer terminals
- Home computers
- Drafting/design aids
- Teaching aids
- Industrial process monitoring
- Home educational systems
- Animation aids

The following example of a typical application may help introduce the user to the TMS 9918A VDP. Figure 1-1 is a block diagram of a typical application. Each of the concepts presented below is described more fully in later sections of this manual.

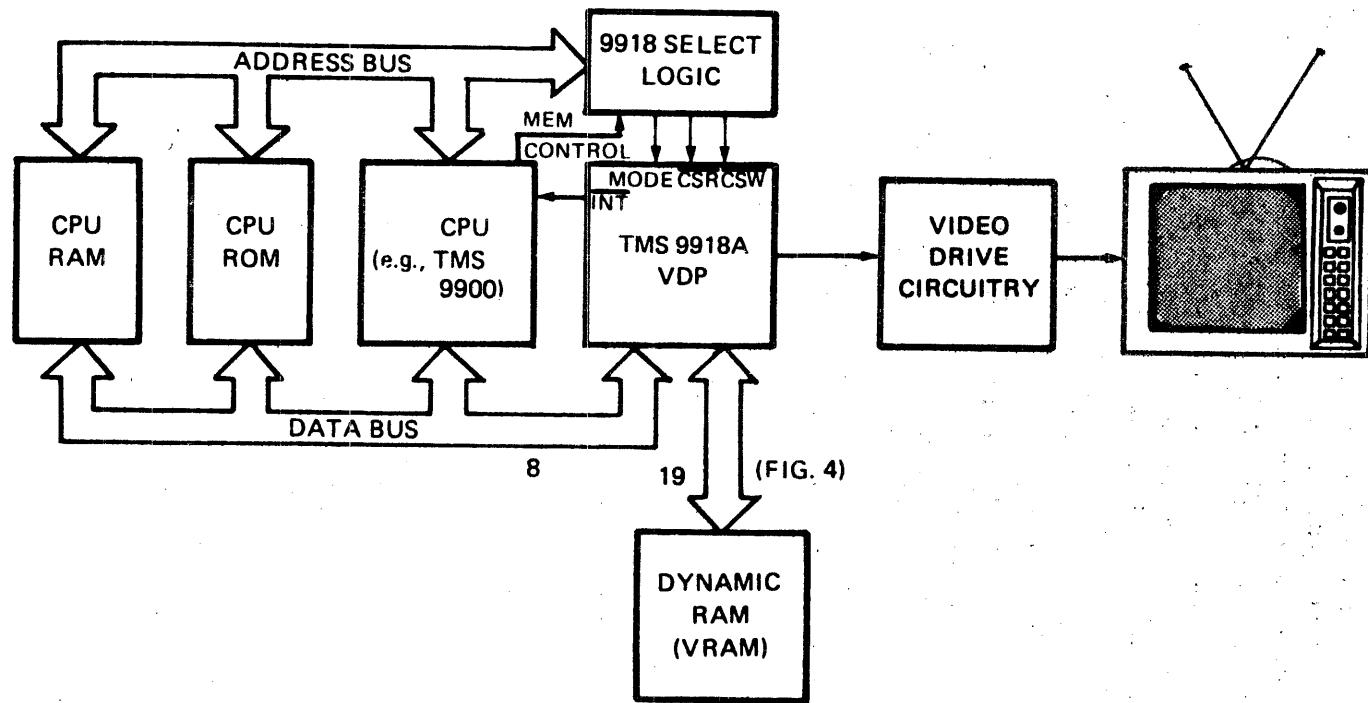


FIGURE 1-1 - SYSTEM BLOCK DIAGRAM

The VDP basically has three interfaces: CPU, color television, and dynamic refresh RAM (VRAM) the contents of which define the TV image. The TMS 9918A VDP also has eight write-only registers and a read-only status register.

The VDP communicates with the CPU via an 8-bit bidirectional data bus. Three control lines, decoded from the CPU address and enable lines, determine interpretation of the bus. Through the bus, the CPU can write to VRAM, read from VRAM, write to VDP registers, and read the VDP status. The VDP also generates an interrupt signal after every refresh of the TV display, which may be useful to the CPU.

The dynamic RAM interface consists of direct wiring of eight 4K X 1, 8K X 1, or 16K X 1 dynamic RAS/CAS-type RAMs to the TMS 9918 VDP. The amount of RAM required is dependent upon the features selected for use in the application.

The interface to the TV can consist of wiring the VDP's composite video output pin (suitably buffered) to the input of a color or black-and-white monitor, or an appropriate RF modulator may be used to feed the signal into a TV antenna terminal.

The VDP can operate in any one of four modes, each of which can affect the way the VRAM is mapped onto the television screen. In Graphics I and II modes, characters are mapped onto the screen in 8 X 8 pixel blocks, yielding 24 lines of 32 blocks (or "pattern positions") each. In Text mode, there are 24 lines of 40 blocks, each of which is 6 X 8 pixels. In Multicolor mode, there are 48 lines of 64 blocks, each of which is composed of 4 X 4 pixels, all of one solid color. In addition to these, objects termed "sprites" can be superimposed onto the television image. Furthermore, signals entering the VDP through the external video input can be used as a background to VDP-generated images.

1.4 DEFINITIONS

The following definitions will be useful in understanding the use of the TMS 9918 VDP:

- pixel — the smallest point on the TV screen that can be independently controlled
- NTSC — National Television Standards Committee which specifies the television signal standard for the USA
- VRAM — Video RAM; refers to the dynamic RAMs that connect to the VDP and whose contents define the TV image
- sprite — An object whose pattern is relative to a specified X, Y coordinate and whose position can therefore be controlled by that coordinate with a positional resolution of one pixel

2. ARCHITECTURE

The TMS 9918A Video Display Processor (VDP) is designed to provide a simple interface between a microprocessor and a raster-scanned color television. Shown in Figure 2-1 is a block diagram of the major portions of the VDP architecture. Described below are details of the various interfaces to the VDP, CPU, VRAM, and color television.

2.1 CPU INTERFACE

The VDP interfaces to the CPU using an 8-bit bidirectional data bus, three control lines, and an interrupt as shown in Figure 2-2. Through this interface the CPU can conduct four operations: write data bytes to VRAM, read data bytes from VRAM, write to one of the eight VDP write-only registers, and read the VDP Status Register. Each of these operations requires one or more data transfers to take place over the CPU/VDP data bus interface. The interpretation of the data transfer is determined by the three control lines of the VDP. It should be noted that the CPU can communicate with the VDP simultaneously and asynchronously with the VDP's TV screen refresh operations. The VDP performs memory management and allows periodic intervals of CPU access to VRAM even in the middle of a raster scan.

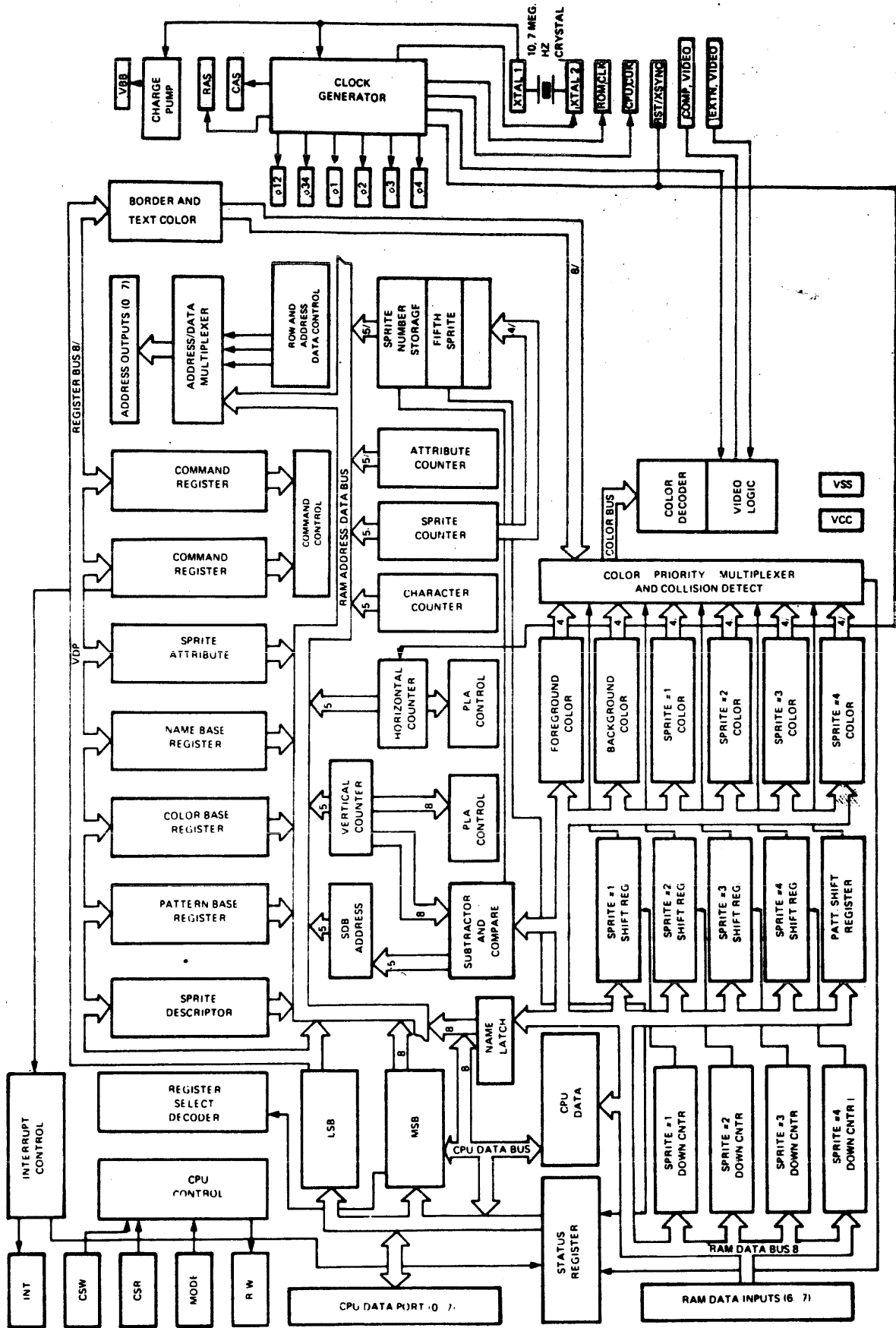


FIGURE 2-1 - TMS 9918 VDP BLOCK DIAGRAM

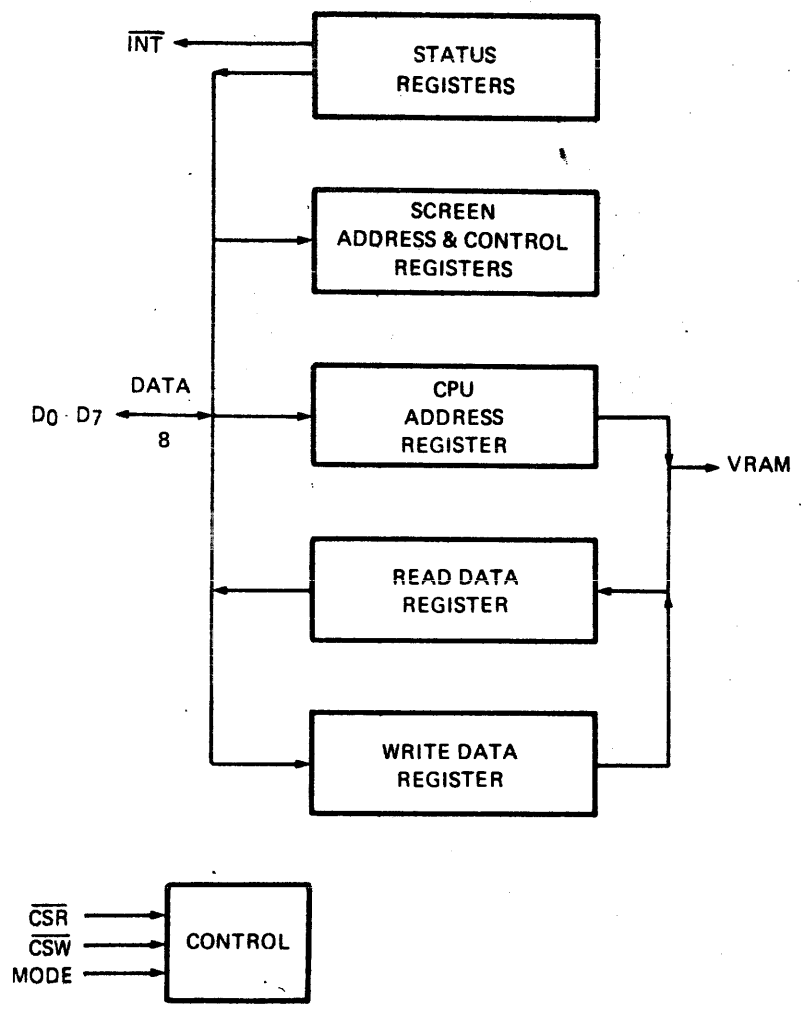


FIGURE 2-2 - VDP TO CPU INTERFACE

2.1.1 CPU Interface Control Signals

The type and direction of data transfers are controlled by the \overline{CSW} , \overline{CSR} , and MODE inputs. \overline{CSW} is the CPU-to-VDP write select. When it is active (low), the 8 bits on D0-D7 are strobed into the VDP. \overline{CSR} is the CPU-from-VDP read select. When it is active (low), the VDP outputs 8 bits on D0-D7 to the CPU. \overline{CSW} and \overline{CSR} should never be simultaneously low. If both are low, the VDP outputs data on D0-D7 and latches in invalid data.

MODE determines the source or destination of a read or write data transfer. MODE is normally tied to a CPU low order address line (A14 for TMS 9900).

2.1.2 CPU Write to VDP Register

The VDP has eight write-only registers and one read-only status register. The write-only registers control the VDP operation and determine the way in which VRAM is allocated. The status register contains interrupt, sprite coincidence and fifth sprite status flags.

Each of the eight VDP write-only registers can be loaded using two 8-bit data transfers from the CPU. Table 1 describes the required format for the two bytes. The first byte transferred is the data byte, and the second byte transferred controls the destination. The most-significant bit of the second byte must be a '1'. The next four bits are '0's, and the lowest three bits make up the destination register number. The MODE input is high for both byte transfers.

To rewrite the data for an internal register after a byte of data has been loaded, the status register must be read so that internal logic will accept the next byte as data and not as a register destination. This situation may be encountered in interrupt-driven program environments. Whenever the status of VDP write parameters is in question, this procedure should be used. Note that the CPU address is destroyed by writing to the VDP register.

2.1.3 CPU Write to VRAM

The CPU transfers data to the VRAM through the VDP using a 14-bit autoincrementing address register. Two-byte transfers are required to set up the address register. A one-byte-transfer is then required to write the data to the addressed VRAM byte. The address register is then autoincremented. Sequential VRAM write require only one-byte-transfer since the address register is already set up. During setup of the address register, the two most-significant bits of the second address byte must be '0' and '1' respectively. MODE is high for both address transfers and low for the data transfer. \overline{CSW} is used in all transfers to strobe the 8 bits into the VDP. See Table 1.

TABLE 1 – CPU/VDP DATA TRANSFERS

OPERATION	BIT								\overline{CSW}	\overline{CSR}	MODE
	0	1	2	3	4	5	6	7			
WRITE TO VDP REGISTER											
BYTE 1 DATA WRITE	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	0	1	1
BYTE 2 REGISTER SELECT	1	0	0	0	0	RS ₀	RS ₁	RS ₂	0	1	1
WRITE TO VRAM											
BYTE 1 ADDRESS SET UP	A ₆	A ₇	A ₈	A ₉	A ₁₀	A ₁₁	A ₁₂	A ₁₃	0	1	1
BYTE 2 ADDRESS SET UP	0	1	A ₀	A ₁	A ₂	A ₃	A ₄	A ₅	0	1	1
BYTE 3 DATA WRITE	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	0	1	0
READ FROM VDP REGISTER											
BYTE 1 DATA READ	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	1	0	1
READ FROM VRAM											
BYTE 1 ADDRESS SET UP	A ₆	A ₇	A ₈	A ₉	A ₁₀	A ₁₁	A ₁₂	A ₁₃	0	1	1
BYTE 2 ADDRESS SET UP	0	0	A ₀	A ₁	A ₂	A ₃	A ₄	A ₅	0	1	1
BYTE 3 DATA READ	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇	1	0	0

2.1.4 CPU Read from VDP Status Register

The CPU can read the contents of the status register with a single-byte transfer. \overline{MODE} is high for the transfer. \overline{CSR} is used to signal the VDP that a read operation is required.

2.1.5 CPU Read from VRAM

The CPU reads data from the VRAM through the VDP using the autoincrementing address register. A one-byte transfer is then required to read the data from the addressed VRAM byte. The address register is then autoincremented. Sequential VRAM data reads require only a one-byte transfer since the address register is already set up. During setup of the address register, the two most-significant bits of the second address byte must be '0's. By setting up the address this way, a read cycle to VRAM is initiated and read data will be available for the first data transfer to the CPU. (see Table 1). \overline{MODE} is high for the address byte transfers and low for the data transfers. The VDP requires approximately 8 microseconds to fetch the VRAM byte following a data transfer and 3 microseconds following address setup.

2.1.6 VDP Interrupt

The VDP \overline{INT} output pin is used to generate an interrupt at the end of each active-display scan, which is about every 1/60 second (color burst frequency/60, 192). The \overline{INT} output is active when the interrupt Enable bit (IE) in VDP register 1 is a '1' and the F bit of the status register is a '1'. Interrupts are cleared when the status register is read.

2.1.7 VDP Initialization

The VDP is externally initialized whenever the \overline{RESET} input is active (low) and must be held low for a minimum of 3 microseconds. The external reset synchronizes all clocks with its falling edge, sets the horizontal and vertical counters to known states, and clears VDP registers 0 and 1. The video display is automatically blanked since the BLANK bit in VDP register 1 becomes a '0'. The VDP, however, continues to refresh the VRAM even though the display is blanked. While the \overline{RESET} line is active, the VDP does not refresh VRAM.

2.2 VDP/VRAM INTERFACE

The VDP can access up to 16,384 bytes of VRAM using a 14-bit VRAM address. The VDP fetches data from the VRAM in order to process the video image as described later. The VDP also stores data in or reads in data from the VRAM during a CPU-VRAM data transfer. The VDP automatically refreshes the VRAM.

2.2.1 VRAM Interface Control Signals

The VDP-VRAM interface consists of two unidirectional 8-bit data buses and three control lines as shown in Figure 2-3. The VRAM outputs data to the VDP on the VRAM read data bus (RD0-RD7). The VDP outputs both the address and data to the VRAM over the VRAM address/data bus (AD0-AD7). The VRAM row address is output when \overline{RAS} is active (low). The column address is output when \overline{CAS} is active (low). Data is output to the VRAM when $\overline{R/W}$ is active (low).

2.2.2 VRAM Memory Types

The VDP can utilize 4027-type 4K, 4108-type 8K, or 4116-type 16K dynamic RAMs. The 4/16K bit in VDP register 1 is a '0' for 4027 type RAMs and a '1' for 4108- and 4116-type RAMs. Note that there is a minor difference between the way 4027's and 4108's/4116's are wired to the VDP. In the 4027, all \overline{CE} pins are tied to ground. In the 4108/4116 the A6 lines on the 4116 and 4108 (the same pin as \overline{CE} on 4027's) are all tied to AD1 on the TMS 9918A. A jumper can be used to select the VRAM-type.

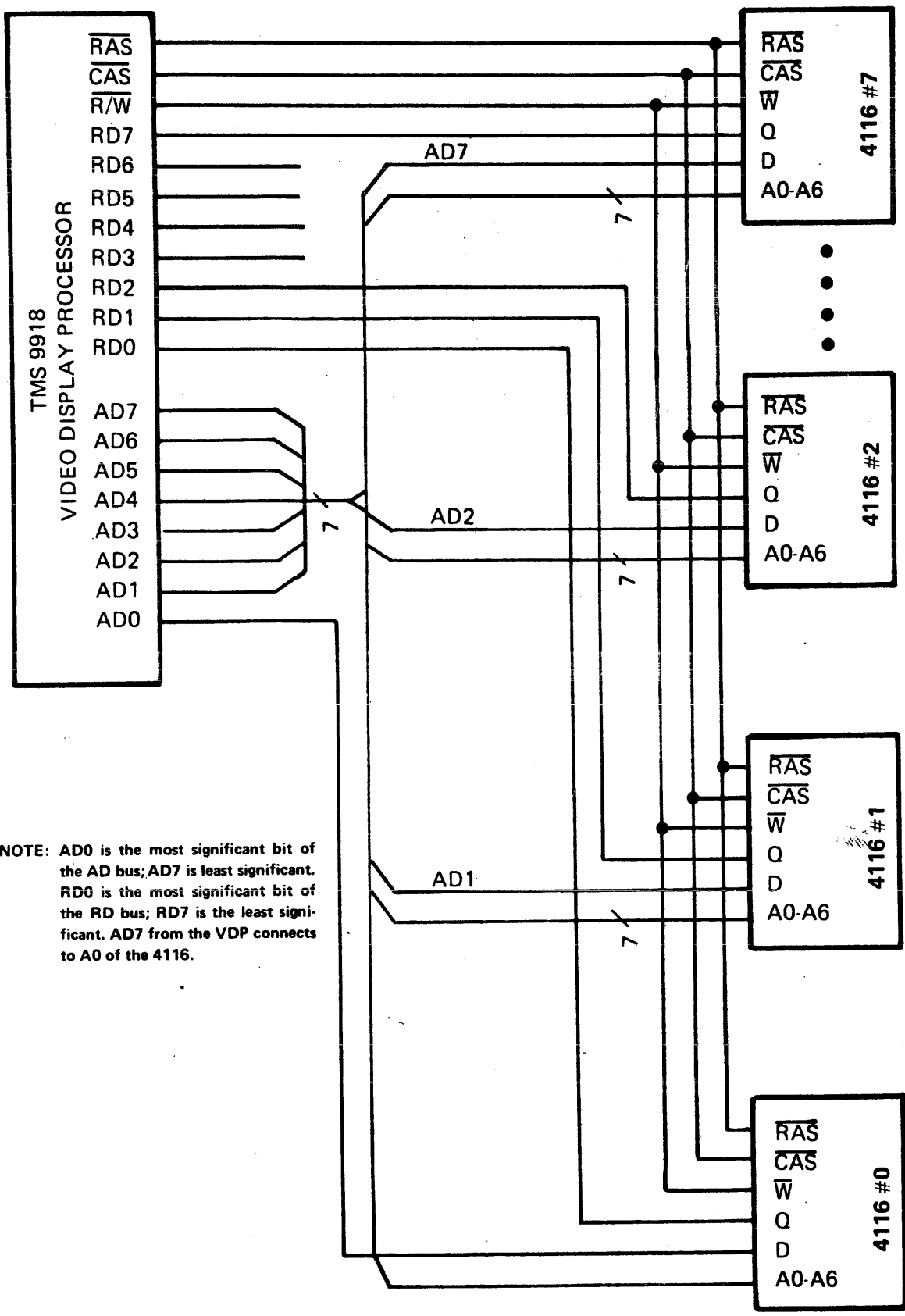


FIGURE 2-3 - VRAM INTERFACE

2.3 VDP/TV INTERFACE

The composite video output signal coming from the VDP drives an NTSC color monitor. This signal incorporates all necessary horizontal and vertical synchronization signals as well as luminance and chrominance information. In monitor applications, the requirements of the monitor should be studied to determine if the VDP can be connected directly to it. The internal output buffer device on the composite video pin is a source-follower MOS transistor that requires an external pull-down resistor to V_{SS} as shown in Figure 2-4. Typically a 1 kilohm resistor is recommended to provide a 1.9-volt synchronization level.

In some cases, it may be necessary to provide a simple interface circuit to match the VDP output voltages with the monitor specifications. To drive a standard television that is not outfitted with a composite video input, the signal can be run into the television antenna terminals by using an appropriate RF modulator on the VDP output. Care should be taken to ensure proper matchup between VDP, RF modulator, and TV.

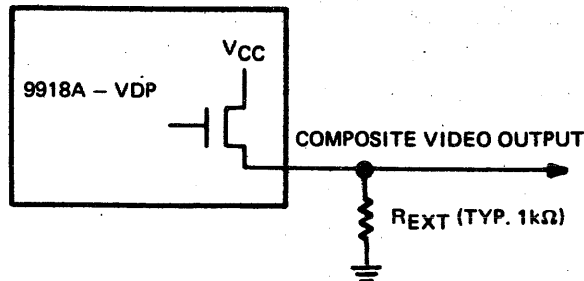


FIGURE 2-4 – COMPOSITE VIDEO PULL-DOWN CIRCUIT

2.4 WRITE-ONLY REGISTERS

The eight VDP write-only registers are shown in Figure 2-5. They are loaded by the CPU as described in Section 2.1.2. Registers 0 and 1 contain flags to enable or disable various VDP features and modes. Registers 2 through 6 contain values that specify starting locations of various sub-blocks of VRAM. The definitions of these sub-blocks are described in Section 2.7. Register 7 is used to define backdrop and text colors.

The following is a description of each register:

2.4.1 Register 0

Register 0 contains two VDP option control bits. All other bits are reserved for future use and must be '0's.

BIT 6 M3 (mode bit 3) See Section 2.4.2 for table and description.

BIT 7 External Video enable/disable

'1' enables external video input

'0' disables external video input

2.4.2 Register 1

Register 1 contains 8 VDP option control bits.

BIT 0 4/16K selection

'0' selects 4027 RAM operation

'1' selects 4108/4116 RAM operation

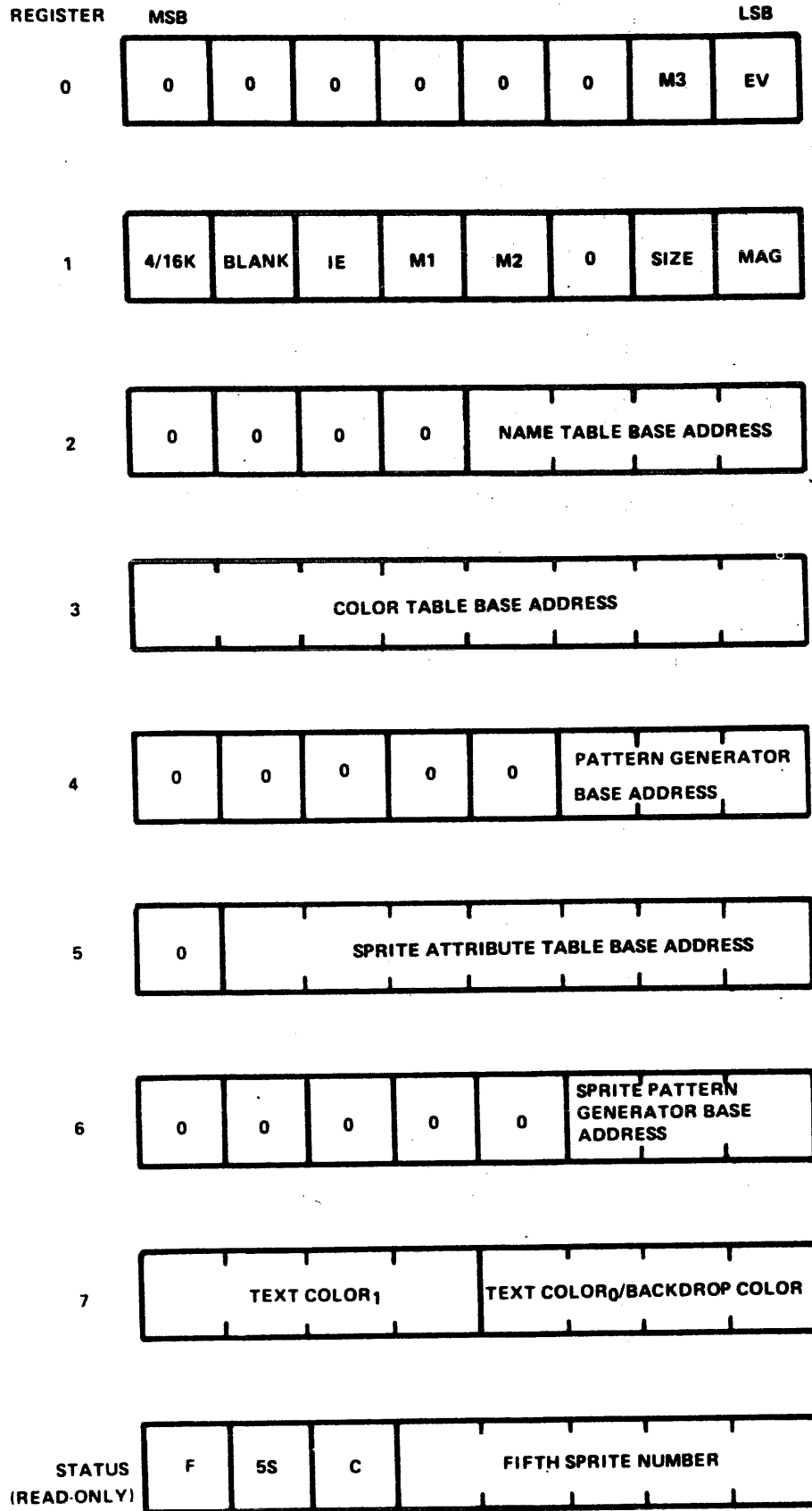


FIGURE 2-5 -- VDP REGISTERS

- BIT 1** BLANK enable/disable
- '0' causes the active display area to blank
 - '1' enables the active display
 - Blanking causes the display to show border color only
- BIT 2** IE (Interrupt Enable)
- '0' disable VDP interrupt
 - '1' enable VDP interrupt
- BIT 3,4** M1, M2 (mode bits 1 and 2)
- M1, M2 and M3 determine the operating mode of the VDP:
- | M1 | M2 | M3 | Mode |
|----|----|----|------------------|
| 0 | 0 | 0 | Graphics I mode |
| 0 | 0 | 1 | Graphics II mode |
| 0 | 1 | 0 | Multicolor mode |
| 1 | 0 | 0 | Text mode |
- BIT 5** Reserved
- BIT 6** Size (sprite size select)
- '0' selects Size 0 sprites (8 X 8 bit)
 - '1' selects Size 1 sprites (16 X 16 bits)
- BIT 7** MAG (Magnification option for sprites)
- '0' selects MAG0 sprites (1X)
 - '1' selects MAG1 sprites (2X)

2.4.3 Register 2

Register 2 defines the base address of the Name Table sub-block. The range on its contents is from 0 to 15. The contents of the register form the upper 4 bits of the 14-bit Name Table addresses; thus the Name Table base address is equal to (register 2) * 400 (hex).

XX XXXX XXXX XXXX 0-999
0-36F

2.4.4 Register 3

Register 3 defines the base address of the Color Table sub-block. The range on its contents is from 0 to 255. The contents of the register form the upper 8 bits of the 14-bit Color Table addresses; thus the Color Table base address is equal to (register 3) * 40 (hex).

XX XXXX XXXX XXXX mode 1 0-31₁₀
" 2

2.4.5 Register 4

Register 4 defines the base address of the Pattern, Text or Multicolor Generator sub-block. The range of its contents is 0 through 7. The contents of the register form the upper 3 bits of the 14-bit Generator addresses; thus the Generator base address is equal to (register 4) * 800 (hex).

YY YXXX XXXX XXXX 0-2047
0-7FF

2.4.6 Register 5

Register 5 defines the base address of the Sprite Attribute Table sub-block. The range of its contents is from 0 through 127. The contents of the register form the upper 7 bits of the 14-bit Sprite Attribute Table addresses; thus the base address is equal to (register 5) * 80 (hex).

XY YXXX YXXX XXXX 0-128

2.4.7 Register 6

Register 6 defines the base address of the Sprite Pattern Generator sub-block. The range of its contents is 0 through 7. The contents of the register form the upper 3 bits of the 14-bit Sprite Pattern Generator addresses; thus the Sprite Pattern Generator base address is equal to (register 6) * 800 (hex).

XY XXXX XXXX XXXX 0-2047
0-7FF

2.4.8 Register 7

The upper 4 bits of register 7 contain the color code of color 1 in the Text mode. The lower 4 bits contain the color code for color 0 in the Text mode and the backdrop color in all modes. See Table 3 for color codes.

2.5 STATUS REGISTER

The VDP has a single 8-bit status register that can be accessed by the CPU. The status register contains the interrupt pending flag, the sprite coincidence flag, the fifth sprite flag, and the fifth sprite number, if one exists. The format of the status register is shown in Figure 2-5. A discussion of the contents follows.

The status register may be read at any time to test the F, C, and 5S status bits. Reading the status register will clear the interrupt flag, F. Asynchronous reads will, however, cause the frame flag (F) bit to be reset and therefore missed. Consequently, the status register should be read only when the VDP interrupt is pending.

2.5.1 Interrupt Flag (F)

The F status flag in the status register is set to '1' at the end of the raster scan of the last line of the active display. It is reset to a '0' after the status register is read or when the VDP is externally reset. If the Interrupt Enable bit in VDP register 1 is active ('1'), the VDP interrupt output (\overline{INT}) will be active (low) whenever the F status flag is a '1'.

2.5.2 Coincidence Flag (C)

The C status flag in the status register is set to a '1' if two or more sprites "coincide". Coincidence occurs if any two sprites on the screen have one or more overlapping pixels. Transparent colored sprites, as well as those that are partially or completely off the screen, are also considered. Sprites beyond the Sprite Attribute Table terminator (D016) are not considered. The 'C' flag is cleared to a '0' after the status register is read or the VDP is externally reset.

2.5.3 Fifth Sprite Flag (5S) and Number

The 5S status flag in the status register is set to a '1' whenever there are five or more sprites on a horizontal line (lines 0 to 192) and the frame flag is equal to a '0'. The 5S status flag is cleared to a '0' after the status register is read or the VDP is externally reset. The number of the fifth sprite is placed into the lower 5 bits of the status register when the 5S flag is set and is valid whenever the 5S flag is '1'. The setting of the fifth sprite flag will not cause an interrupt.

2.6 OSCILLATOR AND CLOCK GENERATION

The VDP is designed to operate with a 10.738635 megahertz ± 50 ppm crystal input to generate the required internal clock signals. A fundamental frequency, parallel-mode crystal is used as the frequency reference for the internal clock oscillator, which is the master time base for all system operations. This master clock is divided by two to generate the pixel clock (5.3 megahertz) and by three to provide the CPUCLK (3.58 megahertz). The GROMCLK is developed from the master clock frequency divided by 24.

2.6.1 Color Phase Generation

The 10.7+ megahertz master clock and its complement are used to generate an internal six-phase 3.579545 megahertz (± 10 hertz) clock to provide the video color signals and the color burst reference for use in developing the composite video output signal. While the VDP signals are not exact equivalents to the standard NTSC colors, the differences can easily be adjusted with the color and tint controls of the target color television.

2.6.2 Video Sync and Control Generation

The vertical and horizontal control signals are generated by decoding the outputs of the horizontal and vertical counters. The horizontal counter is driven by the pixel clock, and the horizontal counter in turn increments the vertical counter. Table 2 gives the relative count values of the screen display parameters. Within the active display area during Graphics I mode, the 3 least-significant bits of the horizontal counter address the individual picture element of each pattern displayed. Also, during the vertical active display period, the 3 least-significant bits of the vertical counter address each individual line in the 8 X 8 patterns. The Graphics II, Multicolor and Text modes use the counters similarly.

The VDP operates at 262 lines per frame and approximately 60 frames per second in a non-interlaced mode of operation.

TABLE 2 – SCREEN DISPLAY PARAMETERS

PARAMETER	PIXEL CLOCK CYCLES	
	PATTERN OR MULTICOLOR	TEXT
HORIZONTAL		
HORIZONTAL ACTIVE DISPLAY	256	240
RIGHT BORDER	15	25
RIGHT BLANKING	8	8
HORIZONTAL SYNC	26	26
LEFT BLANKING	2	2
COLOR BURST	14	14
LEFT BLANKING	8	8
LEFT BORDER	13	19
	342	342
VERTICAL	LINE	
VERTICAL ACTIVE DISPLAY	192	
BOTTOM BORDER	24	
BOTTOM BLANKING	3	
VERTICAL SYNC	3	
TOP BLANKING	13	
TOP BORDER	27	
	262	

2.7 VIDEO DISPLAY MODES

The VDP displays an image on the screen that can best be envisioned as a set of display planes sandwiched together. Figure 2-6a shows the definition of each of the planes. Objects on planes closest to the viewer have higher priority. In cases where two entities on two different planes are occupying the same spot on the screen, the entity on the higher priority plane will show at that point. For an entity on a specific plane to show through, all planes in front of that plane must be transparent at that point. The first 32 planes (Figure 2-6b) each may contain a single sprite. (Sprites are pattern objects whose positions on the screen are defined by horizontal and vertical coordinates in VRAM.) The areas of the Sprite Planes, outside of the sprite itself, are transparent. Since the coordinates of the sprite are in terms of pixels, the sprite can be positioned and moved about very accurately. Sprites are available in three sizes: 8 X 8 pixels, 16 X 16 pixels, and 32 X 32 pixels. Behind the Sprite Plane is the Pattern Plane. The Pattern Plane is used for textual and graphics images generated by the Text, Graphics I, Graphics II, or Multicolor modes. Behind the Pattern Plane is the backdrop, which is larger in area than the other planes so that it forms a border around the other planes. The last and lowest priority plane is the External Video Plane. Its image is defined by the external video input pin. The backdrop consists of a single color used for the display borders and as the default color for the active display area. The default color is stored in the VDP register 7. When the backdrop color register contains the transparent code, the backdrop automatically defaults to black if the external video mode is not selected.

The 32 Sprite Planes are used for the 32 sprites in the Multicolor and Graphics modes. They are not used in the Text mode and are automatically transparent. Each of the sprites can cover an 8 X 8, 16 X 16, or 32 X 32 pixel area on its plane. Any part of the plane not covered by the sprite is transparent. All or part of each sprite may also be transparent. Sprite 0 is on the outside or highest plane, and sprite 31 is on the plane immediately adjacent to Pattern Plane. Whenever a pixel in a Sprite Plane is transparent, the color of the next plane can be seen through that plane. If, however, the sprite pixel is non-transparent, the colors of the lower planes are automatically replaced by the sprite color. There is also a restriction on the number of sprites on a line. Only four sprites can be active on any horizontal line. Additional sprites on a line will be automatically made transparent for that line. Only those sprites that are active on the display will cause the coincidence flag to set. The VDP status register provides a flag bit and the number of the fifth sprite whenever this occurs. The Pattern Plane is used in the Text, Multicolor, and Graphics modes for display of the graphic patterns of characters. Whenever a pixel on the Pattern Plane is non-transparent, the backdrop color is automatically replaced by the Pattern Plane color. When a pixel in the Pattern Plane is transparent, the backdrop color can be seen through the Pattern Plane.

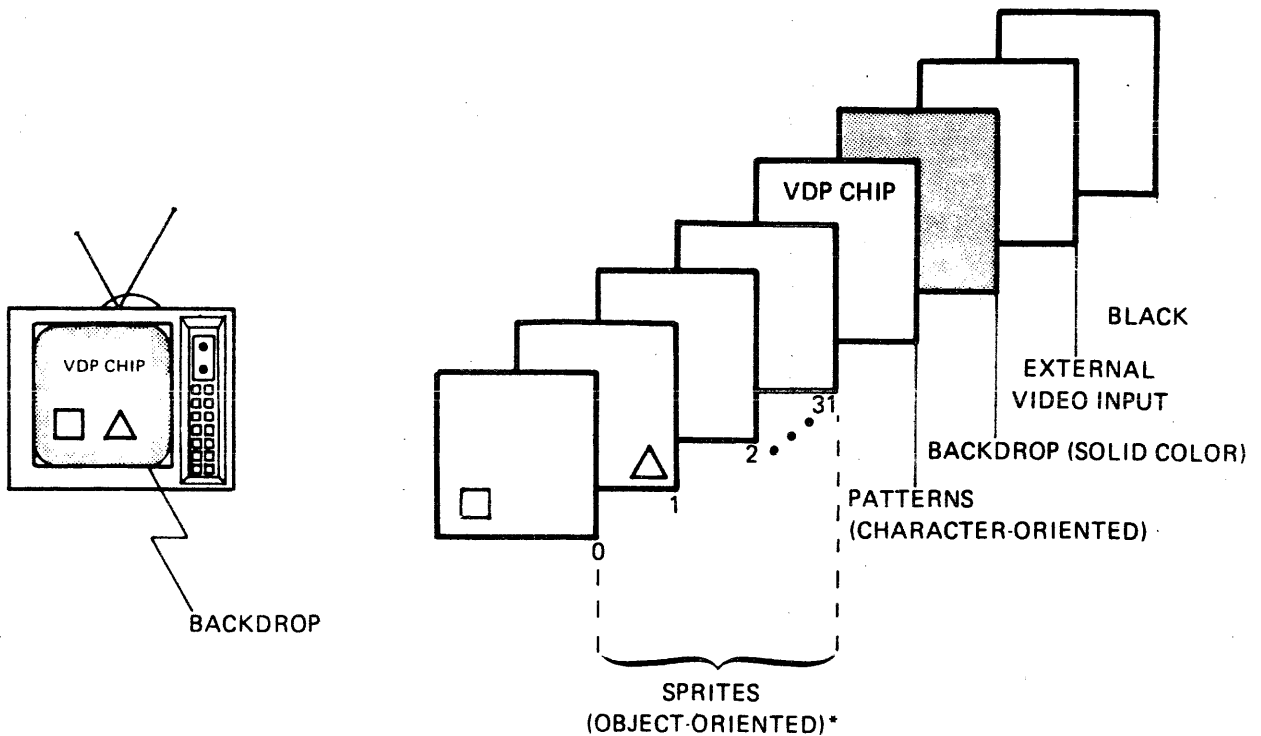


FIGURE 2-6a – VDP DISPLAY PLANES

The VDP has four video color display modes that appear on the Pattern Plane: Graphics I mode, Graphics II mode, Text mode, and Multicolor mode. Graphics I and Graphics II modes cause the Pattern Plane to be broken up into groups of 8 X 8 pixels, called pattern positions. Since the full image is 256 X 192 pixels, there are 32 X 24 pattern positions on the screen in the graphics modes. In Graphics I mode, 256 possible patterns may be defined for the 768 pattern positions with two unique colors allowed for each pattern definition. Graphics II mode provides, through a unique mapping scheme, 768 pattern definitions for the 768 pattern positions. Graphics II mode also allows the selection of two unique colors for each line of a pattern definition. Thus, all 15 colors plus transparent may be used in a single pattern position. In Text mode, the Pattern Plane is broken into groups of 6 X 8 pixels, called text positions. There are 40 X 24 text positions on the screen in this mode. In Text mode, sprites do not appear on the screen and two colors are defined for the entire screen. In Multicolor mode, the screen is broken into a grid of 64 X 48 positions, each of which is a 4 X 4 pixel. Within each position, one unique color is allowed.

The VDP registers define the base addresses for several sub-blocks within VRAM. These sub-blocks form tables which are used to produce the desired image on the TV screen. The Pattern Name Table, the Pattern Generator Table and the Sprite Generator Table are used to form the sprites. The contents of these tables must all be provided by the microprocessor. Animation is achieved by altering the contents of VRAM in real time.

The VDP can display the 15 colors shown in Table 3. The VDP colors also provide eight different gray levels for displays on monochrome televisions; the luminance values in the table indicate these levels, 0.00 being black and 1.00 being white. Whenever all planes are of the transparent color at a given point, the color shown at that point will be black.

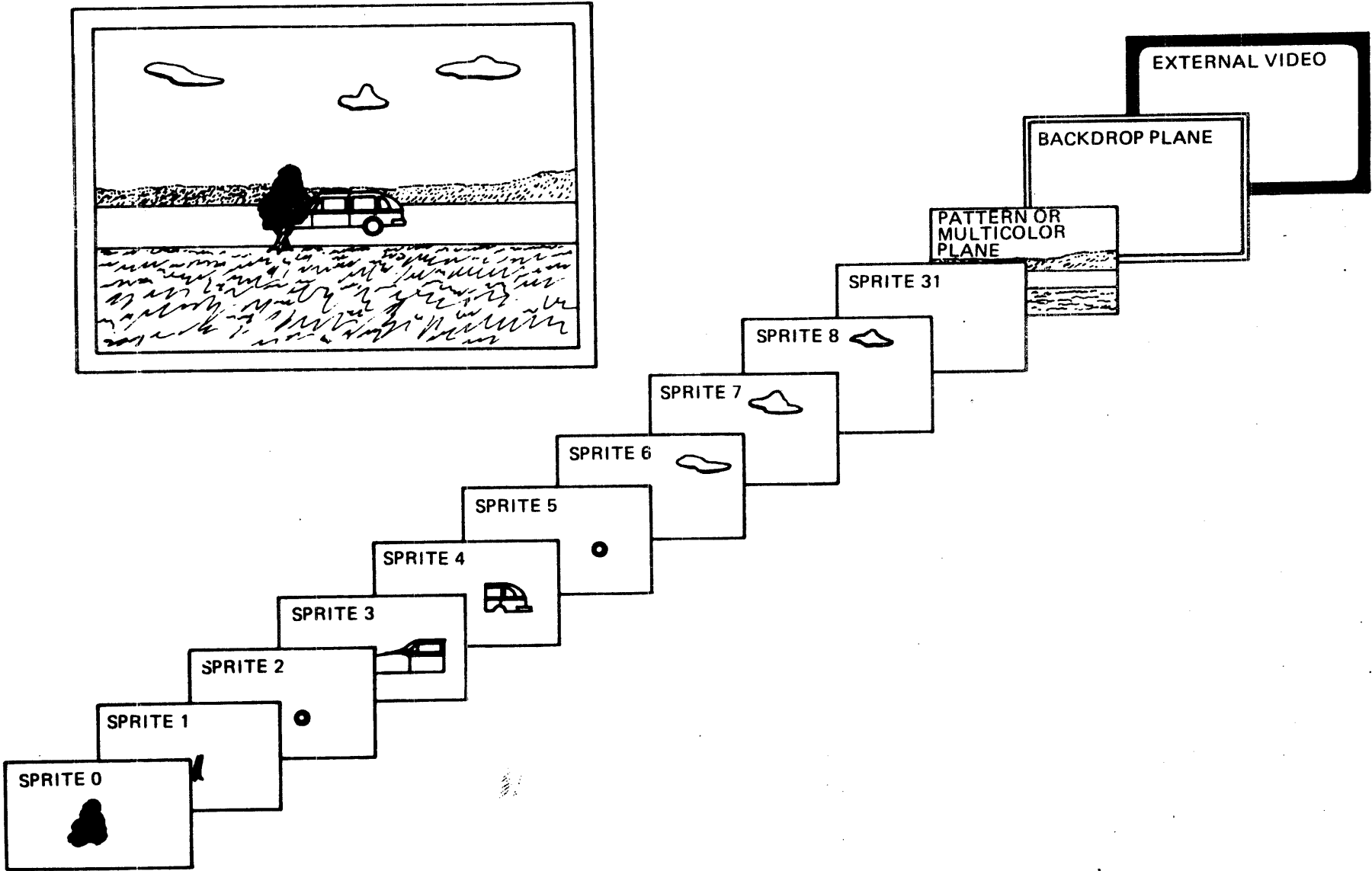


FIGURE 2-6b - VDP DISPLAY PLANES

TABLE 3. COLOR ASSIGNMENTS

COLOR (HEX)	COLOR	LUMINANCE (DC VALUE)	CHROMINANCE (AC VALUE)
0	TRANSPARENT	0.00	—
1	BLACK	0.00	—
2	MEDIUM GREEN	.60	.60
3	LIGHT GREEN	.80	.53
4	DARK BLUE	.47	.73
5	LIGHT BLUE	.67	.60
6	DARK RED	.53	.53
7	CYAN	.80	.73
8	MEDIUM RED	.67	.73
9	LIGHT RED	.80	.73
A	DARK YELLOW	.87	.53
B	LIGHT YELLOW	1.00	.40
C	DARK GREEN	.47	.60
D	MAGENTA	.60	.47
E	GRAY		—
F	WHITE	1.00	—
—	BLACK LEVEL	0.00	—
—	COLOR BURST	0.00	.40
—	SYNC LEVEL	-0.40	—

2.7.1 Graphics I Mode

The VDP is in Graphics I mode when M1, M2, and M3 bits in VDP registers 1 and 0 are zero. In Graphics I mode the Pattern Plane is divided into a grid of 32 columns by 24 rows of pattern positions (see Figure 2-7). Each of the pattern positions contains 8 X 8 pixels. The tables in VRAM used to generate the Pattern Plane are the Pattern Color Table. Figure 2-8 illustrates the mapping of these tables into the Pattern Plane. A total of 2848 VRAM bytes are required for the Pattern Name, Color and Generator tables. Less memory is required if all 256 possible pattern definitions are not required. The tables can be overlapped to reduce the amount of VRAM needed for pattern generation. Examples of VRAM memory allocation are provided in Section 3.

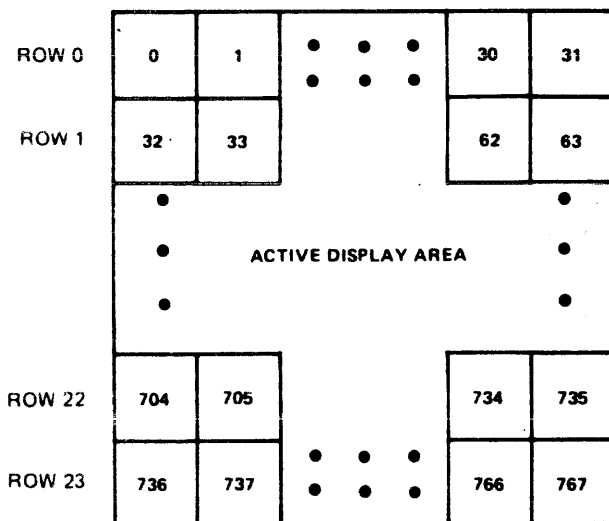


FIGURE 2-7 – PATTERN GRAPHICS NAME TABLE MAPPING

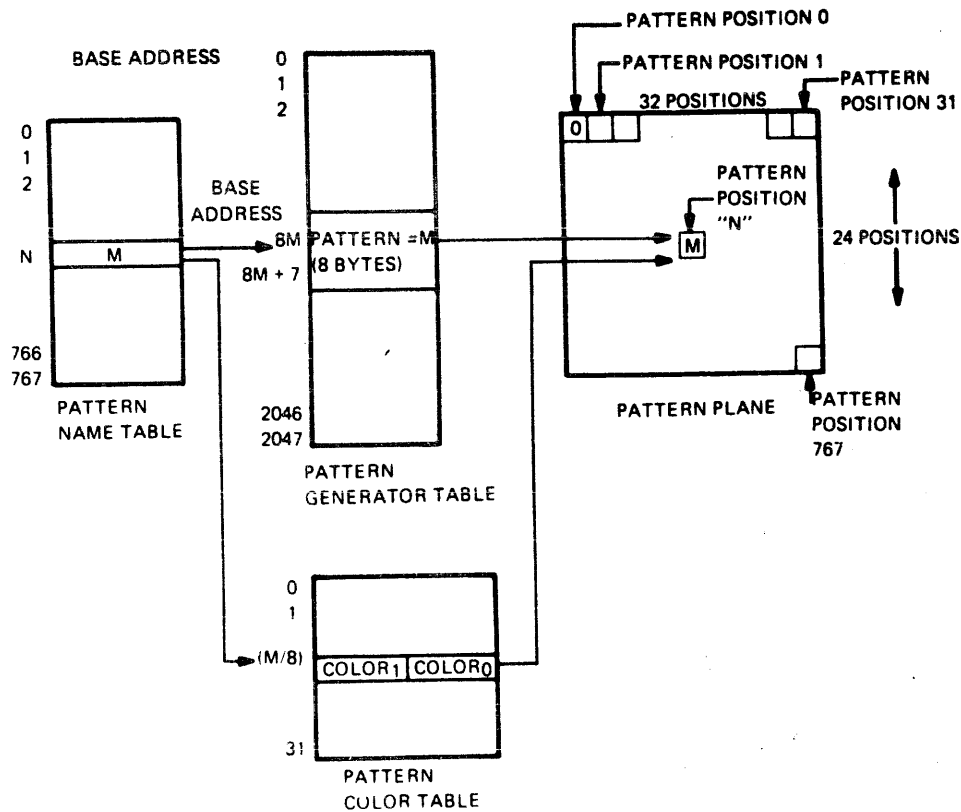


FIGURE 2-8 – PATTERN MODE MAPPING

The Pattern Generator Table contains a library of patterns that can be displayed in the pattern positions. It is 2048 bytes long, and is arranged into 256 patterns, each of which is eight bytes long, yielding 8 X 8 bits. All of the '1's in the eight-byte pattern can designate one color (color 1), while all the '0's can designate another color (color 0).

The full 8-bit pattern name is used to select one of the 256 pattern definitions in the Pattern Generator Table. The table is a 2048-byte block in VRAM beginning on a 2 kilobyte boundary. The starting address of the table is determined by the generator base address in VDP register 4. The base address forms the three most-significant bits of the 14-bit VRAM address for each Pattern Generator Table entry. The next 8 bits indicate the 8-bit name of the selected pattern definition. The lowest 3 bits of the VRAM address indicate the row number within the pattern definition.

Eight bytes are required for each of the 256 possible unique 8 X 8 pattern definitions. The first byte defines the first row of the pattern, and the second byte defines the second row. The first bit of each of the eight bytes define the first column of the pattern. The remaining rows and columns are similarly defined. Each bit entry in the pattern definition selects one of the two colors for that pattern. A '1' bit selects the color code (color 1) contained in the most-significant four bits of the corresponding color table byte. A '0' bit selects the other color code (color 0). An example of pattern definition mapping is provided in Figure 2-9.

ROW/BYTE	PATTERN					PATTERN DEFINITION							
	0	1	2	3	4	0	1	2	3	4	5	6	7
0	C	C	C	C	C	0	1	1	1	1	1	0	0
1					C	0	0	0	0	0	1	0	0
2					C	0	0	0	0	0	1	0	0
3		C	C	C	C	0	0	1	1	1	1	0	0
4					C	0	0	0	0	0	1	0	0
5					C	0	0	0	0	0	1	0	0
6	C	C	C	C	C	0	1	1	1	1	1	0	0
7						0	0	0	0	0	0	0	0

NOTES: VDP register 7 entry: 71₁₆.
Color code 7 is cyan (signified above by 'C').
Color code 1 is black (signified above by a space).
Bit 0 is the most significant bit of each data byte.

FIGURE 2-9 – PATTERN DISPLAY MAPPING

The color of the '1's and '0's is defined by the Pattern Color Table that contains 32 entries each of which is one byte long. Each entry defines two colors: the most-significant 4 bits of each entry define the color of the '1's, and the least-significant 4 bits define the color of the '0's. The first entry in the color table defines the colors for patterns 0 to 7; the next entry for patterns 8 to 15, and so on. (See Table 4 for assignments.) Thus, 32 different pairs of colors may be displayed simultaneously.

The Pattern Name Table is located in a contiguous 768-byte block in VRAM beginning on a 1 kilobyte boundary. The starting address of the Name Table is determined by the 4-bit Name Table base address field in VDP register 2. The base address forms the upper four bits of the 14-bit VRAM address. The lower 10 bits of the VRAM address are formed from the row and column counters. An example of pattern name table addressing is given in Section 3.

Each byte entry in the Name Table is the name of or the pointer to a pattern definition in the Pattern Generator Table. The upper five bits of the eight-bit name identify the color group of the pattern. There are 32 groups of eight patterns. The same two colors are used for all eight patterns in a group; the color codes are stored in the VDP Color Table. The Color Table is located in a 32-byte block in VRAM beginning on a 64-byte boundary. The table starting address is determined by the 8-bit Color Table base address in VDP register 3. The base address forms the upper eight bits of the 14-bit Color Table entry VRAM address. The next bit is a '0' and the lowest 5 bits are equal to the upper 5 bits of the corresponding Name Table entries. An example of Color Table addressing is provided in Section 3.

Since the tables in VRAM have their base addresses defined by the VDP registers, a complete switch of the values in the tables can be made by simply changing the values in the VDP registers. This is especially useful when one wishes to time-slice between two or more screens of graphics.

When the Pattern Generator Table is loaded with a pattern set, manipulation of the Pattern Name Table contents can change the appearance of the screen. Alternatively, a dynamically changing set of patterns throughout the course of a graphics session is easily accomplished since all tables are in VRAM.

For textual applications, the desired character set is typically loaded into the Pattern Generator first. The official USASCII character set might be loaded into the Pattern Generator in such a way that the pattern numbers correspond to the 8-bit ASCII codes for that pattern; e.g., the pattern for the letter "A" would be loaded into pattern number 41₁₆ in the Pattern Generator. Next the Pattern Color Table would be loaded up with the proper color set. To print a textual message on the screen, write the proper ASCII codes out to the Pattern Name Table.

Images can be formed using the Pattern Plane. To display an object of size 8 X 8 pixels or smaller, only one pattern would need to be defined. To display a larger figure, the figure should be broken up into smaller 8 X 8 squares. Then multiple patterns can be defined, and the Pattern Generator and Pattern Name Table set up appropriately. Note that rough motion of objects requires merely updating entries in the Pattern Name Table.

TABLE 4. PATTERN COLOR TABLE

Byte No.	Pattern No.
0	0..7
1	8..15
2	16..23
3	24..31
4	32..39
5	40..47
6	48..55
7	56..63
8	64..71
9	72..79
10	80..87
11	88..95
12	96..103
13	104..111
14	112..119
15	120..127
16	128..135
17	136..143
18	144..151
19	152..159
20	160..167
21	168..175
22	176..183
23	184..191
24	192..199
25	200..207
26	208..215
27	216..223
28	224..231
29	232..239
30	240..247
31	248-255

A total of 2848 VRAM bytes are required for the Pattern, Name, Color and Generator tables. Less memory is needed if all 256 possible pattern definitions are not required; the tables can be overlapped to reduce the amount of VRAM needed for pattern generation. Examples of VRAM memory allocation are provided in Section 3.

2.7.2 Graphics II Mode

The VDP is in the Graphics II mode when mode bits M1 = 0, M2 = 0, and M3 = 1. The Graphics II mode is similar to Graphics I mode except it allows a larger library of patterns so that a unique pattern generator entry may be made for each of the 768 (32 X 24) pattern positions on the video screen. Additionally, more color information is included in each 8 X 8 graphics pattern. Thus two unique colors may be specified for each byte of the 8 X 8 pattern. A larger amount of VRAM (12 kilobytes) is required to implement the full usage of the Graphics II mode.

Like Graphics I mode, the Graphics II mode Pattern Name Table contains 768 entries which correspond to the 768 pattern positions on the display screen. Because the Graphics I mode pattern names are only 8 bits in length, a maximum of 256 pattern definitions may be addressed using the addressing scheme discussed in the previous section. Graphics II mode, however, segments the display screen into three equal parts of 256 pattern positions each and also segments the Pattern Generator Table into three equal blocks of 2048 bytes each. Pattern definitions in the first third correspond to pattern positions in the upper third of the display screen. Likewise pattern definitions in the second and third blocks of the Pattern Generator Table correspond to the second and third areas of the Pattern Plane. The pattern Name Table is also segmented into three blocks of 256 names each so that names found in the upper third, reference pattern definitions found in the upper 2048 bytes in Pattern Generator Table. Likewise the second and third blocks reference pattern definitions in the second 2048 byte block and third 2048 byte block respectively. Thus, if 768 patterns are uniquely specified, an 8-bit pattern name will be used three times, once in each segment of the Pattern Name Table. The Pattern Generator Table falls on eight kilobyte boundaries and may be located in the upper or lower half of 16K memory based on the MSB of the pattern generator base in VDP register 4. The LSB's must be set to all '1's.

The ColorTable is also 6144 bytes long and is segmented into three equal blocks of 2048 bytes. Each entry in the Pattern Color Table is eight bytes which provides the capability to uniquely specify color 1 and color 0 for each of the eight bytes of the corresponding pattern definition. The addressing scheme is exactly like that of the Pattern Generator Table except for the location of the table in VRAM. This is controlled by the loading of the MSB of the color base in VDP register 3. The LSB's must be set to all '1's.

Figure 2-10 is an example of the Graphics II mode mapping scheme. Note that pattern names P1, P2, P3 correspond to pattern generator entries in the three blocks of the Pattern Generator Table. Note also how these three names map to the display screen. Figure 2-11 an example of a Pattern Generator and Pattern Color Table entry.

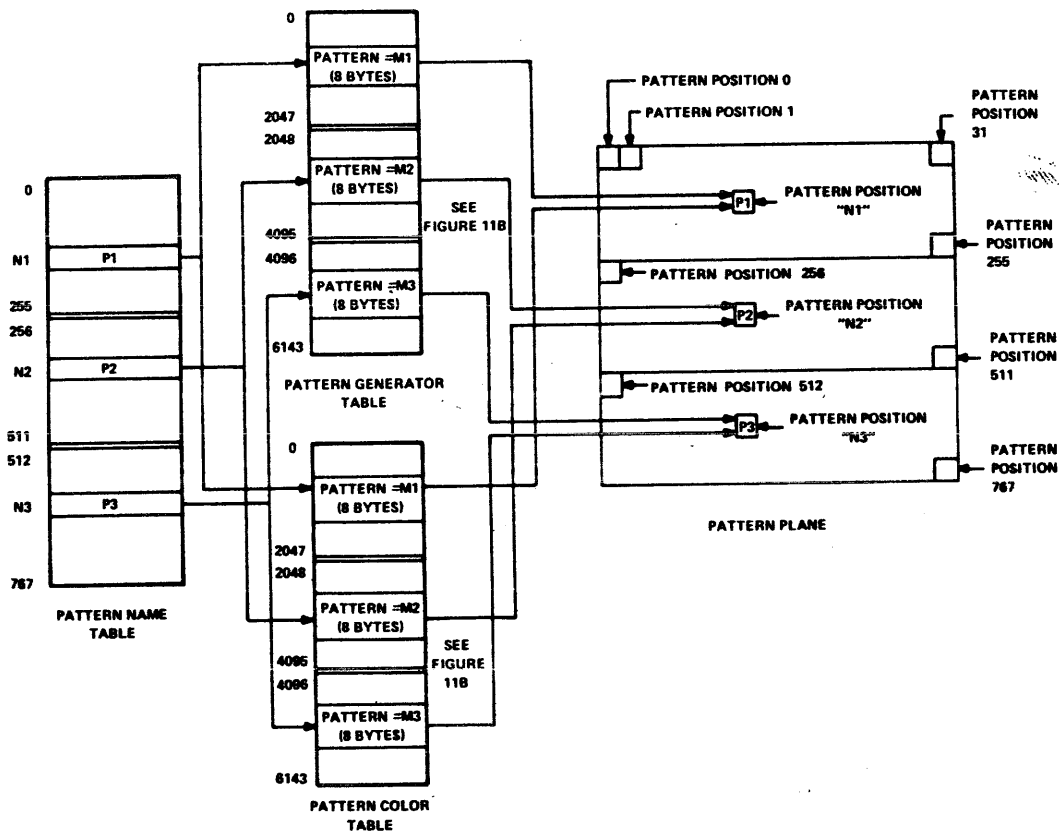


FIGURE 2-10 – GRAPHICS II MODE MAPPING

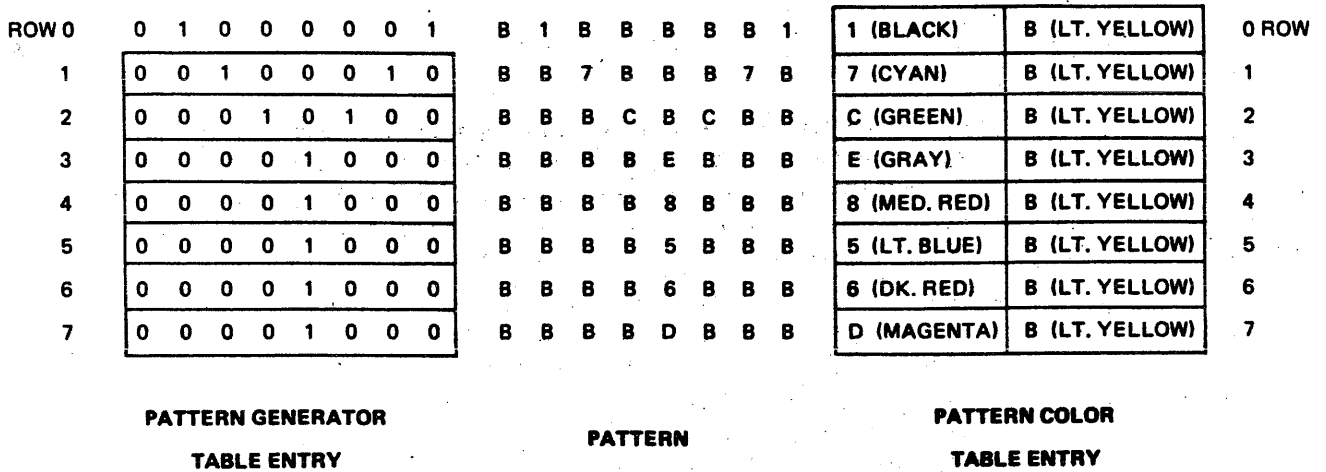


FIGURE 2-11 – PATTERN DISPLAY MAPPING

2.7.3 Multicolor Mode

The VDP is in Multicolor mode when mode bits M1 = 0, M2 = 1, and M3 = 0. Multicolor mode provides an unrestricted 64 X 48 color square display. Each color square contains a 4 X 4 block of pixels. The color of each of the color squares can be any one of the 15 video display colors plus transparent. Consequently, all 15 colors can be used simultaneously in the Multicolor mode. The Backdrop and Sprite Planes are still active in the Multicolor mode.

The Multicolor Name Table is the same as that for the graphics modes, consisting of 768 name entries. The name no longer points to a color list; rather color is now derived from the Pattern Generator Table. The name points to an eight-byte segment of VRAM in the Pattern Generator Table.

Only two bytes of the eight-byte segment are used to specify the screen image. These two bytes specify four colors, each color occupying a 4 X 4 pixel area. The four MSB's of the first byte define the color of the upper left quarter of the multicolor pattern; the LSB's define the color of the upper right quarter. The second byte similarly defines the lower left and right quarters of the multicolor pattern. The two bytes thus map into a 8 X 8 pixel multicolor pattern. (See Figure 2-12).

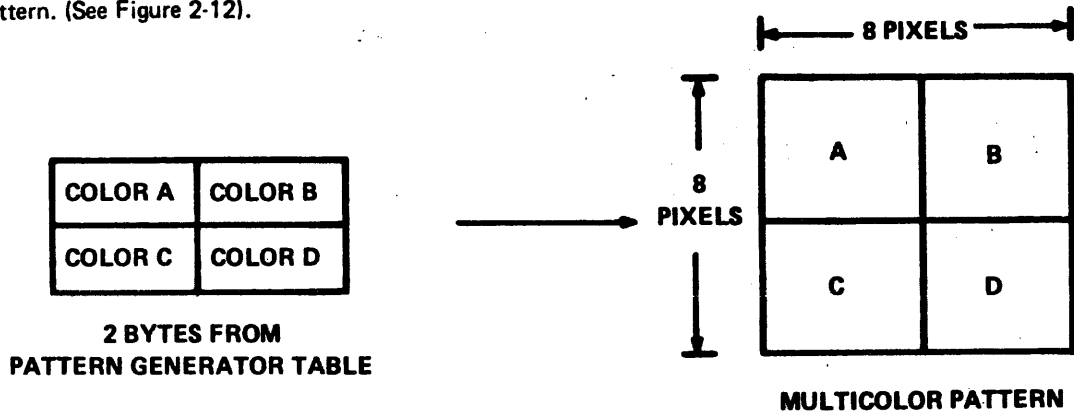


FIGURE 2-12 – MULTICOLOR LIST MAPPING

The location of the two bytes within the eight-byte segment pointed to by the name is dependent upon the screen position where the name is mapped. For names in the top row (names 0-31), the two bytes are the first two within the groups of eight-byte segments pointed to by the names. The next row of names (32-63) uses the third and fourth bytes within the eight-byte segments. The next row of names uses the fifth and sixth bytes while the last row of names uses the seventh and eighth. This series repeats for the remainder of the screen.

For example, referring to Figure 2-13, if Name Table entry 0 (pattern position 0) multicolor block #N (name = N), the multicolor pattern displayed will be an 8 X 8 pixel block consisting of colors A, B, C, D comprising the first two bytes of the Multicolor Table. If, however, name #N is located in Name Table entry 33, (Pattern position 33), the colors displayed will be colors E, F, G, H as specified by the third and fourth bytes of the multicolor block pointed to by the name. Likewise pattern positions which lie in rows 2 and 3 would cause colors I, J, K, L and colors M, N, O, P respectively to be displayed. Thus it can be seen that the color displayed from the multicolor generator block is dependent upon pattern position on the screen. Figure 2-14 illustrates the multicolor mode mapping scheme.

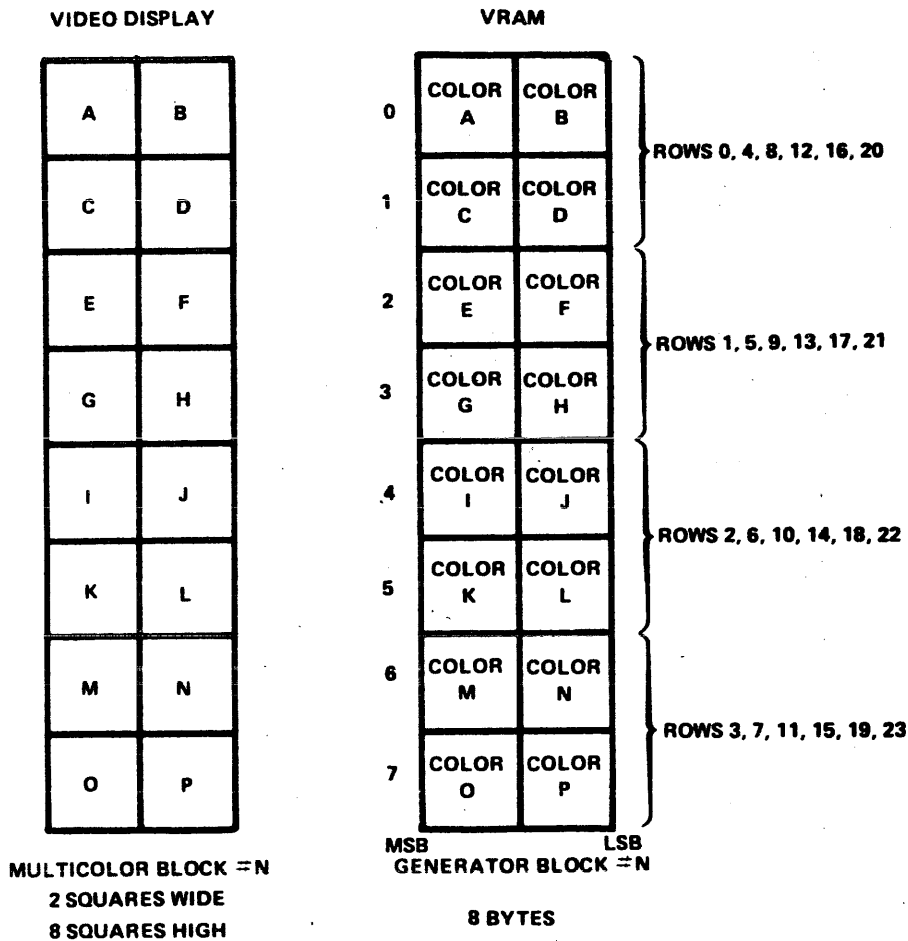


FIGURE 2-13 – MULTICOLOR BLOCK DISPLAY

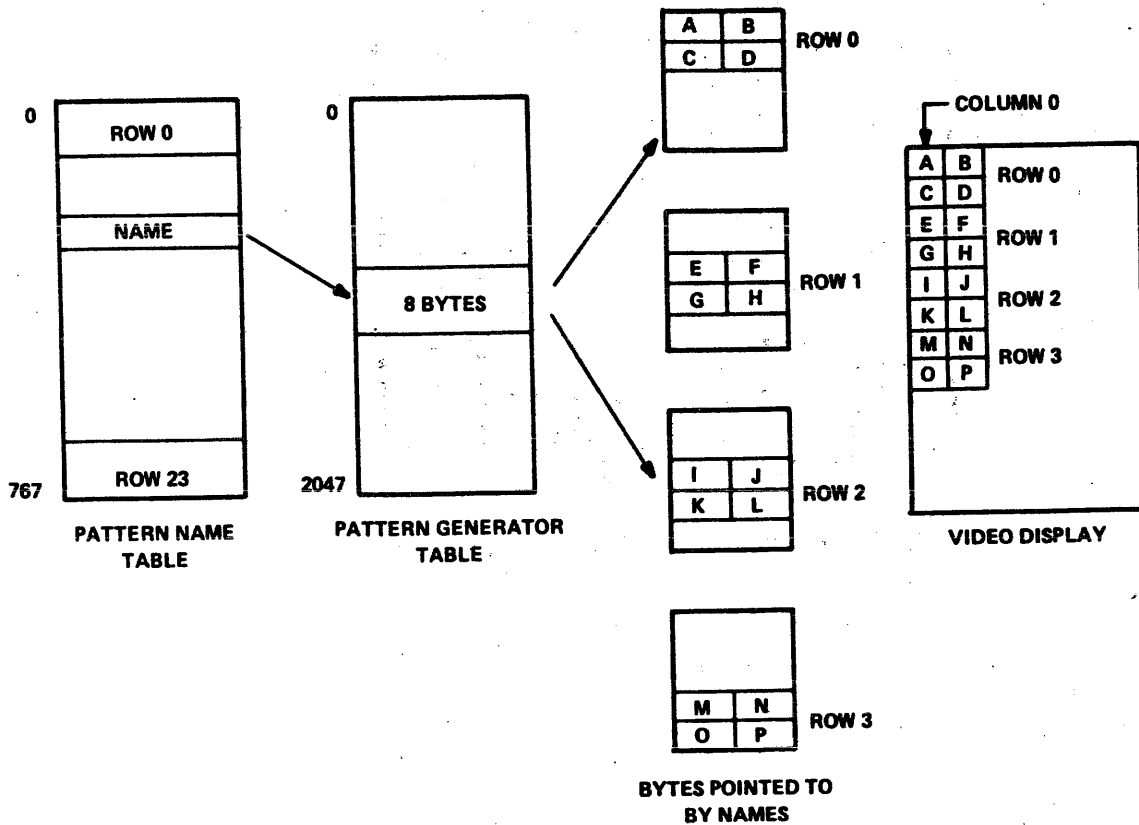


FIGURE 2-14 – MULTICOLOR MODE MAPPING

The mapping of VRAM contents to screen image is simplified by using duplicate names in the Name Table. Since the series of bytes used within the eight-byte segment repeats every four rows, the four rows in the same column can use the same name. Then the eight-byte segment specifies a 2 X 8 color square pattern on the screen as a straightforward translation from the eight-byte segment in VRAM pointed to by the common name.

When used in this manner, 768 bytes are still used for the Name Table and 1536 bytes are used for the color information in the Pattern Generator Table (24 rows X 32 columns X 8 bytes/pattern position). Thus a total of 1728 bytes in VRAM are required. It should be noted that the tables begin on even 1K and 2K boundaries and are therefore not contiguous. An example of multicolor VRAM memory allocation is provided in Section 3.

2.7.4 Text Mode

The VDP is in Text mode when mode bits M1 = 1, M2 = 0, and M3 = 0. In the Text mode, the screen is divided into a grid of 40 text positions across and 24 down. (See Figure 2-15). Each of the text positions contains six pixels across and eight pixels down. The tables used to generate the Pattern Plane are the Pattern Name Table and the Pattern Generator Table. There can be up to 256 unique patterns defined at any time. The pattern definitions are stored in the Pattern Generator Table in VRAM and can be dynamically changed. The VRAM contains a Pattern Name Table which maps the pattern definitions into each of the 960 pattern cells on the Pattern Plane (Figure 2-16). Sprites are not available in Text mode.

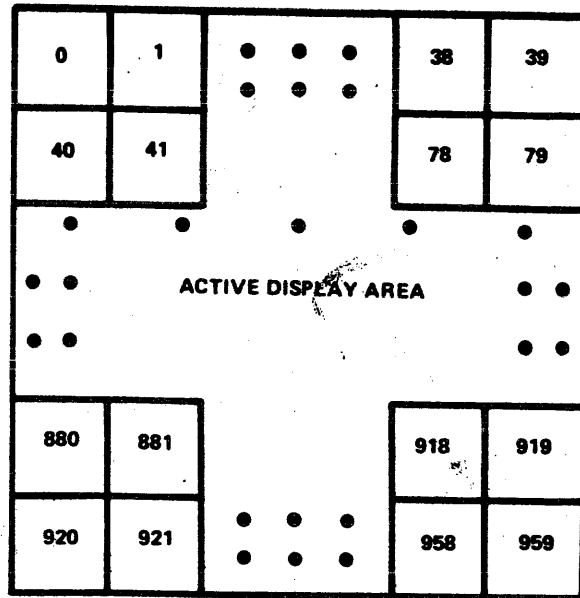


FIGURE 2-15 – TEXT MODE NAME TABLE PATTERN POSITIONS

As in the case of the Graphics modes, the Pattern Generator Table contains a library of text patterns that can be displayed in the text positions. It is 2048 bytes long, and is arranged in 256 text patterns, each of which is eight bytes long. Since each text position on the screen is only six pixels across, the least-significant 2 bits of each text pattern are ignored, yielding 6 X 8 bits in each text pattern. Each block of eight bytes defines a text pattern in which all the '1's in the text pattern take on one color when displayed on the screen, while all the '0's take on another color. These colors are chosen by loading VDP register 7 with the color 1 and color 0 in the left and right nibbles respectively (see Section 2.4).

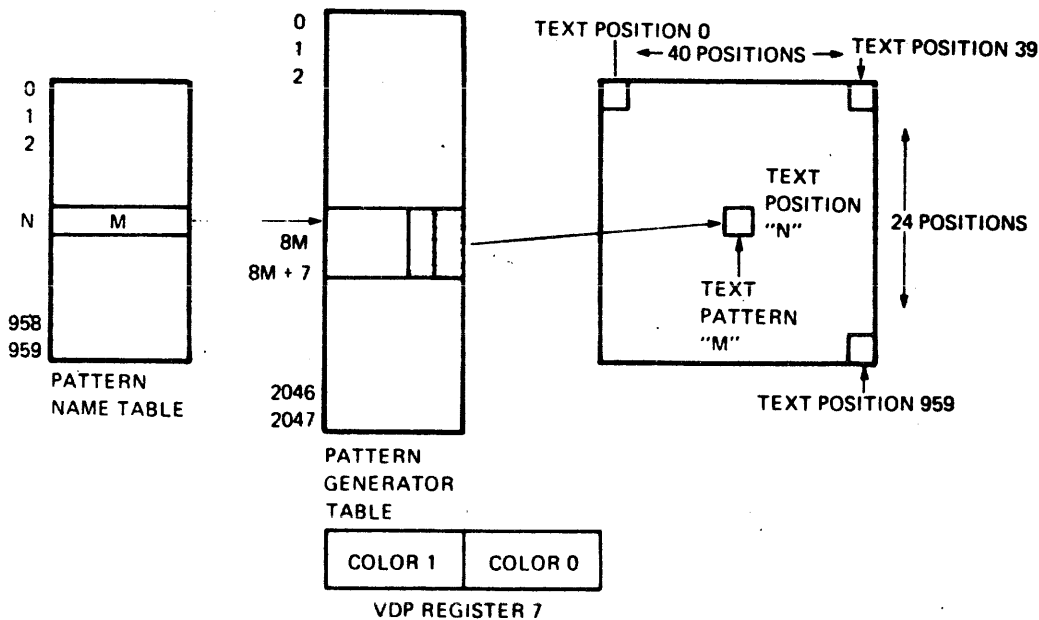


FIGURE 2-16 – MAPPING OF VRAM INTO THE PATTERN PLANE IN TEXT MODE

In the Text mode, the Pattern Name Table determines the position of the text pattern on the screen. There are 960 entries in the Pattern Name Table, each one byte long. There is a one-to-one correspondence between text pattern positions on the screen and entries in the Pattern Name Table ($40 \times 24 = 960$). The first 40 entries corresponds to the top row of text pattern positions on the screen, the next forty to the second row, and so on. The value of an entry in the Pattern Name Table indicates which of the 256 text patterns is to be placed at that spot on the Pattern plane. The Pattern Name Table is located in a contiguous 960-byte block in VRAM beginning on a 1 kilobyte boundary. The starting address of the name table is determined by the 4-bit Name Table base address field in VDP register 2. The base address forms the upper 4 bits of the 14-bit VRAM address. The lower 10 bits of the VRAM address point to one of 960 pattern cells. The name table is organized by rows. An example of Pattern Name Table addressing is given in Section 3. Each byte entry in the name table is the pointer to a pattern definition in the Pattern Generator Table. The same two colors are used for all 256 patterns; the color codes are stored in VDP register 7.

As its name implies, the Text mode is intended mainly for textual applications, especially those in which the 32 patterns-per-line in Graphics modes is insufficient. The advantage is that eight more patterns can be fitted onto one line; the disadvantages are that sprites cannot be used, and only two colors are available for the entire screen. With care, the same text pattern set that is used in Text mode can be also used in Graphics I mode. This is done by ensuring that the least-significant 2 bits of all the character patterns are '0'. A switch from Text mode to Pattern mode, then, results in a stretching of the space between characters, and a reduction of the number of characters per line from 40 to 32. As with the Graphics Modes, once a character set has been defined and placed into the Pattern Generator, updating the Pattern Name Table will produce and manipulate textual material on the screen.

The full 8-bit pattern name is used to select one of the 256 pattern definitions in the pattern generator table. The table is a 2048-byte block in VRAM beginning on a 2 kilobyte boundary. The starting address of the table is determined by the generator base address in VDP register 4. The base address forms the 3 most-significant bits of the 14-bit VRAM address for each Pattern Generator Table entry. The next 8 bits are equal to the 8-bit name of the selected pattern definition. The lowest 3 bits of the VRAM address are equal to the row number within the pattern definition.

Eight bytes are required for each of the 256 possible unique 6 X 8 pattern definitions. The first byte defines the first row of the pattern, and the second byte defines the second row. The two least-significant bits in each byte are not used. It is, however, strongly recommended that these bits be '0's. Each bit entry in the pattern definition selects one of the two colors for that pattern. A '1' bit selects the color code (color 1) contained in the most-significant 4 bits of VDP register 7. A '0' bit selects the other color code (color 0) which is in the least-significant 4 bits of the same VDP Register. An example of pattern definition mapping is provided in Figure 2-16.

A total of 3005 VRAM bytes are required for the Pattern Name and Generator Tables. Less memory is required if all 256 possible pattern definitions are not required; the tables can be overlapped to reduce the amount of VRAM needed for pattern generation. Examples of VRAM memory allocation are provided in Appendix B.

2.7.5 Sprites

The video display can have up to 32 sprites on the highest priority video planes. The sprites are special animation patterns which provide smooth motion and multilevel pattern overlaying. The location of a sprite is defined by the top left-hand corner of the sprite pattern. The sprite can be easily moved pixel-by-pixel by redefining the sprite origin. This provides a simple but powerful method of quickly and smoothly moving special patterns. The sprites are not active in the Text mode. The 32 Sprite Planes are fully transparent outside of the sprite itself.

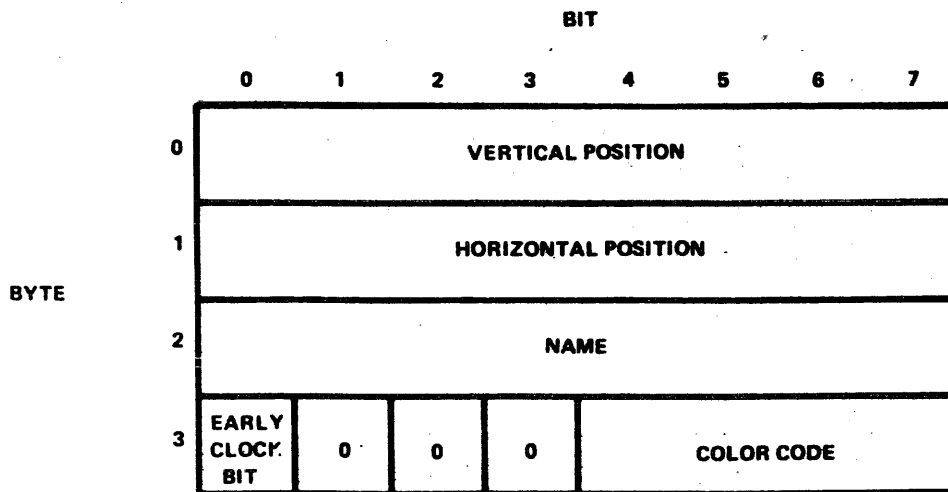


FIGURE 2-17 – SPRITE ATTRIBUTE TABLE ENTRY

The sub-blocks in VRAM that define sprites are the Sprite Attribute Table (see example of entry in Figure 2-17) and the Sprite Generator Table. These tables are similar to their equivalents in the pattern realm in that the Sprite Attribute Table specifies where the sprite goes on the screen, while the Sprite Generator Table describes what the sprite looks like. Sprite Pattern formats are given in Table 5.

Figure 2-18 illustrates the manner in which the VRAM tables map into the existence of sprites on the display. Since there are 32 sprites available for display, there are 32 entries in the Sprite Attribute Table. Each entry consists of four bytes. The entries are ordered so that the first entry corresponds to the sprite on the sprite 0 plane, the next to the sprite on the sprite 1 plane, and so on. The Sprite Attribute Table is $4 \times 32 = 128$ bytes long. The Sprite Attribute Table is located in a contiguous 128-byte block in VRAM beginning on a 128-byte boundary. The starting address of the Attribute Table is determined by the 7-bit Sprite Attribute Table base address in VDP register 5. The base address forms the upper seven bits of the 14-bit VRAM address. The next 5 bits of the VRAM address are equal to the sprite number. The lowest 2 bits select one of the four bytes in the Attribute Table entry for each sprite. Each Sprite Attribute Table entry contains four bytes which specify the sprite position, sprite pattern name, and color as shown in Figure 2-17.

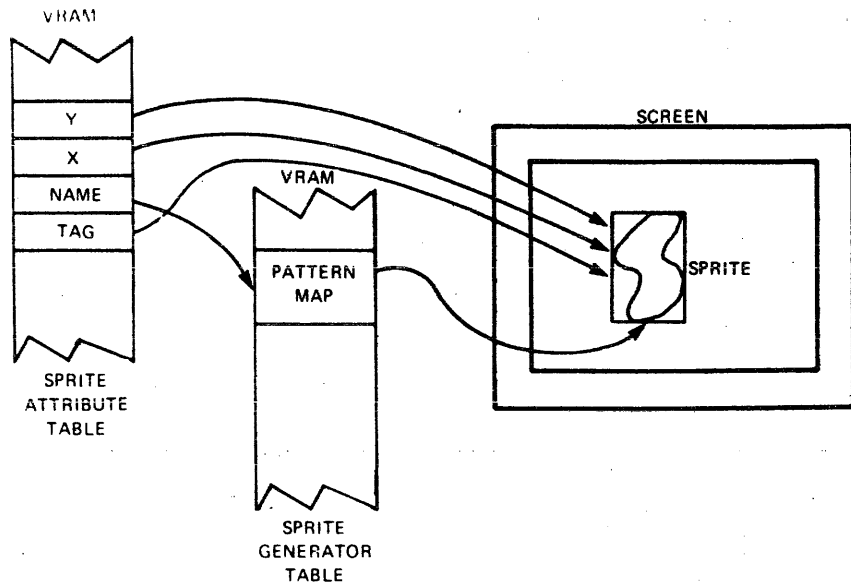


FIGURE 2-18 – SPRITE MAPPING

TABLE 5. SPRITE PATTERN FORMATS

SIZE	MAG	AREA	RESOLUTION	BYTES/PATTERN
0	0	8 × 8	single pixel	8
1	0	16 × 16	single pixel	32
0	1	16 × 16	2 × 2 pixels	8
1	1	32 × 32	2 × 2 pixels	32

The first two bytes of each entry of the Sprite Attribute Table determine the position of the sprite on the display. The first byte indicates the vertical distance of the sprite from the top of the screen, in pixels. It is defined such that a value of -1 puts the sprite butted up at the top of the screen, touching the backdrop area. The second bytes describes the horizontal displacement of the sprite from the left edge of the display. A value of 0 butts the sprite up against the left edge of the backdrop. Note that it is from the upper left pixel of the sprite that all measurement are taken.

When the first two bytes of an entry position a sprite overlapping the backdrop, the part of the sprite that is within the backdrop is displayed normally. The part of the sprite that overlaps the backdrop is hidden from view by the backdrop. This allows the animator to move a sprite into the display from behind the backdrop. The displacement in the first byte is partially signed, in that values for vertical displacement between -31 and 0 (E116 to 0) allow a sprite to "bleed in" from the top edge of the backdrop. Likewise, values in the range of 207 to 191 allow the sprite to bleed in from the bottom edge of the backdrop. Similarly, horizontal displacement values in the vicinity of 255 allow a sprite to bleed-in from the right side of the screen. To allow sprites to bleed-in from the left edge of the backdrop, a special bit in the third byte of the Sprite Attribute Table entry is used, as described in a later paragraph.

Byte 3 of the Sprite Attribute Table entry contains the pointer to the Sprite Generator Table that specifies what the sprite should look like. This is an 8-bit pointer to the sprite patterns definition, the Sprite Generator Table. The sprite name is similar to that in the Patterns Graphics mode.

Byte 4 of the Sprite Attribute Table entry contains the color of the sprite in its lower 4 bits (see Table 2 for color codes). The most-significant bit is the Early Clock bit (EC). This bit, when set to a '0', does nothing. When set to '1', the horizontal position of the sprite is shifted to the left by 32 pixels. This allows a sprite to bleed-in from the left edge of the backdrop. Values for horizontal displacement (byte 2 in the entry) in the range 0 to 32 cause the sprite to overlap with the left-hand border of the backdrop.

The Sprite Generator Table is a maximum of 2048 bytes long beginning on the 2 kilobyte boundaries. It is arranged into 256 blocks of 8 bytes each. The third byte of the Sprite Attribute Table entry, then, specifies which eight byte block to use to specify a sprite's shape. The '1's in the Sprite Generator cause the sprite to be defined at that point; '0's cause the transparent color to be used. The starting address of the table is determined by the sprite generator base address in VDP register 6. The base address forms the 3 most-significant bits of the 14-bit VRAM address. The next 8 bits of the address are equal to sprite name, and the last 3 bits are equal to the row number within the sprite pattern. The address formation is slightly modified for SIZE₁ sprites.

There is a maximum limit of four sprites that can be displayed on one horizontal line. If this rule is violated, the four highest-priority sprites on the line are displayed normally. The fifth and subsequent sprites are not displayed on that line. Furthermore, the fifth-sprite bit in the VDP status register is set to a '1', and the number of the violating fifth sprite is loaded into the status register (see Section 2.5).

Larger sprites than 8 X 8 pixels can be used if desired. The MAG and SIZE bits in VDP register 1 are used to select the various options. The options are described here:

- MAG = 0, SIZE = 0: No options chosen;
- MAG = 1, SIZE = 0: Eight bytes are still used in the Sprite Generator Table to describe the sprite; however, each bit in the Sprite Generator maps into 2 X 2 pixels on the TV screen, effectively doubling the size of the sprite to 16 X 16.
- MAG = 0, SIZE = 1: 31 bytes are used in the Sprite Generator Table to define the sprite shape; the result is a 16 X 16 pixel sprite. The mapping of the 32 bytes into the sprite image is as shown in Figure 2-19. Mapping is still one bit-to-one pixel.
- MAG = 1, SIZE = 1: Same as MAG = 0, SIZE = 1 except each bit now maps into a 2 X 2 pixel area, yielding a 32 X 32 sprite.

The VDP provides sprite coincidence checking. The coincidence status flag in the VDP status register is set to a '1' whenever two active sprites have '1' bits at the same screen location.

Sprite processing is terminated if the VDP finds a value of 208 (D0₁₆) in the vertical position field of any entry in the Sprite Attribute Table. This permits the Sprite Attribute Table to be shortened to the minimum size required; it also permits the user to blank out part or all of the sprites by simply changing one byte in VRAM.

A total of 2176 VRAM bytes are required for the Sprite Name and Pattern Generator Tables. Significantly less memory is required if all 256 possible sprite pattern definitions are not required. The Sprite Attribute Table can also be shortened as described above. The tables can be overlapped to reduce the amount of VRAM required for sprite generation. Examples of VRAM memory allocation are provided in Section 3.

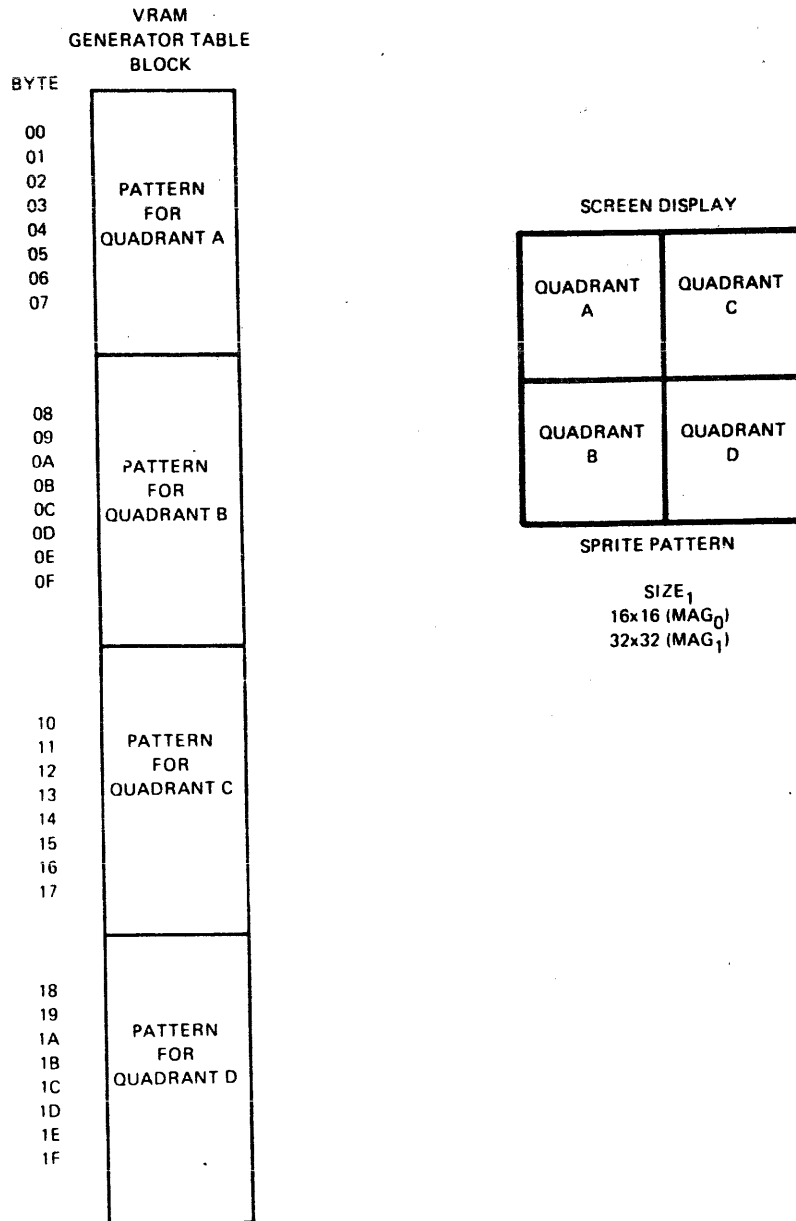


FIGURE 2-19 – SIZE 1 SPRITE MAPPING

2.8 EXTERNAL VIDEO

The external video interface allows mixing an external video source and VDP generated video under software control. As shown in Figure 2-20, composite video signals that are input to the 9918A through the external video input pin appear on the television screen in the plane behind the backdrop when the transparent color is programmed. Thus VDP-generated images on the Pattern Plane and the Sprite Planes can be superimposed upon an incoming signal. The source of the signal may be a standard NTSC broadcast signal, the output from an NTSC-compatible video-tape recorder, another VDP chip output, or any other NTSC-compatible composite video signal.

2.8.1 Hardware Design for External Video

The interface for external video operation consists of an external composite video signal on the EXTVID pin, a composite sync signal derived from the external composite video signal on the RESET/SYNC pin, and a method of synchronizing the external color subcarrier with the VDP's colors. The interface is designed so that the external video source provides the sync, blanking, and color burst to the television or monitor. The VDP controls the visible portion of the frame by gating in the external video signal when the transparent color is programmed on all planes. This preserves the colors of the external video picture by causing any inaccuracy in the external color lock circuit to modify the VDP colors but not, for example, the flesh tones of an external image.

The external video signal must be biased correctly and be of the proper amplitude so that the luminance levels of the external and VDP colors are matched and the external video does not bleed through into the composite video output of the VDP. The internal circuit assures that a perfect match results if the external video is of the same amplitude as the VDP's composite video and its dc level is increased by a MOS threshold voltage (typically 0.7 volts). This adjustment may be varied to change the relative luminance levels of the two video signals and thus modify the picture appearance.

The composite sync signal must also be of the proper levels. The RESET/SYNC pin is a tri-level input, utilizing 0, 5, and 12 volts. The 0-volt level activates reset of the VDP as described in section 2.1.7. The 5-volt level is the inactive state. The 12-volt level is the sync level. The input composite sync should thus swing from 5 volts to 12 volts. The timing of the composite sync controls the VDP in the following manner. A positive going edge to the RESET/SYNC pin is recognized as a horizontal sync pulse and resets the internal horizontal counter into the horizontal sync state. A sync pulse, which lasts for greater than 7.2 microseconds, is recognized as a vertical sync pulse and resets the internal vertical counter to the vertical sync state. Thus the VDP will be interlaced or not interlaced depending upon the composite sync signal.

To lock the colors of the external video with the VDP colors, a phase-lock loop maintains the VDP's input clock frequency at three times the external video subcarrier frequency. CPUCLK of the VDP is used as the feedback to maintain the color lock. See Figure 2-20 for a block diagram of the external video interface.

The input sync and phase-locked loop are not required when the external video source is another VDP. By running the VDP's from the same crystal or clock input and resetting the VDP's together with a fast edge (rise/fall time less than 30 ns), the VDP's will be locked on an open loop basis. See Figure 2-21 for the VDP to VDP interface.

There is a limitation in the resolution of the VDP-generated images when the external video signal is not from another VDP. The VDP can resolve only ± 1 pixel (186 nanoseconds) and, for a true NTSC-timed composite video signal, will cause a small sawtooth edge to surround vertical edges of the VDP-generated image. This results because the relationship of NTSC color subcarrier ($2/3$ of the VDP pixel rate) to the NTSC horizontal scan rate results in $34\frac{1}{4}$ pixels/line. Since the VDP can't resolve a $\frac{1}{4}$ pixel, a sawtooth edge is created by the timing. This sawtooth edge may be eliminated with loss of color by reducing the VDP input frequency to an integer multiple of the external horizontal sync. The phase-locked loop should lock the sync to the VDP input frequency. Thus the color lock is replaced with a sync lock. Color is no longer available since it is not within the subcarrier frequency.

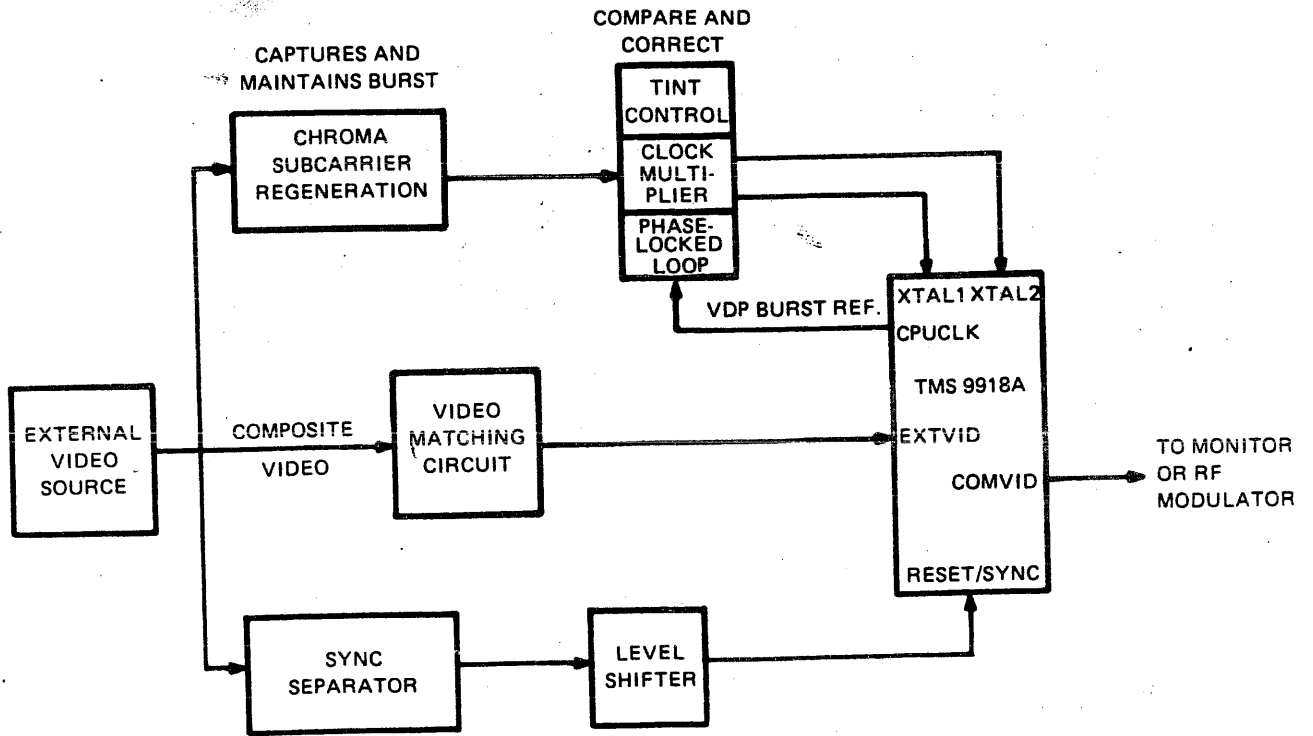


FIGURE 2-20 – EXTERNAL VIDEO INTERFACE BLOCK DIAGRAM

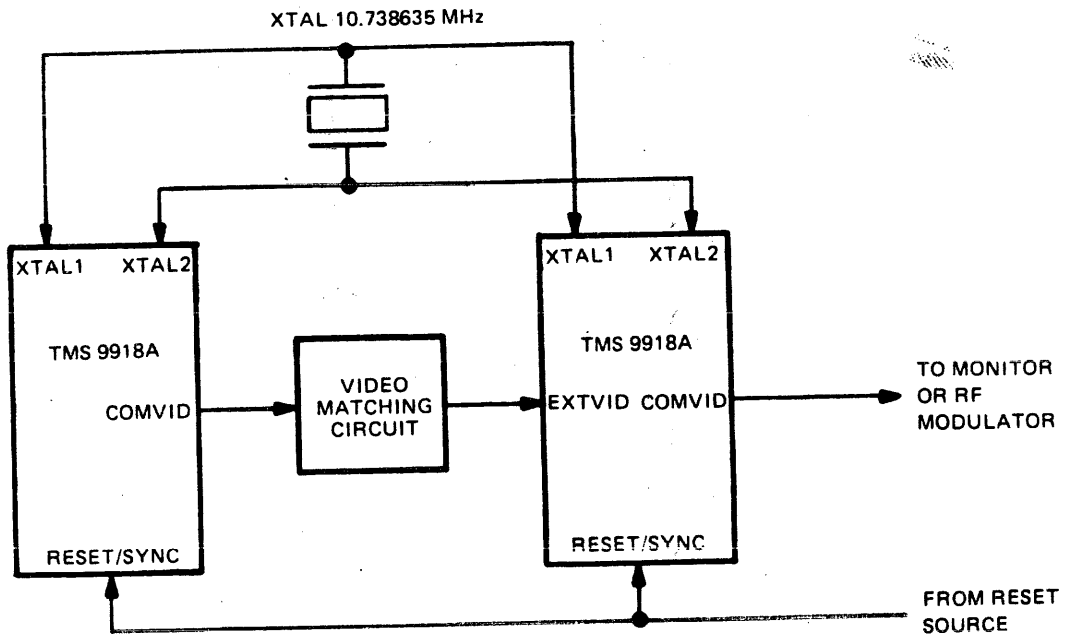


FIGURE 2-21 – CASCADING TWO TMS 9918A VDP'S

2.8.2 Software Requirements for External Video

For the External Video input plane to be visible, the External Video Enable bit in VDP Register 0 (EXVID) should be set to a '1'. The backdrop color (VDP Register 7, lower 4 bits) should be set to transparent (0). For the External Video plane to show through at a given spot on the screen, the Pattern color at that spot should be transparent, and all sprites should not be in the way (alternatively, a sprite that was in the way could be made transparent in color). Note that the External Video feature can be used in Graphics I, Graphics II, Text or Multicolor mode.

2.9 VRAM ADDRESS DERIVATION

Table summarizes the VRAM address derivation for all modes of operation for the TMS 9918A. Section 3 contains examples of how typical VRAM addresses are computed by the VDP.

TABLE 6 – PATTERN GRAPHICS ADDRESS LOCATION TABLE

GRAPHICS I MODE ADDRESS LOCATION TABLE

ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS	
1) PATTERN NAME ADDRESS		NTB			ROW				COLUMN						PATTERN NAME TABLE BASE (VDP REG2) PATTERN POSITION	
2) PATTERN COLOR ADDRESS			COLB						0		NAME (0-4)					PATTERN COLOR TABLE BASE (VDP REG3) ALWAYS "0" IN BIT 8 FIVE MOST SIGNIFICANT BITS OF NAME
3) PATTERN GENERATOR ADDRESS		PGB			NAME								XXX		PATTERN GENERATOR BASE (VDP REG4) ALL 8 BITS OF NAME THREE LSB'S FORM PATTERN ROW POSITION	

GRAPHICS II MODE ADDRESS LOCATION TABLE

ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS
1) PATTERN NAME ADDRESS		NTB			ROW				COLUMN						PATTERN NAME TABLE BASE (VDP REG2) PATTERN POSITION ROW PATTERN POSITION COLUMN
2) PATTERN COLOR ADDRESS			XX		NAME							XXX		PATTERN COLOR TABLE BASE MSB (VDP REG3) TWO MSB FROM VERTICAL COUNTER ALL 8 BITS OF NAME COLOR TABLE BYTE/LINE	
3) PATTERN GENERATOR ADDRESS			XX		NAME							XXX		PATTERN NAME TABLE BASE MSB (VDP REG4) TWO MSB FROM VERTICAL COUNTER ALL 8 BITS OF NAME PATTERN GENERATOR BYTE/LINE NUMBER	

TEXT MODE ADDRESS LOCATION TABLE

ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS	
TEXT MODE NAME ADDRESS		NTB			TEXT POSITION											PATTERN NAME TABLE BASE (VDP REG2) EQUAL (TEXT POSITION ROW * TIMES 40) PLUS (TEXT POSITION COLUMN NUMBER)
TEXT MODE PATTERN ADDRESS		PGB			NAME							XXX		PATTERN GENERATOR BASE (VDP REG4) NAME BYTE/LINE NUMBER		

SPRITE ADDRESS LOCATION TABLE

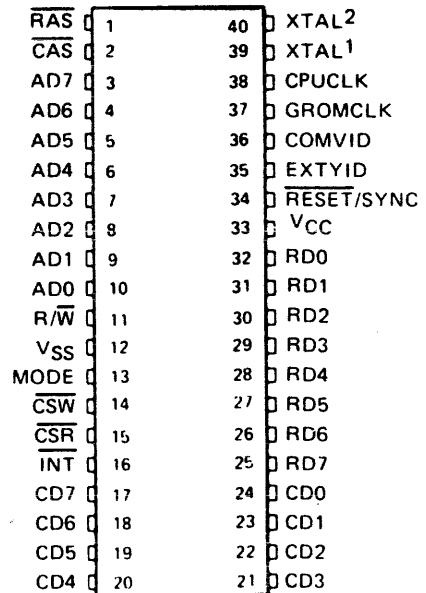
ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS
SPRITE ATTRIBUTE ADDRESS	SAB						SPRITE				XX			SPRITE ATTRIBUTE TABLE BASE (VDP REG5) SPRITE NUMBER ATTRIBUTE NUMBER: 00 FOR VERTICAL POSITION 01 FOR HORIZONTAL POSITION 10 FOR NAME 11 FOR TAG (EARLY CLOCK AND COLOR)	
	SPGB		NAME							XXX			SPRITE PATTERN GENERATOR BASE (VDP REG4) NAME ATTRIBUTE OF SPRITE THREE LSB'S GIVE BYTE/LINE NUMBER		
	SPGB		NAME (0-5)				XXXXX			SPRITE PATTERN GENERATOR BASE (VDP REG4) SIX MSB OF NAME SIZE = 1 SPRITE BYTE NUMBER (SEE FIGURE 19)					
SIZE = 0 SPRITE PATTERN GENERATOR	SPGB		NAME								XXX			SPRITE PATTERN GENERATOR BASE (VDP REG4) NAME ATTRIBUTE OF SPRITE THREE LSB'S GIVE BYTE/LINE NUMBER	
SIZE = 1 SPRITE PATTERN GENERATOR	SPGB		NAME (0-5)				XXXXX			SPRITE PATTERN GENERATOR BASE (VDP REG4) SIX MSB OF NAME SIZE = 1 SPRITE BYTE NUMBER (SEE FIGURE 19)					

MULTICOLOR ADDRESS LOCATION TABLE

ADDRESS TYPE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	COMMENTS
4) MULTICOLOR NAME ADDRESS	NTB			ROW					COLUMN				NAME TABLE BASE (VDP REG2) PATTERN POSITION ROW PATTERN POSITION COLUMN		
	PGB		NAME						XXX			PATTERN GENERATOR BASE (VDP REG4) NAME FROM NAME FETCH THREE LSB'S FORM BYTE/SQUARE ROW			
5) MULTICOLOR COLOR GENERATOR ADDRESS	PGB		NAME						XXX				PATTERN GENERATOR BASE (VDP REG4) NAME FROM NAME FETCH THREE LSB'S FORM BYTE/SQUARE ROW		
	PGB		NAME						XXX						

2.10 VDP TERMINAL ASSIGNMENTS

SIGNATURE	TERMINAL	I/O	DESCRIPTION
CD0 MSB	24	I/O	CPU data bus (CD0) is the most significant bit
CD1	23	I/O	
CD2	22	I/O	
CD3	21	I/O	
CD4	20	I/O	
CD5	19	I/O	
CD6	18	I/O	
CD7	17	I/O	
MODE	13	I	CPU interface mode select; usually a processor address line
$\overline{\text{CSR}}$	15	I	CPU-VDP read strobe
$\overline{\text{CSW}}$	14	I	CPU-VDP write strobe
VCC	33	I	+5 volt supply
VSS	12	I	Ground Reference
RD0 MSB	32	I	VRAM read data bus (RD0 is the most significant bit)
RD1	31	I	
RD2	30	I	
RD3	29	I	
RD4	28	I	
RD5	27	I	
RD6	26	I	
RD7	25	I	
AD0 MSB	10	O	VRAM address/data bus (multiplexed high and low order VRAM address and output data bytes)
AD1	9	O	AD0 is the most significant bit and is used only for data and not for addressing.*
AD2	8	O	
AD3	7	O	
AD4	6	O	
AD5	5	O	
AD6	4	O	
AD7	3	O	
$\overline{\text{RAS}}$	1	O	VRAM row address strobe
$\overline{\text{CAS}}$	2	O	VRAM column address strobe
$\overline{\text{R/W}}$	11	O	VRAM write strobe
XTAL1, XTAL2	40,39	I	10.7 + MHz crystal inputs.**
GROMCLK	37	O	VDP output clock = XTAL/24. Typically not used
$\overline{\text{RESET/SYNC}}$	34	I	$\overline{\text{RESET}}$ —This pin is a trilevel input pin. When it is below 0.8 volts, $\overline{\text{RESET}}$ initializes the VDP. When it is above 9 volts, $\overline{\text{RESET}}$ is the synchronizing input for external video.



SIGNATURE	TERMINAL	I/O	DESCRIPTION
EXTVID	35	I	External video input
CPUCLK	38	O	NTSC color burst frequency clock. Typically not used.
INT	16	O	CPU interrupt output.
COMVID	36	O	NTSC composite video output.

- * The least-significant address bit, AD7, is wired to A0 of the dynamic RAMs. Likewise, AD6 is wired to A1 of the RAMs. Care must be exercised in assuring proper orientation of the 9918A address outputs to the dynamic RAM address inputs.
- ** When driven externally, both inputs must be driven.

4. TMS 9918A PRELIMINARY ELECTRICAL SPECIFICATIONS

4.1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (unless otherwise noted)*

Supply voltage, V_{CC}	-0.3 to 20 V
All input voltages	-0.3 to 20 V
Output voltage	-2 to 7 V
Continuous power dissipation	1.8 W
Operating free-air temperature range	0°C to 55°C
Storage temperature range	-55°C to +150°C

* Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

4.2 RECOMMENDED OPERATING CONDITIONS*

PARAMETER		MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}		4.75		5.25	V
Supply voltage, V_{SS}			0		V
Input voltage, V_I , RESET/SYNC pin	SYNC active	10		12	V
	RESET active			0.6	V
	SYNC and RESET inactive	3		6	V
High-level input voltage, V_{IH}	XTAL1, XTAL2	2.75			V
	All other inputs	2.2			V
Input voltage, V_I , EXTVID pin	SYNC level				V
	White level				V
	Black level				V
Low-level input voltage, V_{IL}				0.8	V
Operating free-air temperature, T_A		0		55	°C

* All voltage values are with respect to V_{SS} .

4.3 ELECTRICAL CHARACTERISTICS OVER FULL RANGES OF RECOMMENDED OPERATING CONDITIONS (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP [†]	MAX	UNIT	
V _{OH}	High-level output voltage	$\overline{\text{RAS}}, \overline{\text{CAS}}, \overline{\text{R/W}}$	I _{OH} = -400 μ A			V	
		All other outputs	2.7		2.4		
V _{OL}	Low-level output voltage	CPU data	I _{OL} = 1.2 mA			V	
		DRAM interface	I _{OL} = 800 μ A				
I _{OZH}	Off-state output current high-level voltage applied, D0-D7 outputs	V _O = 5.5 V			100	μ A	
I _{OZL}	Off-state output current low-level voltage applied, D0-D7 outputs	V _O = 0 V			-100	μ A	
I _{IH}	High-level input current	V _I = 5.5 V, All other pins at 0 V			10	μ A	
I _{IL}	Low-level input current	V _I = 0 V, All other pins at 0 V			-10	μ A	
	Video voltage level of white, COMVID output			3.2		V	
	Video voltage level of black (blank), COMVID output			2.3		V	
	Video voltage level of sync, COMVID output			1.9		V	
	Video voltage (peak-to-peak) of burst, COMVID output			0.5		V	
	Video voltage difference, white - black, COMVID output			0.5		V	
I _{CC}	Average supply current from V _{CC}		T _A = 25°C		200	250	mA
C _i	Input capacitance	D0-D7	f = 1 MHz, Unmeasured pins at 0 V			20	pF
		All other inputs				10	
C _o	Output capacitance		f = 1 MHz, Unmeasured pins at 0 V			20	pF

[†]All typical values are at V_{CC} = 5.25 V, T_A = 25°C.

4.4 TIMING REQUIREMENTS OVER FULL RANGES OF RECOMMENDED OPERATING CONDITIONS

CPU-VDP interface

PARAMETER		MIN	NOM	MAX	UNIT
t _{su} (ARL)	Address setup time before CSR low		0		ns
t _{su} (AWL)	Address setup time before CSW low		30		ns
t _h (WLA)	Address hold time after CSW low		30		ns
t _{su} (DWH)	Data setup time before CSW high		100		ns
t _h (WHD)	Data hold time after CSW high		30		ns
t _w (WL)	Pulse width, CSW low		200		ns
t _w (CSH1)	Pulse width, chip select high (requesting memory access)		8		μ s
t _w (CSH2)	Pulse width, chip select high (not requesting memory access)		3		μ s

VDP-VRAM interface

PARAMETER		MIN	NOM	MAX	UNIT
t _c	Memory read or write cycle time	372			ns
t _{su} (DCH)	Input data setup time before CAS high	160			ns
t _h (CHD)	Input data hold time after CAS high	0			ns

external clock source

PARAMETER		MIN	TYP	MAX	UNIT
f _{ext}	External source frequency		10.738		MHz
t _r /T _f	External source rise/fall time		10		ns
t _{WH}	External source high level pulse width	42	47	52	ns
t _{WL}	External source low level pulse width	42	47	52	ns
t _{PD}	External source phase delay from XTAL1 falling edge to XTAL2 falling edge	42	47	52	ns

4.5 SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

CPU-VDP interface

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_a(\text{CSR})$	Data access time from $\overline{\text{CSR}}$ low	$C_L = 300 \text{ pF}$		300		ns
t_{PVX}	Data disable time after $\overline{\text{CSR}}$ high			200		ns
$t_{\text{PVX,A}}$	Data invalid time from address changes			0		ns
f_{CPUCLK}	CPU clock output frequency ($f_{\text{ext}} \div 3$)			3.58		MHz
f_{GROMCLK}	GROM clock output frequency ($\text{ext} \div 24$)			447.5		kHz

VDP-VRAM interface

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_w(\text{CH})$	Pulse width, $\overline{\text{CAS}}$ high	$C_L = 50 \text{ pF}$	60			ns
$t_w(\text{CL})$	Pulse width, $\overline{\text{CAS}}$ low		100		10,000	ns
$t_w(\text{RH})$	Pulse width, $\overline{\text{RAS}}$ high		100			ns
$t_w(\text{RL})$	Pulse width, $\overline{\text{RAS}}$ low		150		10,000	ns
$t_w(\text{W})$	Pulse width, write pulse		45			ns
$t_{\text{CA-CL}}$	Delay time, column address to $\overline{\text{CAS}}$ low		-10			ns
$t_{\text{RA-RL}}$	Delay time, row address to $\overline{\text{RAS}}$ low		0			ns
$t_{\text{d-WL}}$	Delay time, data to W low		0			ns
$t_{\text{WH-CL}}$	Delay time, R/W high to $\overline{\text{CAS}}$ low		0			ns
$t_{\text{W-CH}}$	Delay time, R/W low to $\overline{\text{CAS}}$ high		60			ns
$t_{\text{W-RH}}$	Delay time, R/W low to $\overline{\text{RAS}}$ high		60			ns
$t_{\text{CL-CA}}$	Column address valid after $\overline{\text{CAS}}$ low		45			ns
$t_{\text{RL-RA}}$	Row address valid after $\overline{\text{RAS}}$ low		20			ns
$t_{\text{RL-CA}}$	Column address valid after $\overline{\text{RAS}}$ low		95			ns
$t_{\text{CL-D}}$	Data valid after $\overline{\text{CAS}}$ low		45			ns
$t_{\text{RL-D}}$	Data valid after $\overline{\text{RAS}}$ low		95			ns
$t_{\text{WL-D}}$	Data valid after R/W low		45			ns
$t_{\text{CH-WL}}$	Read command valid after $\overline{\text{CAS}}$ high		0			ns
$t_{\text{CL-W}}$	Write command valid after $\overline{\text{CAS}}$ low		45			ns
$t_{\text{CH-RL}}$	Delay time, $\overline{\text{CAS}}$ high to $\overline{\text{RAS}}$ low		-20			ns
$t_{\text{CL-RH}}$	Delay time, $\overline{\text{CAS}}$ low to $\overline{\text{RAS}}$ high		100			ns
$t_{\text{RL-CL}}$	Delay time, $\overline{\text{RAS}}$ low to $\overline{\text{CAS}}$ low		20		50	ns

composite video output

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{xsad}	SYNC to active display		11.2		μs
t_{xhsw}	Horizontal SYNC width	1		5	μs
t_{xvsw}	Vertical SYNC width	7.8		50	μs

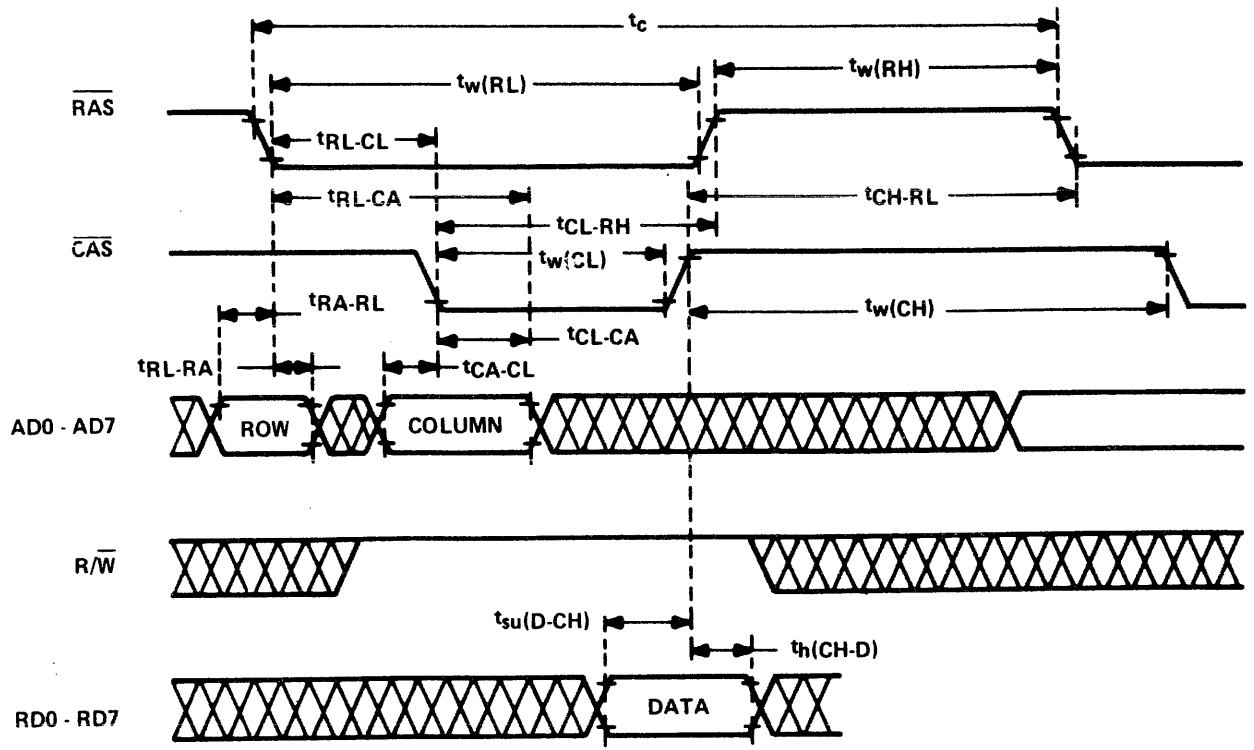


FIGURE 4-1 - VRAM READ CYCLE

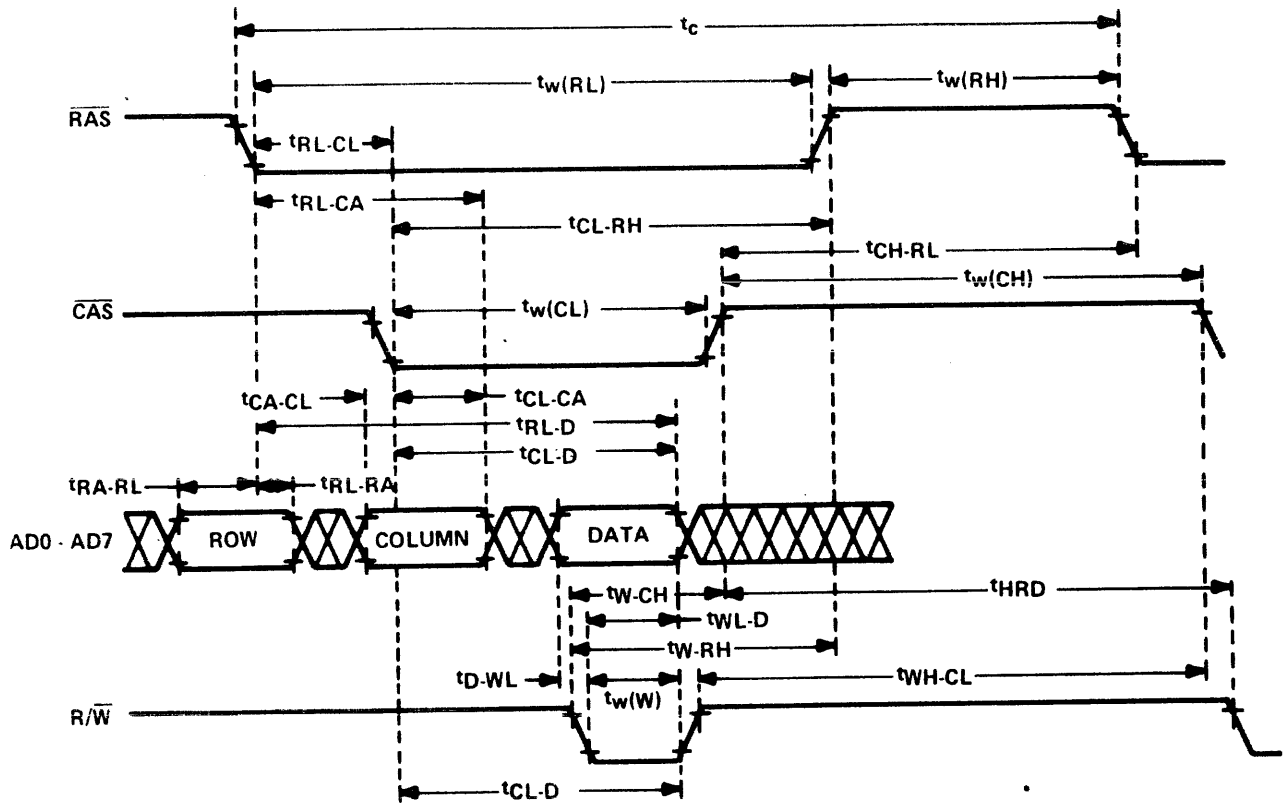
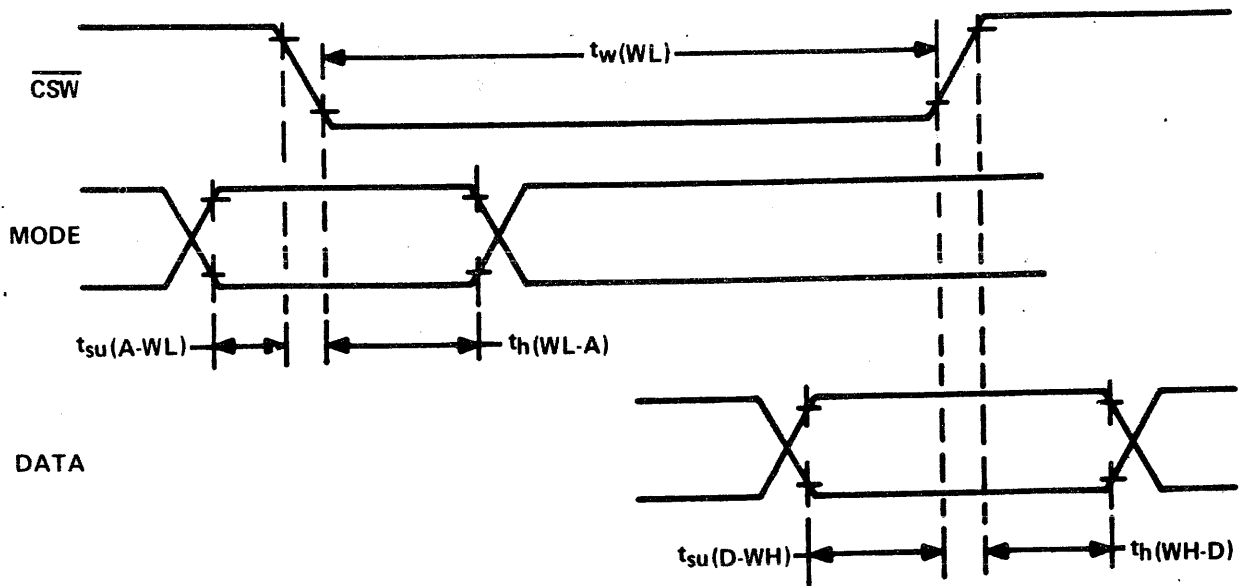
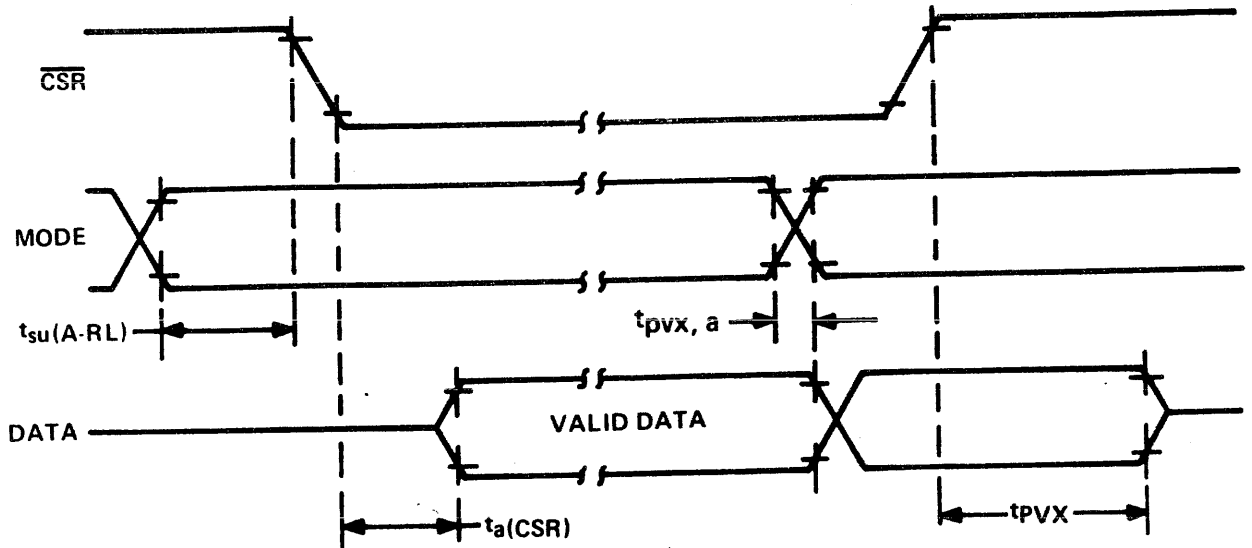


FIGURE 4-2 - VRAM WRITE CYCLE

WRITE CYCLE



READ CYCLE



NOTE: All measurements are made at 10% and 90% points.

FIGURE 4-3 – CPU-VDP WRITE CYCLE

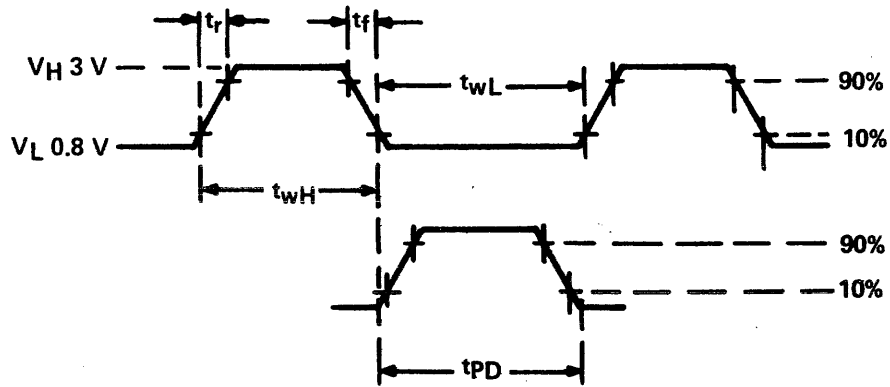
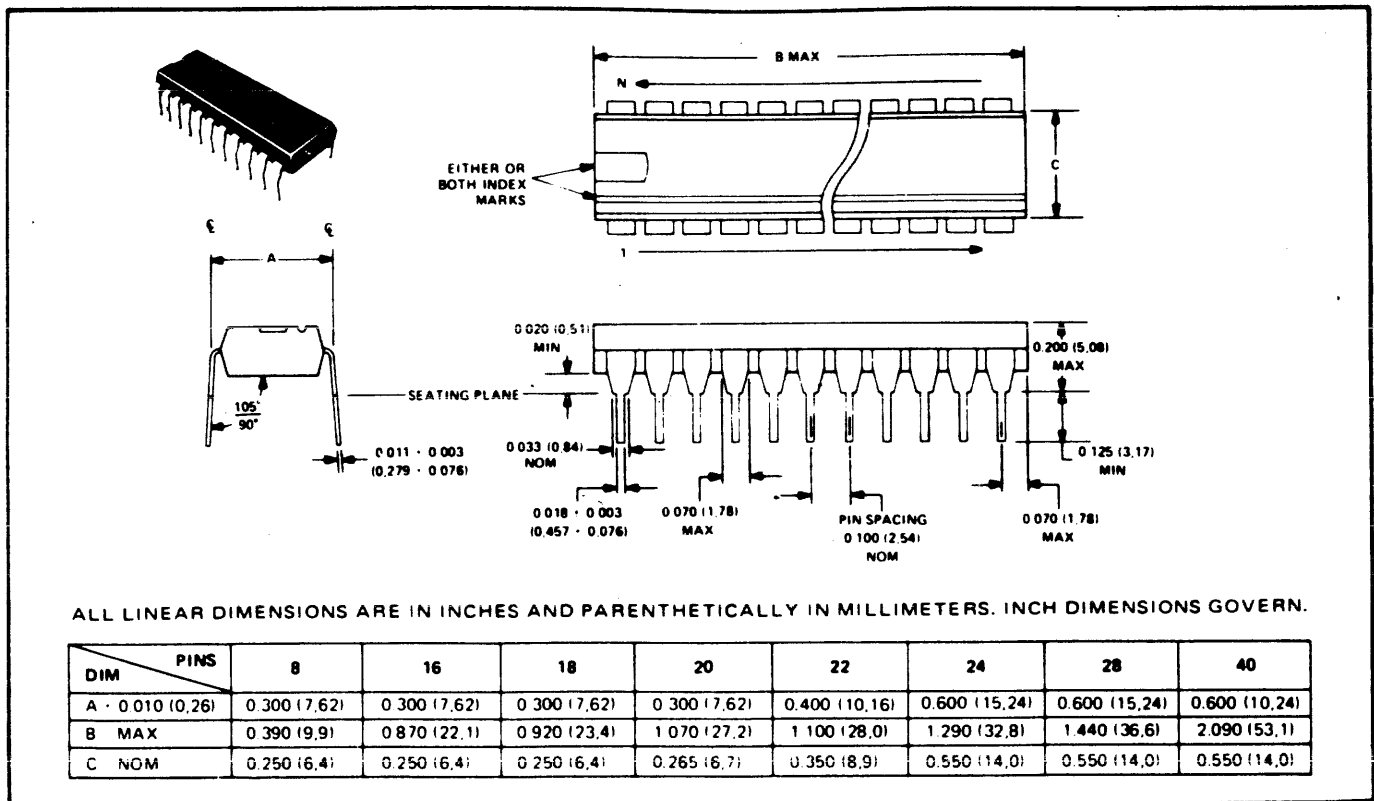


FIGURE 4-4 – EXTERNAL CLOCK TIMING WAVEFORM

5. MECHANICAL DATA

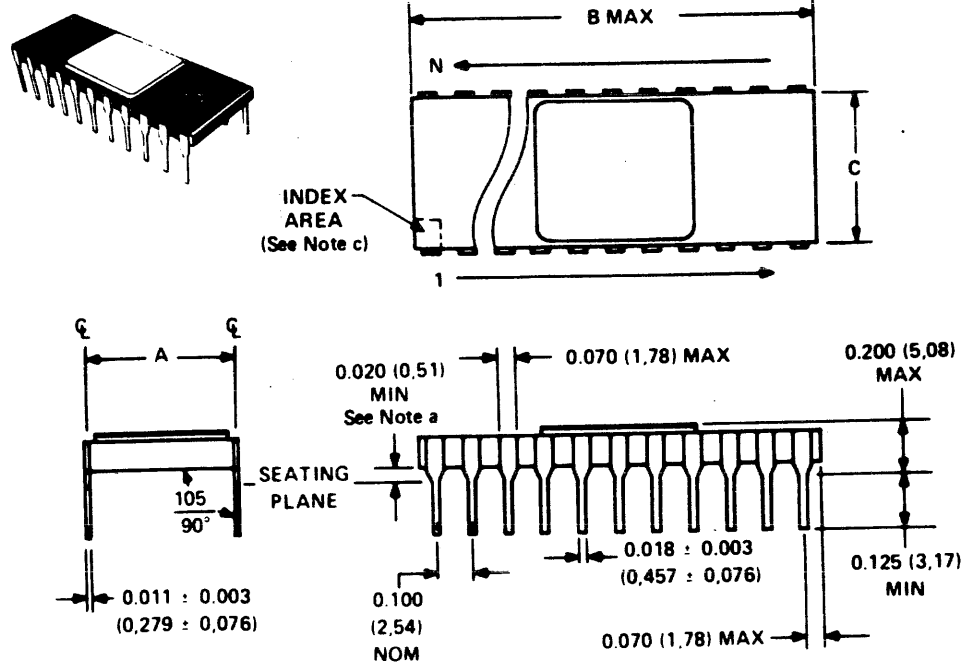
5.1 TMS 9918 — 40-PIN PLASTIC PACKAGE

plastic packages



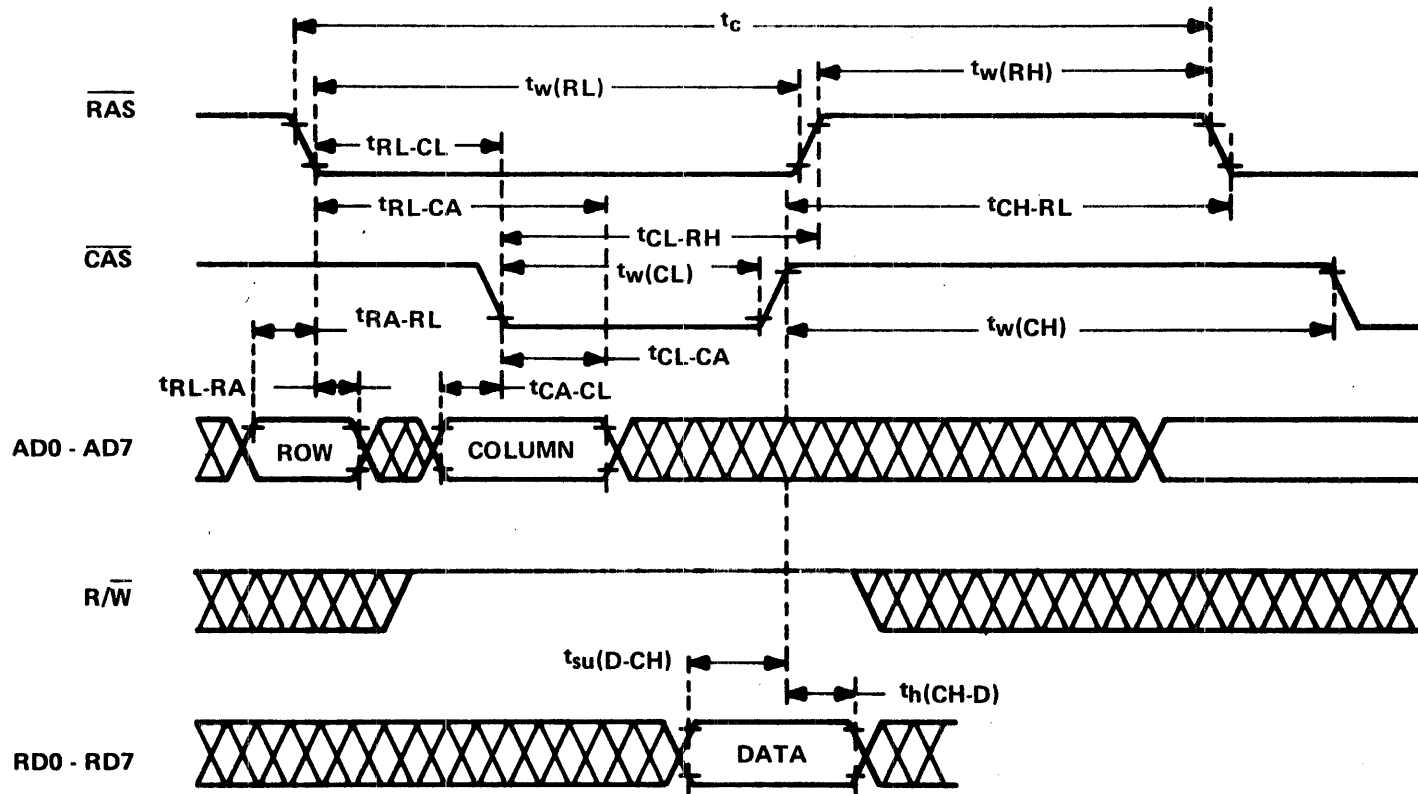
5.2 TMS 9918 — 40-PIN CERAMIC PACKAGE

ceramic packages with side-brazed leads and metal or epoxy or glass lid seal



- NOTES: a. This minimum spacing is valid for printed circuit board mounting with 0.033 (0,84) diameter holes for the leads.
 b. All linear dimensions are in inches and parenthetically in millimeters. Inch dimensions govern.
 c. The index is placed in this area to identify pin 1 and to provide other information as follows:
- 1 Pin 1 connected to chip-mounting pad.
 - ΔXX Pin XX connected to chip-mounting pad.
 - No connection to chip-mounting pad.
- Other symbols may indicate any combination of up to 4 pins connected to the chip-mounting pad.

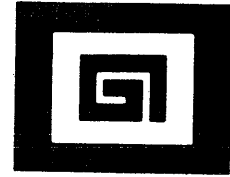
DIM	PINS	16	18	20	22	24	28	40
A ±	0.010 (0,26)	0.300 (7,62)	0.300 (7,62)	0.300 (7,62)	0.400 (10,16)	0.600 (15,24)	0.600 (15,24)	0.600 (15,24)
B MAX		0.840 (21,4)	0.910 (23,1)	1.020 (25,9)	1.100 (28,0)	1.290 (32,8)	1.415 (36,0)	2.020 (51,3)
C NOM		0.290 (7,4)	0.290 (7,4)	0.290 (7,4)	0.390 (9,9)	0.590 (15,0)	0.590 (15,0)	0.590 (15,0)



Appendix C

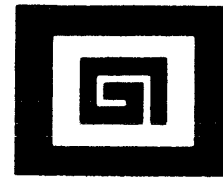
AY-3-8910 Data Sheet
(PSG)

The following material is copyrighted by General Instrument Corporation. It is reprinted here with the permission of General Instruments. This data sheet may not be reproduced for any purpose in whole or part without the expressed written consent of General Instruments.



**AY-3-8910/8912
PROGRAMMABLE
SOUND GENERATOR
DATA MANUAL**

**GENERAL INSTRUMENT CORPORATION
MICROELECTRONICS**



AY-3-8910/8912 PROGRAMMABLE SOUND GENERATOR DATA MANUAL

ARCHITECTURE

OPERATION

INTERFACING

MUSIC GENERATION

SOUND EFFECTS GENERATION

ELECTRICAL SPECIFICATIONS

TABLE OF CONTENTS—PAGE 2

FEBRUARY 1979

©Copyright 1979 GENERAL INSTRUMENT CORPORATION

All information in this book is subject to change without notice.

General Instrument Corporation cannot assume responsibility for the use of any circuits described herein.

Table of Contents

1. INTRODUCTION	
1.1 Description	5
1.2 Features	6
1.3 Scope	6
2. ARCHITECTURE	
2.1 Basic Functional Blocks	7
2.1.1 Register Array	7
2.1.2 Sound Generating Blocks	10
2.1.3 I/O Ports	10
2.2 Pin Assignments	12
2.3 Pin Functions	13
2.4 Bus Timing	15
2.5 State Timing	16
2.5.1 Address PSG Register Sequence	16
2.5.2 Write Data to PSG Sequence	17
2.5.3 Read Data from PSG Sequence	17
2.5.4 Write to/Read from I/O Port Sequence	17
3. OPERATION	
3.1 Tone Generator Control	18
3.2 Noise Generator Control	20
3.3 Mixer Control—I/O Enable	21
3.4 Amplitude Control	22
3.5 Envelope Generator Control	24
3.5.1 Envelope Period Control	24
3.5.2 Envelope Shape/Cycle Control	25
3.6 I/O Port Data Store	28
3.7 D/A Converter Operation	29
4. INTERFACING	
4.1 Introduction	32
4.2 Clock Generation	33
4.3 Audio Output Interface	34
4.4 External Memory Access	35
4.5 Microprocessor/Microcomputer Interface	36
4.6 Interfacing to the PIC 1650	37
4.6.1 Write Data Routine	37
4.6.2 Read Data Routine	38
4.6.3 Read ROM Routine	38
4.7 Interfacing to the CP1600/1610	40
4.7.1 Write Data Routine	40
4.7.2 Read Data Routine	40
4.8 Interfacing to the M6800	42
4.8.1 Latch Address Routine	42
4.8.2 Write Data Routine	42
4.8.3 Read Data Routine	42
4.9 Interfacing to the 8080 S100 Bus	44
4.9.1 Latch Address Routine	44
4.9.2 Write Data Routine	44
4.9.3 Read Data Routine	44

5. MUSIC GENERATION	
5.1 Note Generation	46
5.2 Tune Entry/Playback	48
5.3 Tune Variation	48
5.3.1 Octave Shift	48
5.3.2 Key	48
5.3.3 Tempo	48
5.3.4 Chords	49
5.4 Sound Variation	50
5.4.1 Relative Channel Volume	50
5.4.2 Decay	50
5.4.3 Other Effects	50
5.5 Applications	51
5.5.1 Organ Envelope Generation	51
5.5.2 Organ Rhythm Generation	52
6. SOUND EFFECTS GENERATION	
6.1 Tone Only Effects	53
6.2 Noise Only Effects	54
6.3 Frequency Sweep Effects	55
6.4 Multi-Channel Effects	56
7. ELECTRICAL SPECIFICATIONS	
7.1 Maximum Ratings	57
7.2 Standard Conditions	57
7.3 DC Characteristics	57
7.4 AC Characteristics	58
7.5 Package Outlines	60

List of Illustrations

Fig. 1 Typical System Diagram	6
Fig. 2 PSG Block Diagram	8
Fig. 3 PSG Register Array	11
Fig. 4 AY-3-8910 Pin Assignments	12
Fig. 5 AY-3-8912 Pin Assignments	12
Fig. 6 Variable Amplitude Control	23
Fig. 7 Envelope Shape/Cycle Control	26
Fig. 8 Detail of Two Cycles of Fig. 7	27
Fig. 9 D/A Converter Output	29
Fig. 10 Single Tone with Envelope Shape/Cycle Pattern 1000	30
Fig. 11 Single Tone with Envelope Shape/Cycle Pattern 1100	30
Fig. 12 Single Tone with Envelope Shape/Cycle Pattern 1010	31
Fig. 13 Mixture of Three Tones with Fixed Amplitudes	31
Fig. 14 System Block Diagram	32
Fig. 15 Clock Generation	33
Fig. 16 Audio Output Interface	34
Fig. 17 External Memory Access	35
Fig. 18 Microprocessor/Microcomputer Interface	36
Fig. 19 PIC 1650/AY-3-8910 System Example	39
Fig. 20 CP1600/1610/AY-3-8910 Interface	41
Fig. 21 M6800/AY-3-8910 Interface	43
Fig. 22 8080 S100 Bus/AY-3-8910 Interface	45
Fig. 23 Equal Tempered Chromatic Scale	47
Fig. 24 Chord Selection Chart	49
Fig. 25 Organ Envelope Generation	51
Fig. 26 Organ Rhythm Generation	52
Fig. 27 European Siren Sound Effect Chart	53
Fig. 28 Gunshot Sound Effect Chart	54
Fig. 29 Explosion Sound Effect Chart	54

List of Illustrations (cont.)

Fig. 30 Laser Sound Effect Chart	55
Fig. 31 Whistling Bomb Sound Effect Chart	55
Fig. 32 Wolf Whistle Sound Effect Chart	56
Fig. 33 Race Car Sound Effect Chart	56
Fig. 34 Analog Channel Output Test Circuit	57
Fig. 35 Current to Voltage Converter	57
Fig. 36 Clock and Bus Signal Timing	58
Fig. 37 Reset Timing	58
Fig. 38 Latch Address Timing	59
Fig. 39 Write Data Timing	59
Fig. 40 Read Data Timing	59
Fig. 41 40 Lead Dual In Line Packages	60
Fig. 42 28 Lead Dual In Line Packages	61

1 INTRODUCTION

It is apparent that any microprocessor is capable of producing acceptable sounds with only a transducer if the processor has no other tasks to perform while the sound is sustained. In real world microprocessor use, however, video games need refreshing, keyboards need scanning, etc. For example, in order to produce a single channel of ninth octave C (8372 Hz) the signal needs attention every sixty microseconds. Software required to produce this simple effect and still perform other activities would in the least be very complex if not impossible. In the extreme, random noise requires periodic attention even more frequently.

This need for software-produced sounds without the constant attention of the processor is now satisfied with the availability of the General Instrument AY-3-8910 and AY-3-8912 Programmable Sound Generators.

1.1 Description

The AY-3-8910/8912 Programmable Sound Generator (PSG) is a Large Scale Integrated Circuit which can produce a wide variety of complex sounds under software control. The AY-3-8910/8912 is manufactured in GI's N-Channel Ion Implant Process. Operation requires a single 5V power supply, a TTL compatible clock, and a microprocessor controller such as the GI 16-bit CP1600/1610 or one of GI's PIC 1650 series of 8-bit microcomputers.

The PSG is easily interfaced to any bus oriented system. Its flexibility makes it useful in applications such as music synthesis, sound effects generation, audible alarms, tone signalling and FSK modems. The analog sound outputs can each provide 4 bits of logarithmic digital to analog conversion, greatly enhancing the dynamic range of the sounds produced.

In order to perform sound effects while allowing the processor to continue its other tasks, the PSG can continue to produce sound after the initial commands have been given by the control processor. The fact that realistic sound production often involves more than one effect is satisfied by the three independently controllable channels available in the PSG.

All of the circuit control signals are digital in nature and intended to be provided directly by a microprocessor/microcomputer. This means that one PSG can produce the full range of required sounds with no change in external circuitry. Since the frequency response of the PSG ranges from sub-audible at its lowest frequency to post-audible at its highest frequency, there are few sounds which are beyond reproduction with only the simplest electrical connections.

Since most applications of a microprocessor/PSG system would also require interfacing between the outside world and the microprocessor, this facility has been designed into the PSG. The AY-3-8910 has two general purpose 8-bit I/O ports and is supplied in a 40 lead package; the AY-3-8912 has one port and 28 leads.

1.2 Features

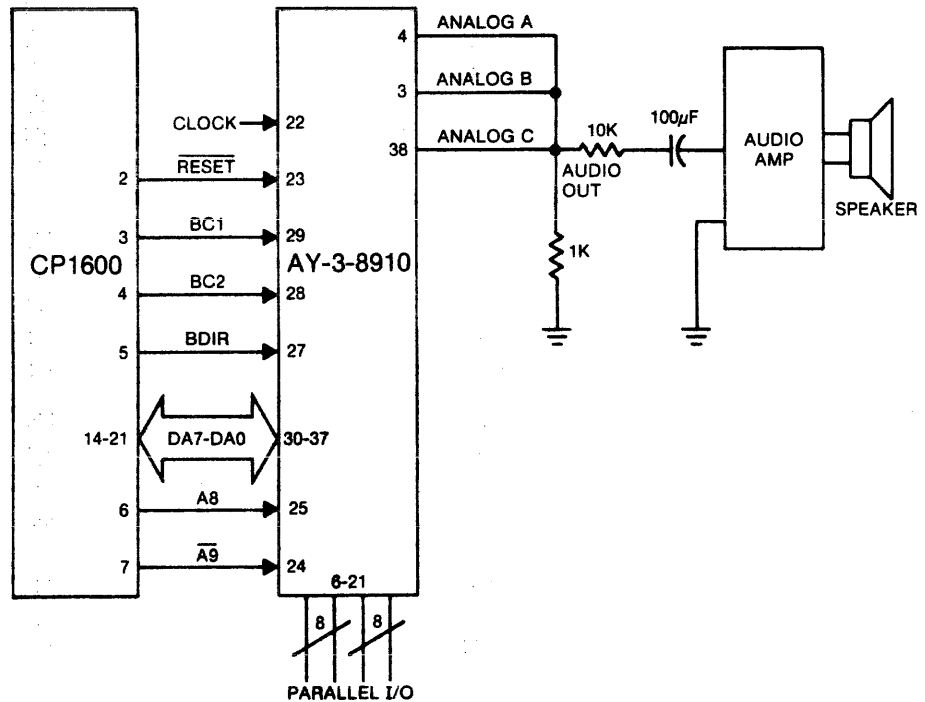
- Full software control of sound generation.
- Interfaces to most 8-bit and 16-bit microprocessors.
- Three independently programmed analog outputs.
- Two 8-bit general purpose I/O ports (AY-3-8910).
- One 8-bit general purpose I/O port (AY-3-8912).
- Single +5 Volt Supply.

1.3 Scope

This Data Manual is intended to introduce the techniques needed to cause the AY-3-8910/8912 Programmable Sound Generator to perform in its intended fashion. All of the programs, programming, and hardware designs have been tested to ensure that the methods are practical rather than purely theoretical.

Although the techniques described will produce powerful results, the range of sounds to be synthesized is so vast and the PSG capabilities so varied that this guide should be viewed merely as an introduction to the applications possibilities of the PSG.

Fig. 1 TYPICAL SYSTEM DIAGRAM



2 ARCHITECTURE

The AY-3-8910/8912 is a register oriented Programmable Sound Generator (PSG). Communication between the processor and the PSG is based on the concept of memory-mapped I/O. Control commands are issued to the PSG by writing to 16 memory-mapped registers. Each of the 16 registers within the PSG is also readable so that the microprocessor can determine, as necessary, present states or stored data values.

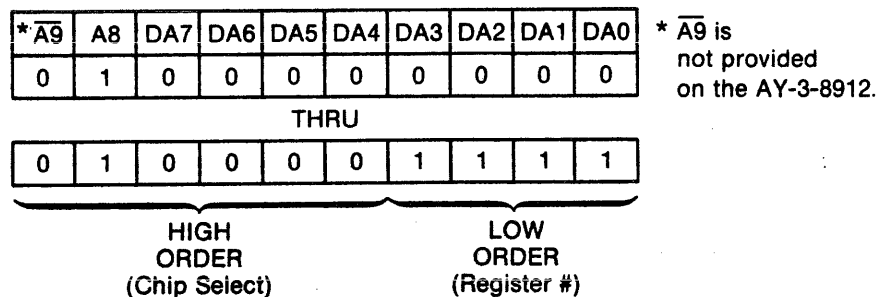
All functions of the PSG are controlled through its 16 registers which once programmed, generate and sustain the sounds, thus freeing the system processor for other tasks.

2.1 Basic Functional Blocks

An internal block diagram of the PSG showing the various functional blocks and data flow is shown in Fig. 2.

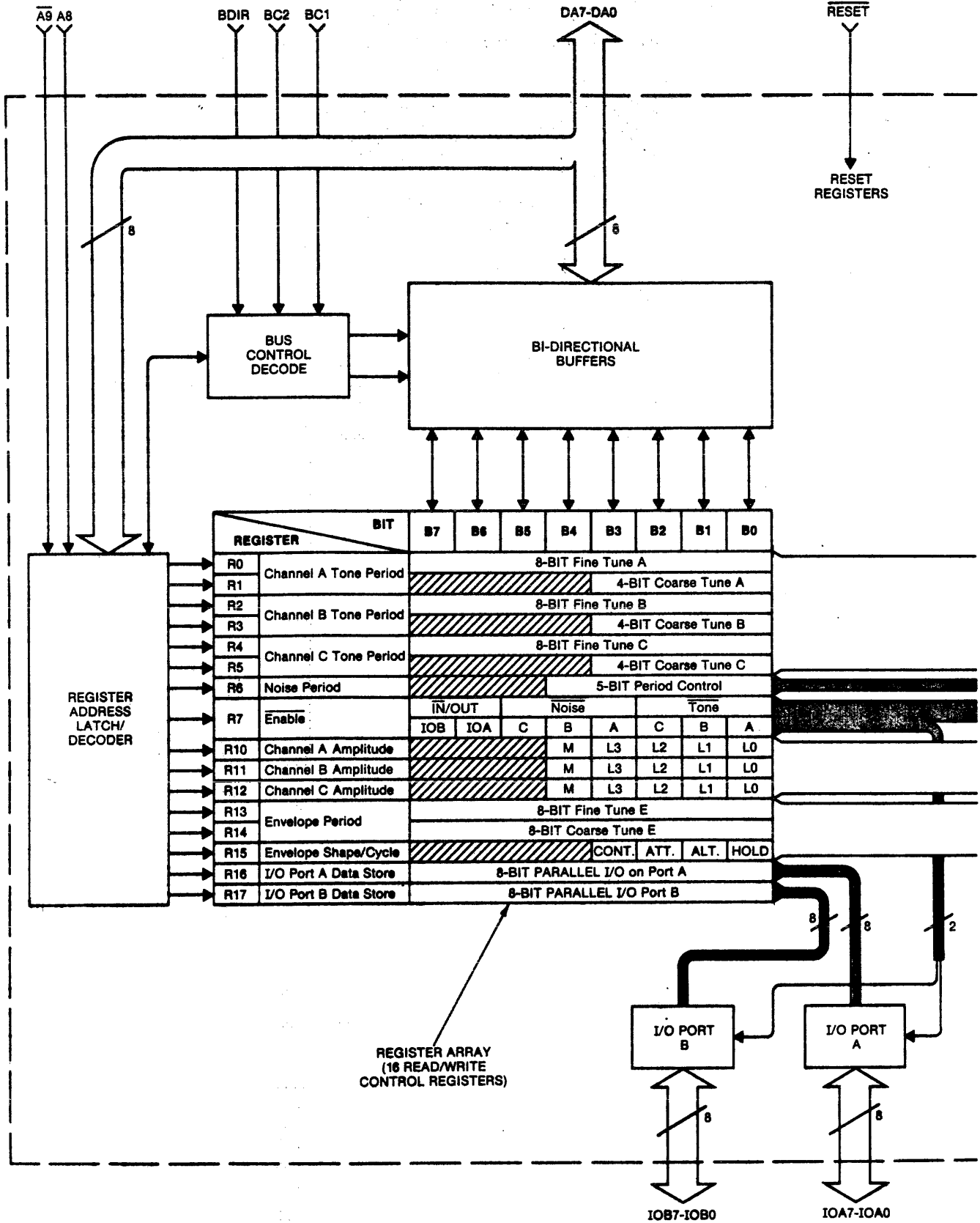
2.1.1 REGISTER ARRAY

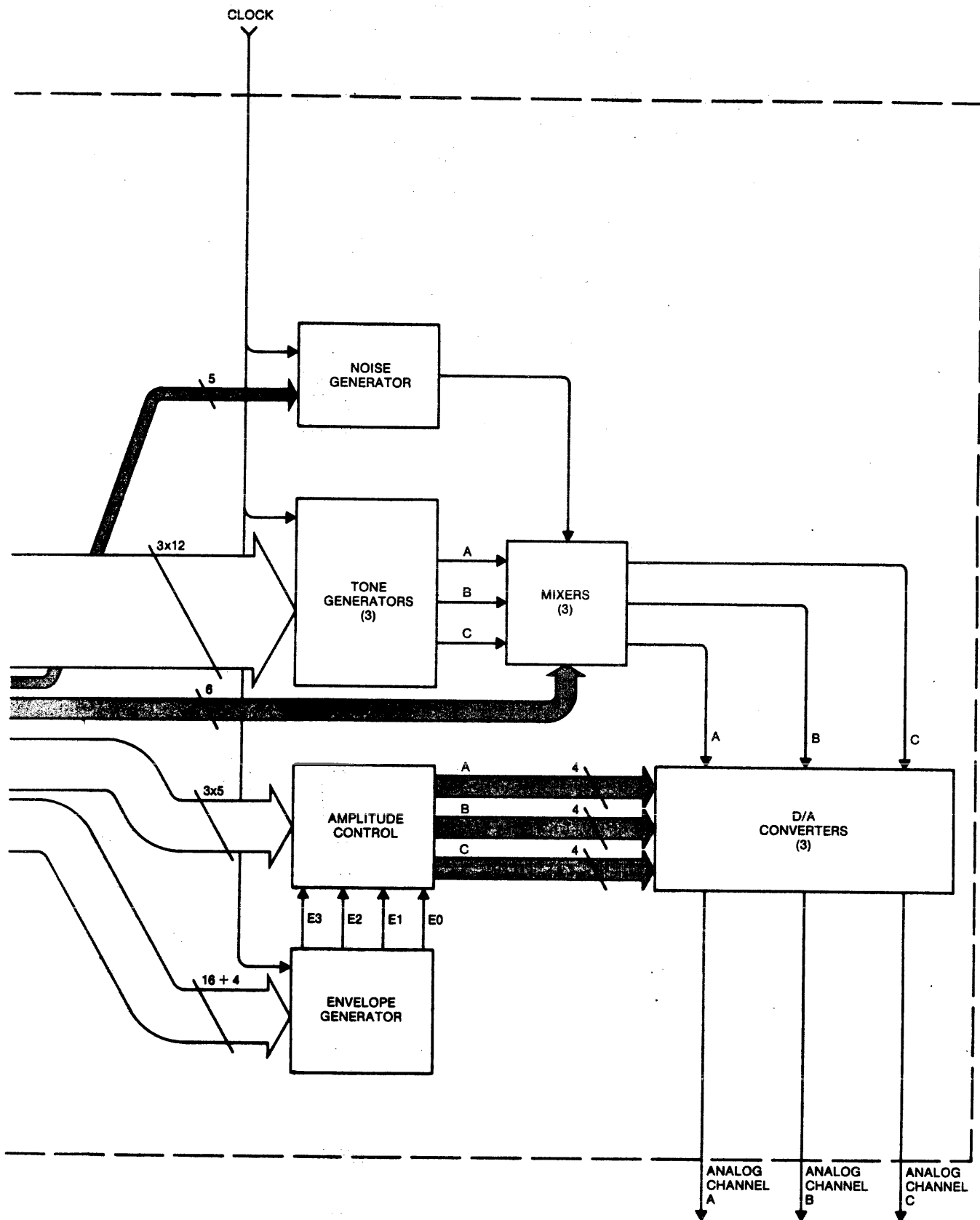
The principal element of the PSG is the array of 16 read/write control registers. These 16 registers look to the CPU as a block of memory and as such occupy a 16 word block out of 1,024 possible addresses. The 10 address bits (8 bits on the common data/address bus, and 2 separate address bits A8 and $\overline{A9}$) are decoded as follows:



The four low order address bits select one of the 16 registers (R0--R15). The six high order address bits function as "chip selects" to control the tri-state bidirectional buffers (when the high order address bits are "incorrect", the bidirectional buffers are forced to a high impedance state). High order address bits $\overline{A9}$ A8 are fixed in the PSG design to recognize a 01 code; high order address bits DA7--DA4 may be mask-programmed to any 4-bit code by a special order factory mask modification. Unless otherwise specified, address bits DA7--DA4 are programmed to recognize only a 0000 code. A valid high order address latches the register address (the low order 4 bits) in the Register Address Latch/Decoder block. A latched address will remain valid until the receipt of a new address, enabling multiple reads and writes of the same register contents without the need for redundant re-addressing.

Fig. 2 PSG BLOCK DIAGRAM





2.1 Basic Functional Blocks (cont.)

Conditioning of the Register Address Latch/Decoder and the Bidirectional Buffers to recognize the bus function required (inactive, latch address, write data, or read data) is accomplished by the Bus Control Decode block.

The function of each of the 16 PSG registers and the data flow of each register's contents are shown in context in Fig. 2 and explained in detail in Section 3, "Operation". For reference purposes, the Register Array details are reproduced in Fig. 3.

2.1.2 SOUND GENERATING BLOCKS

The basic blocks in the PSG which produce the programmed sounds include:

Tone Generators	produce the basic square wave tone frequencies for each channel (A,B,C)
Noise Generator	produces a frequency modulated pseudo random pulse width square wave output.
Mixers	combine the outputs of the Tone Generators and the Noise Generator. One for each channel (A,B,C).
Amplitude Control	provides the D/A Converters with either a fixed or variable amplitude pattern. The fixed amplitude is under direct CPU control; the variable amplitude is accomplished by using the output of the Envelope Generator.
Envelope Generator	produces an envelope pattern which can be used to amplitude modulate the output of each Mixer.
D/A Converters	the three D/A Converters each produce up to a 16 level output signal as determined by the Amplitude Control.

2.1.3 I/O PORTS

Two additional blocks are shown in the PSG Block Diagram which have nothing directly to do with the production of sound—these are the two I/O Ports (A and B). Since virtually all uses of microprocessor-based sound would require interfacing between the outside world and the processor, this facility has been included in the PSG. Data to/from the CPU bus may be read/written to either of two 8-bit I/O Ports without affecting any other function of the PSG. The I/O Ports are TTL-compatible and are provided with internal pull-ups on each pin. Both Ports are available on the AY-3-8910; only I/O Port A is available on the AY-3-8912.

Fig. 3 PSG REGISTER ARRAY

REGISTER		BIT							
		B7	B6	B5	B4	B3	B2	B1	B0
R0	Channel A Tone Period	8-BIT Fine Tune A							
R1		/				4-BIT Coarse Tune A			
R2	Channel B Tone Period	8-BIT Fine Tune B							
R3		/				4-BIT Coarse Tune B			
R4	Channel C Tone Period	8-BIT Fine Tune C							
R5		/				4-BIT Coarse Tune C			
R6	Noise Period	/				5-BIT Period Control			
R7	Enable	IN/OUT		Noise			Tone		
		IOB	IOA	C	B	A	C	B	A
R10	Channel A Amplitude	/			M	L3	L2	L1	L0
R11	Channel B Amplitude	/			M	L3	L2	L1	L0
R12	Channel C Amplitude	/			M	L3	L2	L1	L0
R13	Envelope Period	8-BIT Fine Tune E							
R14		8-BIT Coarse Tune E							
R15	Envelope Shape/Cycle	/				CONT.	ATT.	ALT.	HOLD
R16	I/O Port A Data Store	8-BIT PARALLEL I/O on Port A							
R17	I/O Port B Data Store	8-BIT PARALLEL I/O Port B							

8
9
A 10
B 11
C 12
D 13
E 14
F 15

2.2 Pin Assignments

The AY-3-8910 is supplied in a 40 lead dual in-line package with the pin assignments as shown in Fig. 4. The AY-3-8912 is supplied in a 28 lead dual in-line package with the pin assignments as shown in Fig. 5.

Fig. 4 AY-3-8910 PIN ASSIGNMENTS

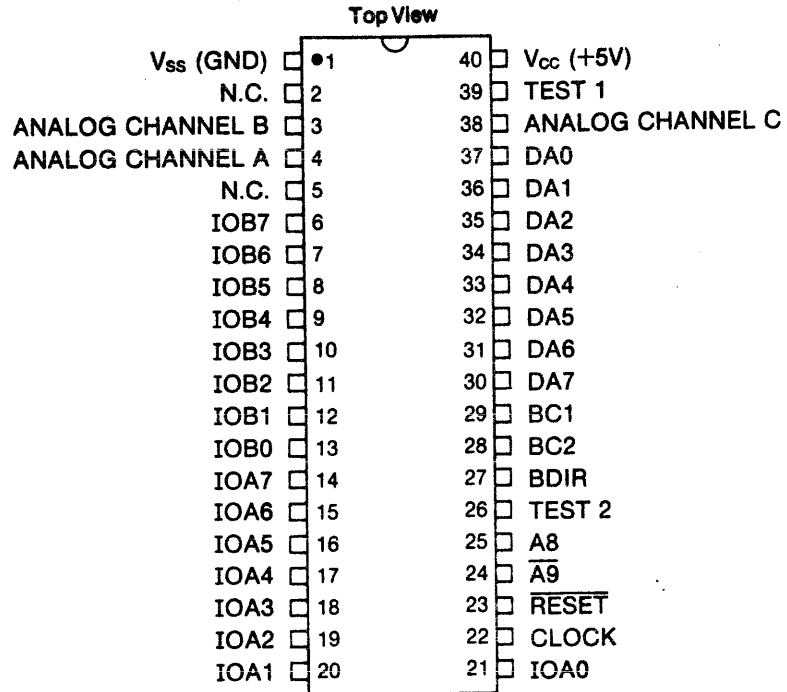
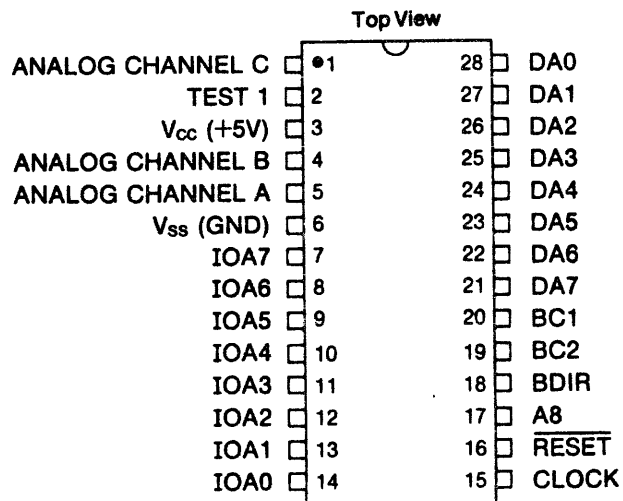


Fig. 5 AY-3-8912 PIN ASSIGNMENTS



2.3 Pin Functions

DA7--DA0 (input/output/high impedance): pins 30--37 (AY-3-8910)
Data/Address 7--0: pins 21--28 (AY-3-8912)

These 8 lines comprise the 8-bit bidirectional bus used by the microprocessor to send both data and addresses to the PSG and to receive data from the PSG. In the data mode, DA7--DA0 correspond to Register Array bits B7--B0. In the address mode, DA3--DA0 select the register # (0--17_h) and DA7--DA4 in conjunction with address inputs $\overline{A9}$ and A8 form the high order address (chip select).

A8 (input): pin 25 (AY-3-8910)

pin 17 (AY-3-8912)

$\overline{A9}$ (input): pin 24 (AY-3-8910)

(not provided on AY-3-8912)

Address 9, Address 8

These "extra" address bits are made available to enable the positioning of the PSG (assigning a 16 word memory space) in a total 1,024 word memory area rather than in a 256 word memory area as defined by address bits DA7--DA0 alone. If the memory size does not require the use of these extra address lines they may be left unconnected as each is provided with either an on-chip pull down ($\overline{A9}$) or pull-up (A8) resistor. In "noisy" environments, however, it is recommended that $\overline{A9}$ and A8 be tied to an external ground and +5V, respectively, if they are not to be used.

RESET (input): pin 23 (AY-3-8910)

pin 16 (AY-3-8912)

For initialization/power-on purposes, applying a logic "0" (ground) to the Reset pin will reset all registers to "0". The Reset pin is provided with an on-chip pull-up resistor.

CLOCK (input): pin 22 (AY-3-8910)

pin 15 (AY-3-8912)

This TTL-compatible input supplies the timing reference for the Tone, Noise and Envelope Generators.

B DIR, BC2, BC1 (inputs): pins 27,28,29 (AY-3-8910)

pins 18,19,20 (AY-3-8912)

Bus DIRection, Bus Control 2,1

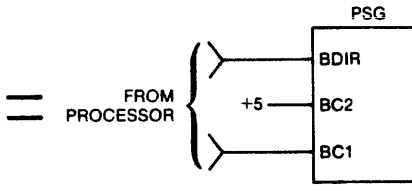
These bus control signals are generated directly by GI's CP1600 series of microprocessors to control all external and internal bus operations in the PSG. When using a processor other than the CP1600, these signals can be provided either by comparable bus signals or by simulating the signals on I/O lines of the processor. The PSG decodes these signals as illustrated in the following:

2.3 Pin Functions (cont.)

BDIR	BC2	BC1	CP1600 FUNCTION	PSG FUNCTION
0	0	0	NACT	INACTIVE. See 010 (IAB) below.
0	0	1	ADAR	LATCH ADDRESS. See 111 (INTAK) below.
0	1	0	IAB	INACTIVE. The PSG/CPU bus is inactive. DA7--DA0 are in a high impedance state.
0	1	1	DTB	READ FROM PSG. This signal causes the contents of the register which is currently addressed to appear on the PSG/CPU bus. DA7--DA0 are in the output mode.
1	0	0	BAR	LATCH ADDRESS. See 111 (INTAK) below.
1	0	1	DW	INACTIVE. See 010 (IAB) above.
1	1	0	DWS	WRITE TO PSG. This signal indicates that the bus contains register data which should be latched into the currently addressed register. DA7--DA0 are in the input mode.
1	1	1	INTAK	LATCH ADDRESS. This signal indicates that the bus contains a register address which should be latched in the PSG. DA7--DA0 are in the input mode.

While interfacing to a processor other than the CP1600 would simply require simulating the above decoding, the redundancies in the PSG functions vs. bus control signals can be used to advantage in that only four of the eight possible decoded bus functions are required by the PSG. This could simplify the programming of the bus control signals to the following, which would only require that the processor generate two bus control signals (BDIR and BC1, with BC2 tied to +5V):

BDIR	BC2	BC1	PSG FUNCTION
0	1	0	INACTIVE.
0	1	1	READ FROM PSG.
1	1	0	WRITE TO PSG.
1	1	1	LATCH ADDRESS.



ANALOG CHANNEL A, B, C (outputs): pins 4, 3, 38 (AY-3-8910)
pins 5, 4, 1 (AY-3-8912)

Each of these signals is the output of its corresponding D/A Converter, and provides an up to 1V peak-peak signal representing the complex sound waveshape generated by the PSG.

IOA7--IOA0 (input/output): pins 14--21 (AY-3-8910)
pins 7--14 (AY-3-8912)

IOB7--IOB0 (input/output): pins 6--13 (AY-3-8910)
(not provided on AY-3-8912)

Input/Output A7--A0, B7--B0

Each of these two parallel input/output ports provides 8 bits of parallel data to/from the PSG/CPU bus from/to any external devices connected to the IOA or IOB pins. Each pin is provided with an on-chip pull-up resistor, so that when in the "input" mode, all pins will read normally high. Therefore, the recommended method for scanning external switches, for example, would be to ground the input bit.

TEST 1: pin 39 (AY-3-8910)
pin 2 (AY-3-8912)

TEST 2: pin 26 (AY-3-8910)
(not connected on AY-3-8912)

These pins are for GI test purposes only and should be left open—do not use as tie-points.

V_{cc}: pin 40 (AY-3-8910)
pin 3 (AY-3-8912)

Nominal +5Volt power supply to the PSG.

V_{ss}: pin 1 (AY-3-8910)
pin 6 (AY-3-8912)

Ground reference for the PSG.

2.4 Bus Timing

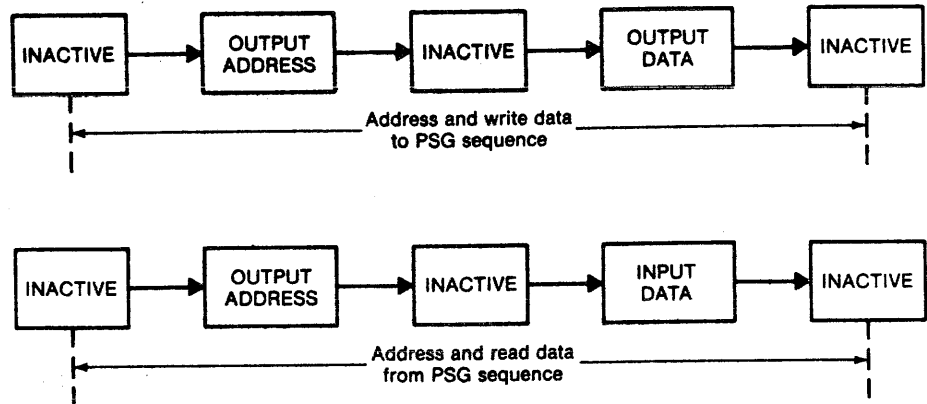
Since the PSG functions are controlled by commands from the system processor, the common data/address bus (DA7--DA0) requires definition as to its function at any particular time. This is accomplished by the processor issuing bus control signals, previously described, defining the state of the bus; the PSG then decodes these signals to perform the requested task.

The conditioning of these bus control signals by the processor is the same as if the processor were interacting with RAM: (1) the processor outputs a memory address; and (2) the processor either outputs or inputs data to/from the memory. The "memory" in this case is the PSG's array of 16 read/write control registers.

The timing relationships in issuing the bus control signals relative to the data or address signals on the bus are reviewed in general in the following section, and in detail in Section 7, Electrical Specifications.

25 State Timing

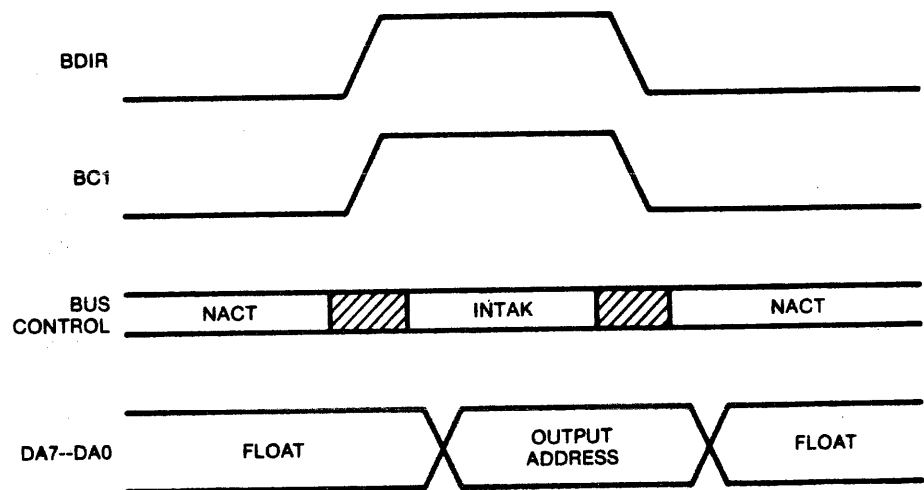
While the state flow for many microprocessors can be somewhat involved for certain operations, the sequence of events necessary to control the PSG is simple and straightforward. Each of the three major state sequences (Latch Address, Write to PSG, and Read from PSG) consists of several operations (indicated below by rectangular blocks), defined by the pattern of bus control signals (BDIR, BC2, BC1).



The functional operation and relative timing of the PSG control sequences are described in the following paragraphs (in all examples, BC2 has been assumed to be tied to logic "1", +5V).

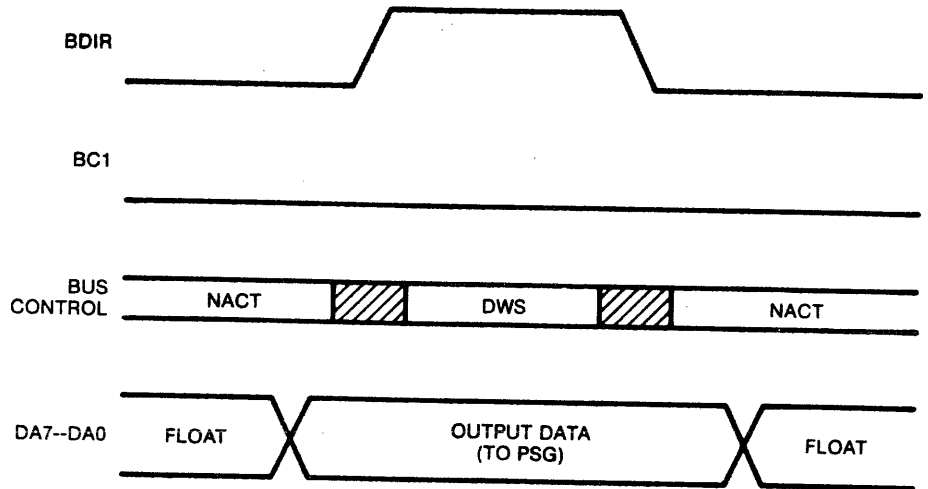
2.5.1 ADDRESS PSG REGISTER SEQUENCE

The "Latch Address" sequence is normally an integral part of the write or read sequences, but for simplicity is illustrated here as an individual sequence. Depending on the processor used the program sequence will normally require four principal microstates: (1) send NACT (inactive); (2) send INTAK (latch address); (3) put address on bus; (4) send NACT (inactive). [Note: within the timing constraints detailed in Section 7, steps (2) and (3) may be interchanged.]



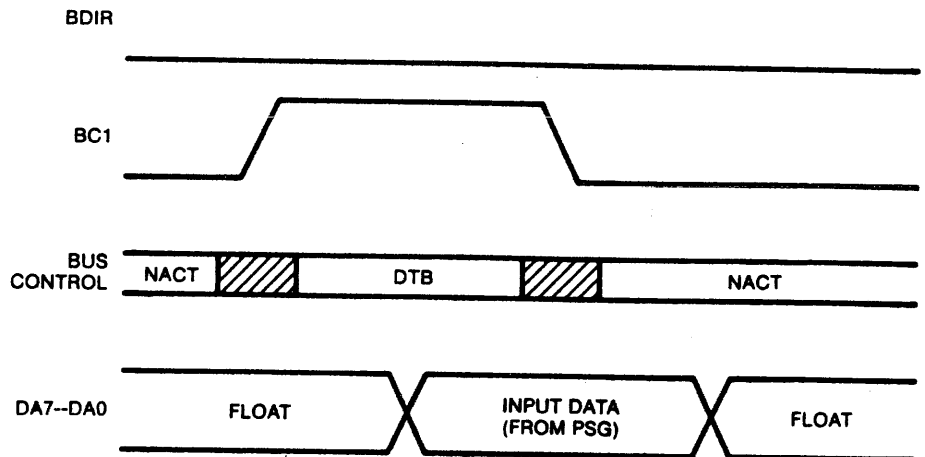
2.5.2 WRITE DATA TO PSG SEQUENCE

The "Write to PSG" sequence, which would normally follow immediately after an address sequence, requires four principal microstates: (1) send NACT (inactive); (2) put data on bus; (3) send DWS (write to PSG); (4) send NACT (inactive).



2.5.3 READ DATA FROM PSG SEQUENCE

As with the "Write to PSG" sequence, the "Read from PSG" sequence would also normally follow immediately after an address sequence. The four principal microstates of the read sequence are: (1) send NACT (inactive); (2) send DTB (read from PSG); (3) read data on bus; (4) send NACT (inactive).



2.5.4 WRITE TO/READ FROM I/O PORT SEQUENCE

Since the two I/O Ports (A and B) each have an 8-bit register assigned as a data store, writing to or reading from either port is identical to writing or reading to any other register. Hence, the state sequences are exactly the same as described in the preceding paragraphs.

3 OPERATION

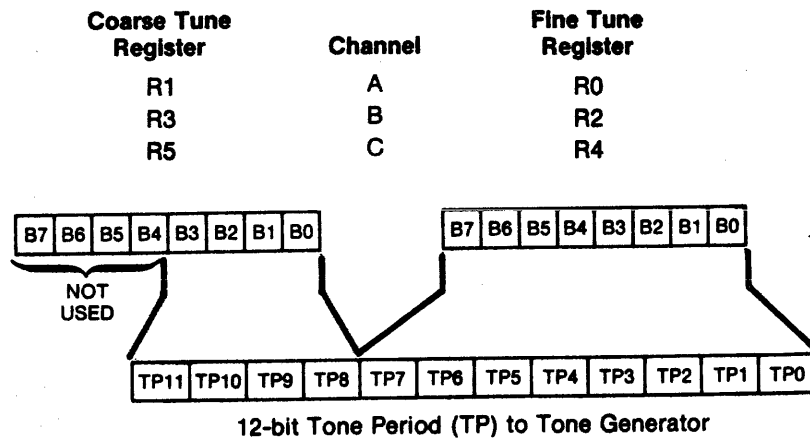
Since all functions of the PSG are controlled by the host processor via a series of register loads, a detailed description of the PSG operation can best be accomplished by relating each PSG function to the control of its corresponding register. The function of creating or programming a specific sound or sound effect logically follows the control sequence listed:

Section	Operation	Registers	Function
3.1	Tone Generator Control	R0--R5	Program tone periods.
3.2	Noise Generator Control	R6	Program noise period.
3.3	Mixer Control	R7	Enable tone and/or noise on selected channels.
3.4	Amplitude Control	R10--R12	Select "fixed" or "envelope-variable" amplitudes.
3.5	Envelope Generator Control	R13--R15	Program envelope period and select envelope pattern.

3.1 Tone Generator Control

(Registers R0, R1, R2, R3, R4, R5)

The frequency of each square wave generated by the three Tone Generators (one each for Channels A, B, and C) is obtained in the PSG by first counting down the input clock by 16, then by further counting down the result by the programmed 12-bit Tone Period value. Each 12-bit value is obtained in the PSG by combining the contents of the relative Coarse and Fine Tune registers, as illustrated in the following:



Note that the 12-bit value programmed in the combined Coarse and Fine Tune registers is a period value—the higher the value in the registers, the lower the resultant tone frequency.

Note also that due to the design technique used in the Tone Period count-down, the lowest period value is 000000000001 (divide by 1) and the highest period value is 111111111111 (divide by 4,095₁₀).

The equations describing the relationship between the desired output tone frequency and the input clock frequency and Tone Period value are:

$$(a) f_T = \frac{f_{\text{LOCK}}}{16TP_{10}} \quad (b) TP_{10} = 256CT_{10} + FT_{10}$$

Where:

- f_T = desired tone frequency
- f_{LOCK} = input clock frequency
- TP_{10} = decimal equivalent of the Tone Period bits TP11--TP0.
- CT_{10} = decimal equivalent of the Coarse Tune register bits B3--B0 (TP11--TP8)
- FT_{10} = decimal equivalent of the Fine Tune register bits B7--B0 (TP7--TP0)

From the above equations it can be seen that the tone frequency can range from a low of $\frac{f_{\text{LOCK}}}{65,520}$ (wherein: $TP_{10}=4,095_{10}$) to a high of $\frac{f_{\text{LOCK}}}{16}$ (wherein: $TP_{10}=1$). Using a 2 MHz input clock, for example, would produce a range of tone frequencies from 30.5 Hz to 125 kHz.

To calculate the values for the contents of the Tone Period Coarse and Fine Tune registers, given the input clock and the desired output tone frequencies, we simply rearrange the above equations, yielding:

$$(a) TP_{10} = \frac{f_{\text{LOCK}}}{16f_T} \quad (b) CT_{10} + \frac{FT_{10}}{256} = \frac{TP_{10}}{256}$$

Example 1: $f_T = 1\text{kHz}$
 $f_{\text{LOCK}} = 2\text{MHz}$

$$TP_{10} = \frac{2 \times 10^6}{16(1 \times 10^3)} = 125$$

Substituting this result into equation (b):

$$CT_{10} + \frac{FT_{10}}{256} = \frac{125}{256}$$

$$\therefore CT_{10} = 0 = 0000 \text{ (B3--B0)}$$

$$FT_{10} = 125_{10} = 01111101 \text{ (B7--B0)}$$

Example 2: $f_T = 100\text{Hz}$
 $f_{\text{LOCK}} = 2\text{MHz}$

$$TP_{10} = \frac{2 \times 10^6}{16(1 \times 10^2)} = 1250$$

Substituting this result into equation (b):

$$CT_{10} + \frac{FT_{10}}{256} = \frac{1250}{256} = 4 + \frac{226}{256}$$

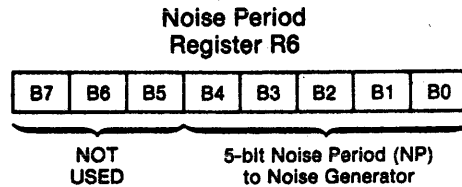
$$\therefore CT_{10} = 4_{10} = 0100 \text{ (B3--B0)}$$

$$FT_{10} = 226_{10} = 11100010 \text{ (B7--B0)}$$

3.2 Noise Generator Control

(Register R6)

The frequency of the noise source is obtained in the PSG by first counting down the input clock by 16, then by further counting down the result by the programmed 5-bit Noise Period value. This 5-bit value consists of the lower 5 bits (B4--B0) of register R6, as illustrated in the following:



Note that the 5-bit value in R11 is a period value—the higher the value in the register, the lower the resultant noise frequency. Note also that, as with the Tone Period, the lowest period value is 00001 (divide by 1); the highest period value is 11111 (divide by 31₁₀).

The noise frequency equation is:

$$f_N = \frac{f_{\text{CLOCK}}}{16 \text{ NP}_{10}}$$

Where: f_N = desired noise frequency

f_{CLOCK} = input clock frequency

NP_{10} = decimal equivalent of the Noise Period register bits B4--B0.

From the above equation it can be seen that the noise frequency can range from a low of $\frac{f_{\text{CLOCK}}}{496}$ (wherein: $\text{NP}_{10} = 31_{10}$) to a high of $\frac{f_{\text{CLOCK}}}{16}$ (wherein: $\text{NP}_{10} = 1$). Using a 2 MHz input clock, for example, would produce a range of noise frequencies from 4 kHz to 125 kHz.

To calculate the value for the contents of the Noise Period register, given the input clock and the desired output noise frequencies, we simply rearrange the above equation, yielding:

$$\text{NP}_{10} = \frac{f_{\text{CLOCK}}}{16 f_N}$$

3.3 Mixer Control-I/O Enable

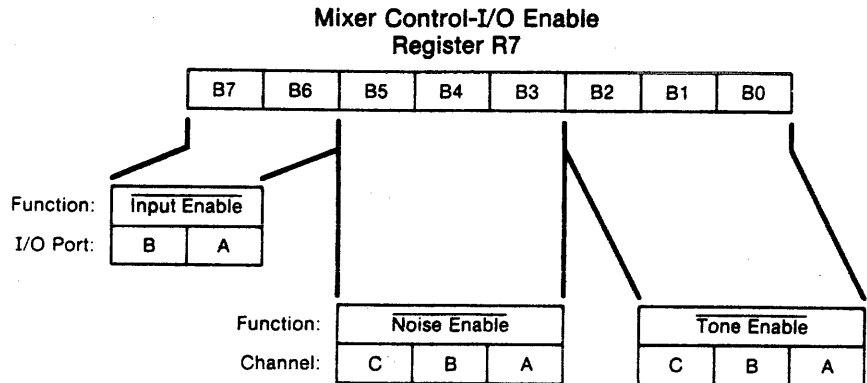
(Register R7)

Register 7 is a multi-function Enable register which controls the three Noise/Tone Mixers and the two general purpose I/O Ports.

The Mixers, as previously described, combine the noise and tone frequencies for each of the three channels. The determination of combining neither/either/both noise and tone frequencies on each channel is made by the state of bits B5--B0 of R7.

The direction (input or output) of the two general purpose I/O Ports (IOA and IOB) is determined by the state of bits B7 and B6 of R7.

These functions are illustrated in the following:



Noise Enable Truth Table:

R7 Bits			Noise Enabled on Channel		
B5	B4	B3	C	B	A
0	0	0	C	B	A
0	0	1	C	B	—
0	1	0	C	—	A
0	1	1	C	—	—
1	0	0	—	B	A
1	0	1	—	B	—
1	1	0	—	—	A
1	1	1	—	—	—

Tone Enable Truth Table:

R7 Bits			Tone Enabled on Channel		
B2	B1	B0	C	B	A
0	0	0	C	B	A
0	0	1	C	B	—
0	1	0	C	—	A
0	1	1	C	—	—
1	0	0	—	B	A
1	0	1	—	B	—
1	1	0	—	—	A
1	1	1	—	—	—

I/O Port Truth Table:

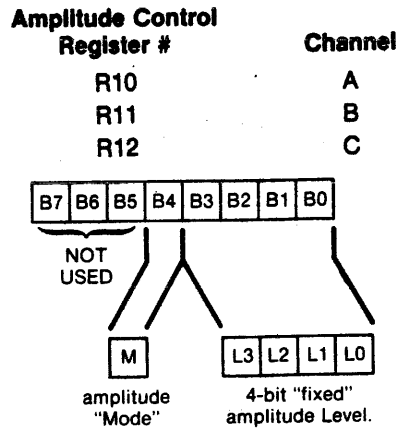
R7 Bits		I/O Port Status	
B7	B6	IOB	IOA
0	0	Input	Input
0	1	Input	Output
1	0	Output	Input
1	1	Output	Output

NOTE: Disabling noise and tone does not turn off a channel. Turning a channel off can only be accomplished by writing all zeroes into the corresponding Amplitude Control register, R10, R11, or R12 (see Section 3.4).

3.4 Amplitude Control

(Registers R10, R11, R12)

The amplitudes of the signals generated by each of the three D/A Converters (one each for Channels A, B, and C) is determined by the contents of the lower 5 bits (B4--B0) of registers R10, R11, and R12 as illustrated in the following:



The amplitude "mode" (bit M) selects either fixed level amplitude (M=0) or variable level amplitude (M=1). It follows then that bits L3--L0, defining the value of a "fixed" level amplitude, are only active when M=0. When fixed level amplitude is selected, it is "fixed" only in the sense that the amplitude level is under the direct control of the system processor (via bits D3--D0). Varying the amplitude when in this "fixed" amplitude mode requires in each instance the direct intervention of the system processor via an address latch/write data sequence to modify the D3--D0 data.

When M=1 (select "variable" level amplitudes), the amplitude of each channel is determined by the envelope pattern as defined by the Envelope Generator's 4-bit output E3 E2 E1 E0.

The amplitude "mode" (bit M) can also be thought of as an "envelope enable" bit; i.e., when M=0 the envelope is not used, and when M=1 the envelope is enabled. (A full description of the Envelope Generator function follows in Section 3.5).

The full chart describing all combinations of the 5-bit Amplitude Control is as follows:

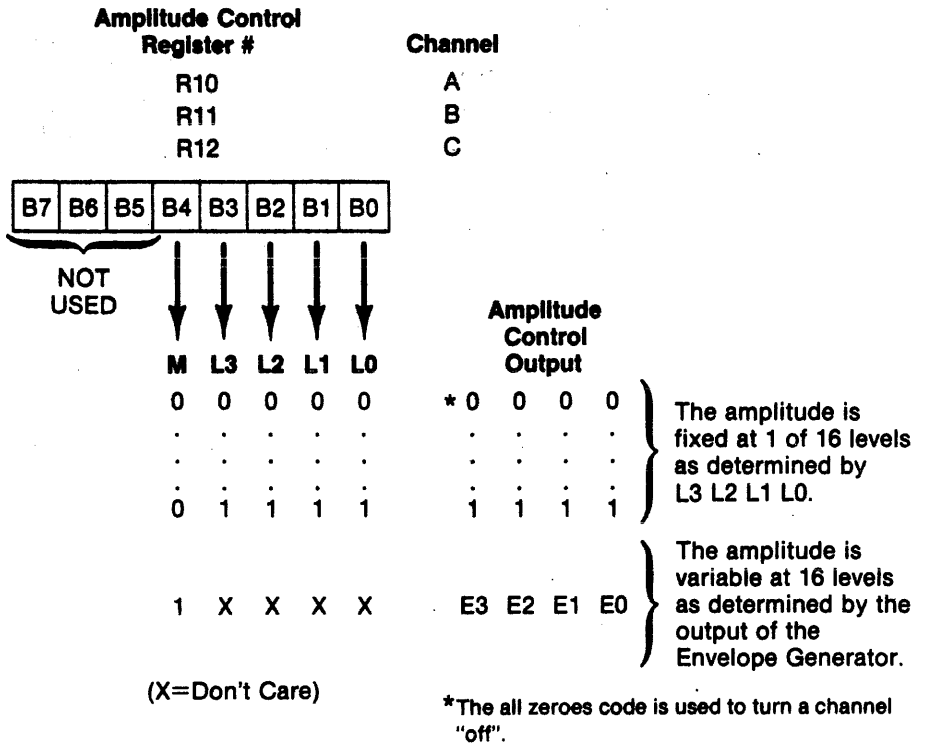
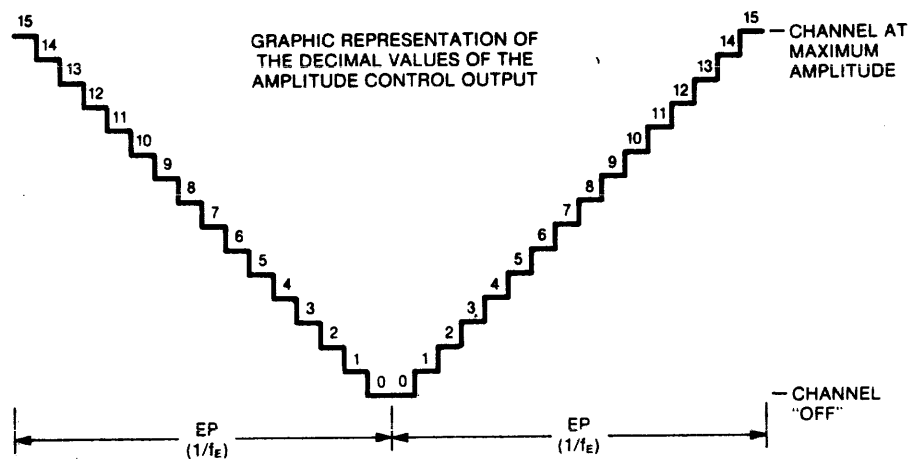


Fig. 6 graphically illustrates a selection of variable level (envelope-controlled) amplitude where the 16 levels directly reflect the output of the Envelope Generator. A fixed level amplitude would correspond to only one of the levels shown, with the level directly determined by the decimal equivalent of bits L3 L2 L1 L0.

Fig. 6 VARIABLE AMPLITUDE CONTROL (M=1)



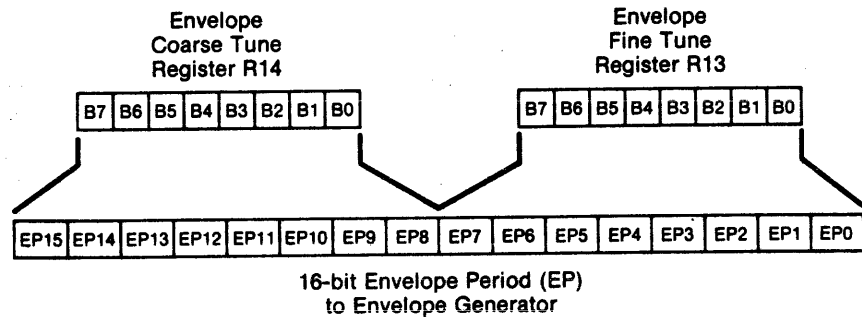
3.5 Envelope Generator Control

(Registers R13, R14, R15)

To accomplish the generation of fairly complex envelope patterns, two independent methods of control are provided in the PSG: first, it is possible to vary the frequency of the envelope using registers R13 and R14; and second, the relative shape and cycle pattern of the envelope can be varied using register R15. The following paragraphs explain the details of the envelope control functions, describing first the envelope period control and then the envelope shape/cycle control.

3.5.1 ENVELOPE PERIOD CONTROL (Registers R13, R14)

The frequency of the envelope is obtained in the PSG by first counting down the input clock by 256, then by further counting down the result by the programmed 16-bit Envelope Period value. This 16-bit value is obtained in the PSG by combining the contents of the Envelope Coarse and Fine Tune registers, as illustrated in the following:



Note that the 16-bit value programmed in the combined Coarse and Fine Tune registers is a period value—the higher the value in the registers, the lower the resultant envelope frequency.

Note also, that as with the Tone Period, the lowest period value is 0000000000000001 (divide by 1); the highest period value is 1111111111111111 (divide by 65,535₁₀).

The envelope frequency equations are:

$$(a) f_E = \frac{f_{\text{CLOCK}}}{256EP_{10}} \qquad (b) EP_{10} = 256CT_{10} + FT_{10}$$

- Where:
- f_E = desired envelope frequency
 - f_{CLOCK} = input clock frequency
 - EP_{10} = decimal equivalent of the Envelope Period bits EP15--EP0
 - CT_{10} = decimal equivalent of the Coarse Tune register bits B7--B0 (EP15--EP8)
 - FT_{10} = decimal equivalent of the Fine Tune register bits B7--B0 (EP7--EP0)

From the above equation it can be seen that the envelope frequency can range from a low of $\frac{f_{\text{CLOCK}}}{16,778,960_{10}}$ (wherein: $EP_{10} = 65,535_{10}$) to a high of $\frac{f_{\text{CLOCK}}}{256}$ (wherein: $EP_{10} = 1$). Using a 2 MHz clock, for example, would produce a range of envelope frequencies from 0.12 Hz to 7812.5 Hz.

To calculate the values for the contents of the Envelope Period Coarse and Fine Tune registers, given the input clock and the desired envelope frequencies, we rearrange the above equations, yielding:

$$(a) EP_{10} = \frac{f_{\text{CLOCK}}}{256f_E} \qquad (b) CT_{10} + \frac{FT_{10}}{256} = \frac{EP_{10}}{256}$$

Example: $f_E = 0.5 \text{ Hz}$
 $f_{\text{CLOCK}} = 2 \text{ MHz}$

$$EP_{10} = \frac{2 \times 10^6}{256(0.5)} = 15,625$$

Substituting this result into equation (b):

$$CT_{10} + \frac{FT_{10}}{256} = \frac{15,625}{256} = 61 + \frac{9}{256}$$

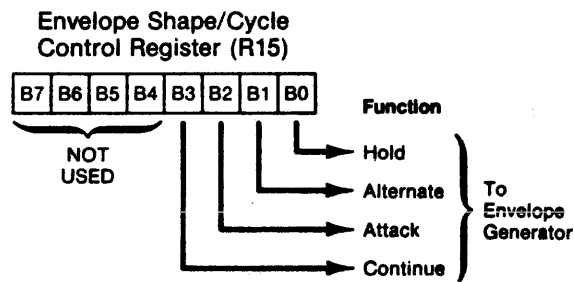
$$CT_{10} = 61_{10} = 00111101 \text{ (B7--B0)}$$

$$FT_{10} = 9_{10} = 00001001 \text{ (B7--B0)}$$

3.5.2 ENVELOPE SHAPE/CYCLE CONTROL (Register R15)

The Envelope Generator further counts down the envelope frequency by 16, producing a 16-state per cycle envelope pattern as defined by its 4-bit counter output, E3 E2 E1 E0. The particular shape and cycle pattern of any desired envelope is accomplished by controlling the count pattern (count up/count down) of the 4-bit counter and by defining a single-cycle or repeat-cycle pattern.

This envelope shape/cycle control is contained in the lower 4 bits (B3--B0) of register R15. Each of these 4 bits controls a function in the envelope generator, as illustrated in the following:



The definition of each function is as follows:

Hold when set to logic "1", limits the envelope to one cycle, holding the last count of the envelope counter (E3--E0=0000 or 1111, depending on whether the envelope counter was in a count-down or count-up mode, respectively).

Alternate when set to logic "1", the envelope counter reverses count direction (up-down) after each cycle.

NOTE: When both the Hold bit and the Alternate bit are ones, the envelope counter is reset to its initial count before holding.

3.5 Envelope Generator Control (cont.)

- Attack** when set to logic "1", the envelope counter will count up (attack) from E3 E2 E1 E0=0000 to E3 E2 E1 E0=1111; when set to logic "0", the envelope counter will count down (decay) from 1111 to 0000.
- Continue** when set to logic "1", the cycle pattern will be as defined by the Hold bit; when set to logic "0", the envelope generator will reset to 0000 after one cycle and hold at that count.

To further describe the above functions could be accomplished by numerous charts of the binary count sequence of E3 E2 E1 E0 for each combination of Hold, Alternate, Attack and Continue. However, since these outputs are used (when selected by the Amplitude Control registers) to amplitude modulate the output of the Mixers, a better understanding of their effect can be accomplished via a graphic representation of their value for each condition selected, as illustrated in Figs. 7 and 8.

Fig. 7 ENVELOPE SHAPE/CYCLE CONTROL

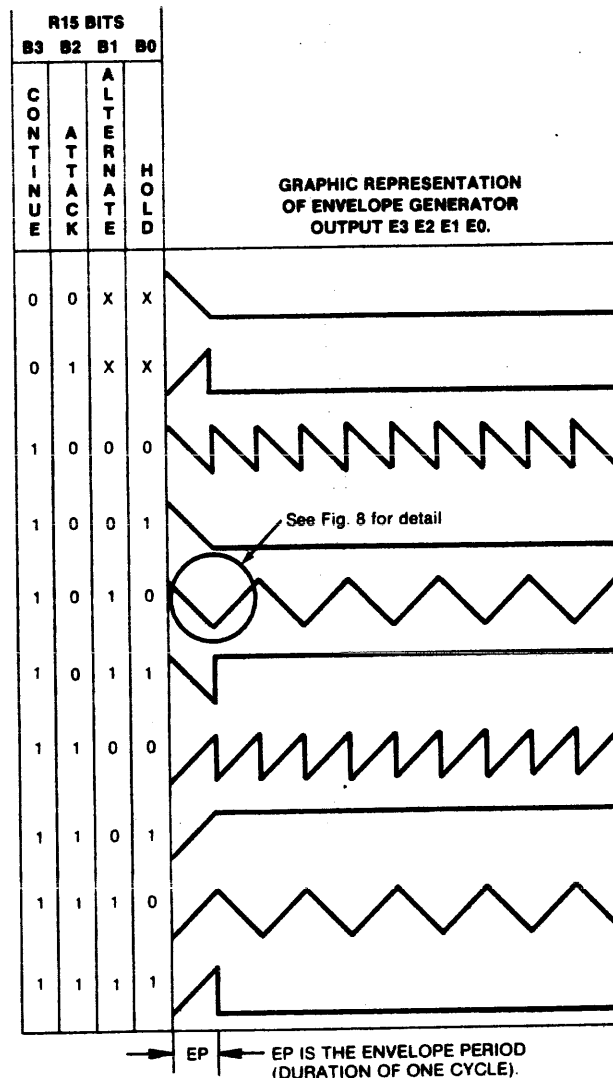
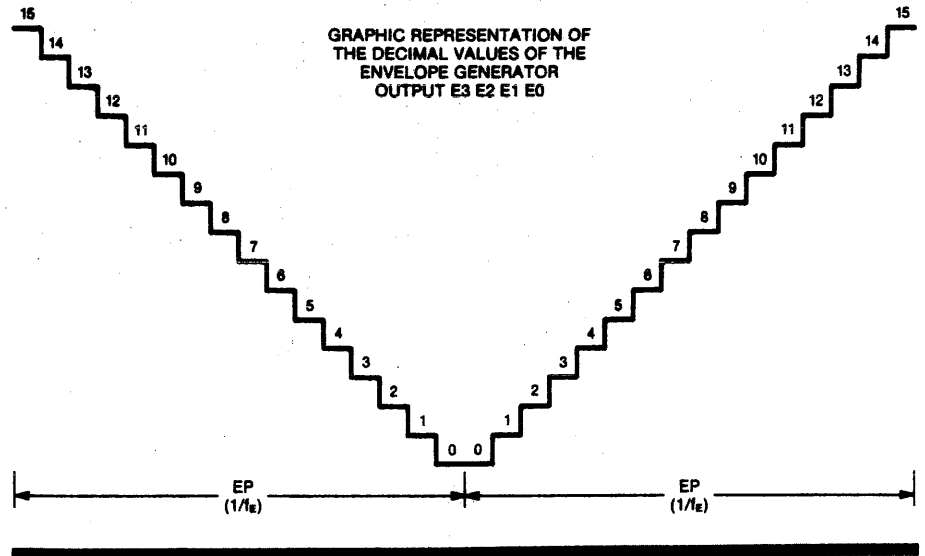


Fig. 8 DETAIL OF TWO CYCLES OF Fig. 7
(ref. waveform "1010" in Fig. 7)



3.6 I/O Port Data Store

(Registers R16, R17)

Registers R16 and R17 function as intermediate data storage registers between the PSG/CPU data bus (DA0--DA7) and the two I/O ports (IOA7--IOA0 and IOB7--IOB0). Both ports are available in the AY-3-8910; only I/O Port A is available in the AY-3-8912. Using registers R16 and R17 for the transfer of I/O data has no effect at all on sound generation.

To output data from the CPU bus to a peripheral device connected to I/O Port A would require only the following steps:

1. Latch address R7 (select $\overline{\text{Enable}}$ register)
2. Write data to PSG (setting B6 of R7 to "1")
3. Latch address R16 (select IOA register)
4. Write data to PSG (data to be output on I/O Port A)

To input data from I/O Port A to the CPU bus would require the following:

1. Latch address R7 (select $\overline{\text{Enable}}$ register)
2. Write data to PSG (setting B6 to R7 to "0")
3. Latch address R16 (select IOA register)
4. Read data from PSG (data from I/O Port A)

Note that once loaded with data in the output mode, the data will remain on the I/O port(s) until changed either by loading different data, by applying a reset (grounding the Reset pin), or by switching to the input mode.

Note also that when in the input mode, the contents of registers R16 and/or R17 will follow the signals applied to the I/O port(s). However, transfer of this data to the CPU bus requires a "read" operation as described above.

3.7 D/A Converter Operation

Since the primary use of the PSG is to produce sound for the highly imperfect amplitude detection mechanism of the human ear, the D/A conversion is performed in logarithmic steps with a normalized voltage range of from 0 to 1 Volt. The specific amplitude control of each of the three D/A Converters is accomplished by the three sets of 4-bit outputs of the Amplitude Control block, while the Mixer outputs provide the base signal frequency (Noise and/or Tone).

Fig. 9 illustrates the D/A Converter output which would result if noise and tones were disabled and an envelope-controlled variable amplitude were selected.

Figs. 10 through 13 illustrate other typical output waveforms.

Fig. 9 D/A CONVERTER OUTPUT (ref. Fig. 6)

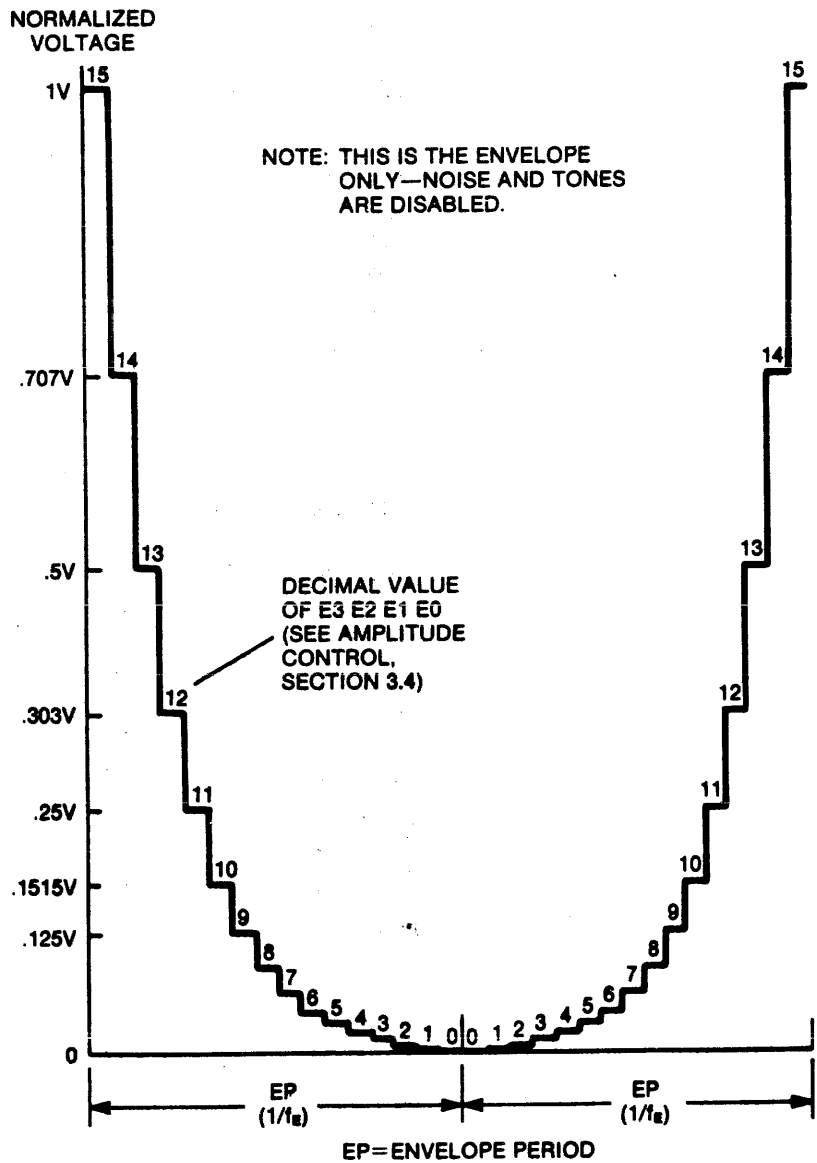


Fig. 10 SINGLE TONE WITH ENVELOPE SHAPE/CYCLE PATTERN 1000
(R0=14₈, R1=37₈, R7=76₈, R12=20₈, R15=10₈, all other registers=0)

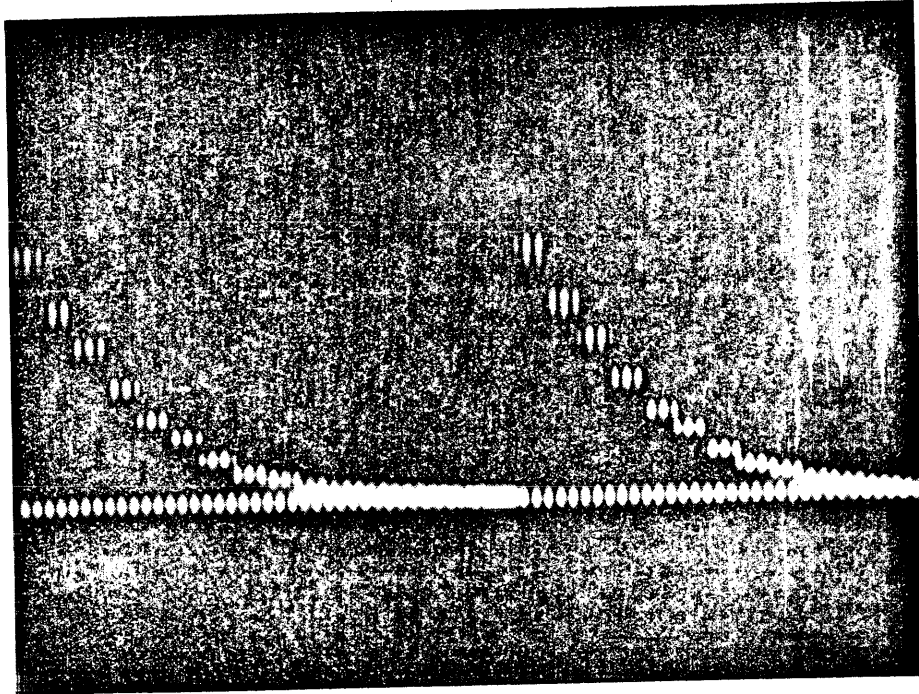


Fig. 11 SINGLE TONE WITH ENVELOPE SHAPE/CYCLE PATTERN 1100
(R15=14₈, all other registers same as Fig. 10)

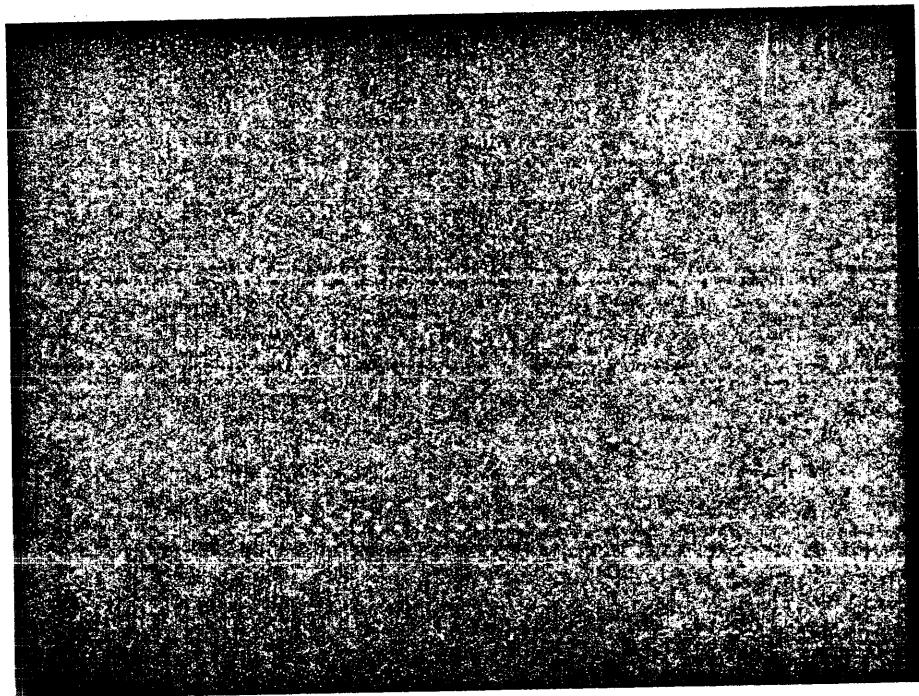


Fig. 12 SINGLE TONE WITH ENVELOPE SHAPE/CYCLE PATTERN 1010
(R15=12₈, all other registers same as Fig. 10)

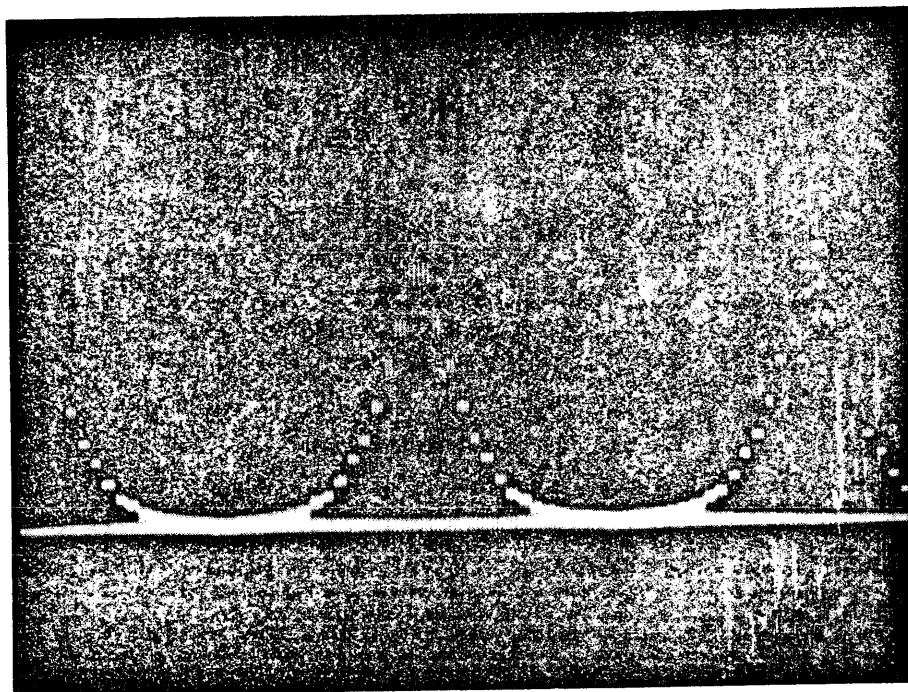
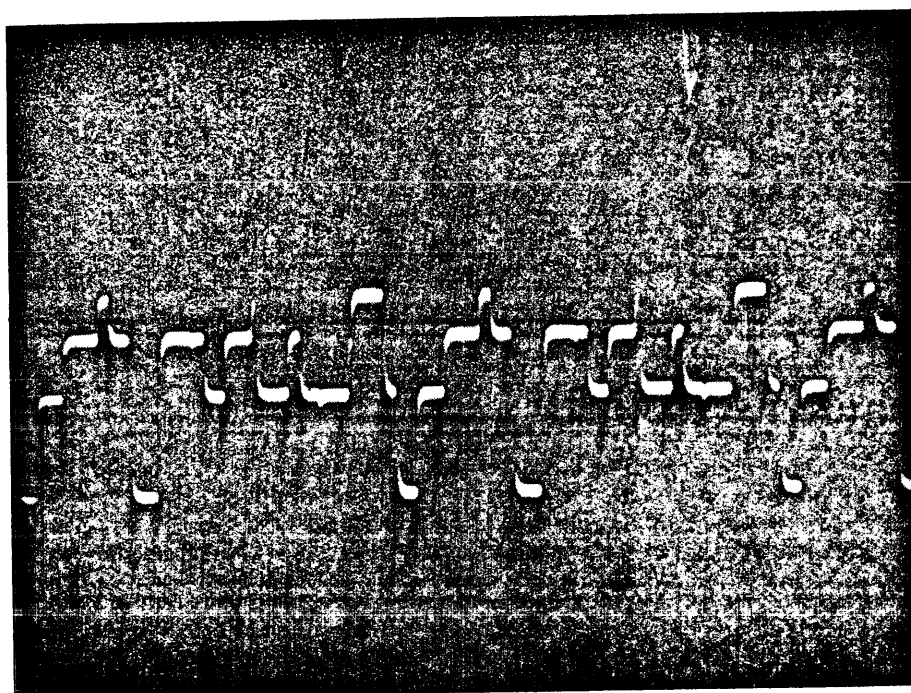


Fig. 13 MIXTURE OF THREE TONES WITH FIXED AMPLITUDES



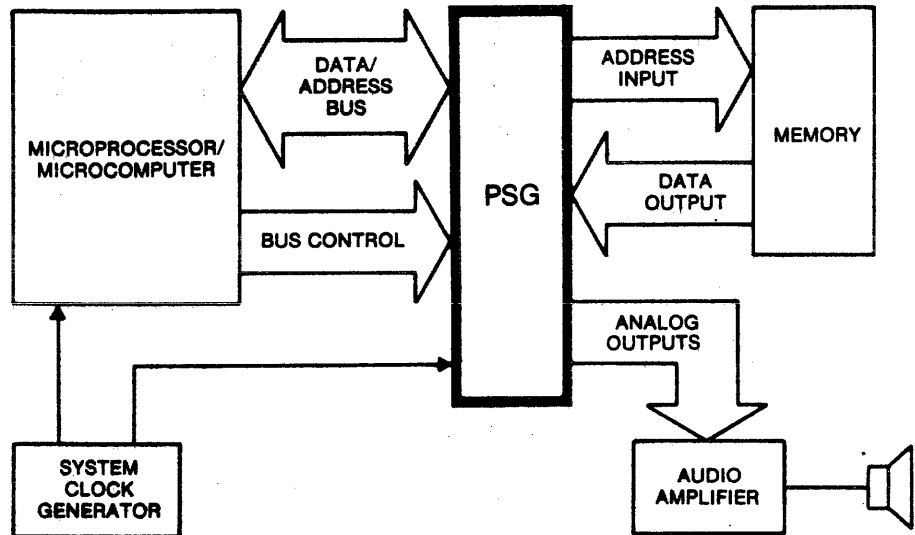
4 INTERFACING

4.1 Introduction

Since the AY-3-8910/8912 PSG must be used with support components, interfacing to the circuit is an obvious requirement. The PSG is designed to be controlled by a microprocessor or microcomputer, and drive directly into analog audio circuitry. It provides the link between the computer and a speaker to provide sounds or sound effects derived from digital inputs.

The following paragraphs provide examples and illustrations showing the ease with which an AY-3-8910/8912 Programmable Sound Generator may be utilized in a microprocessor/microcomputer system.

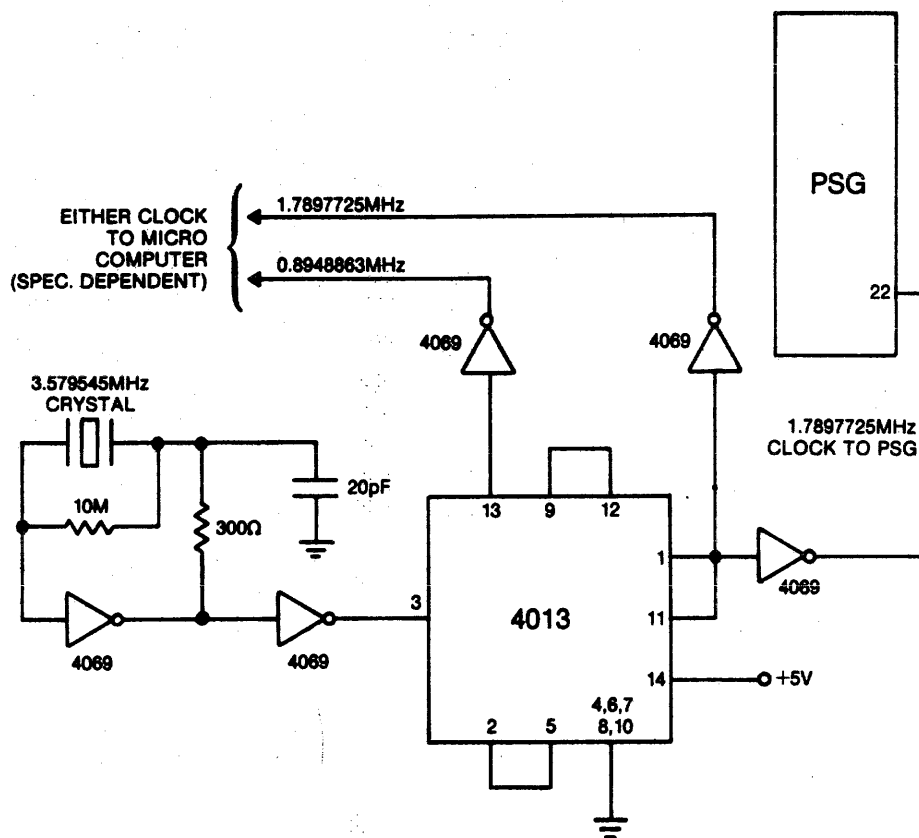
Fig. 14 SYSTEM BLOCK DIAGRAM



4.2 Clock Generation

An economical solution to providing a system clock is shown in Fig. 15. It consists of a 3.579545MHz standard color burst crystal, a CD4069 CMOS inverter, and a CD4013 to divide the color burst frequency in half. The clock produced for the PSG runs at a 1.7897725MHz rate. Depending on the microcomputer used, its clock should be selected within its specified value.

Fig. 15 CLOCK GENERATION

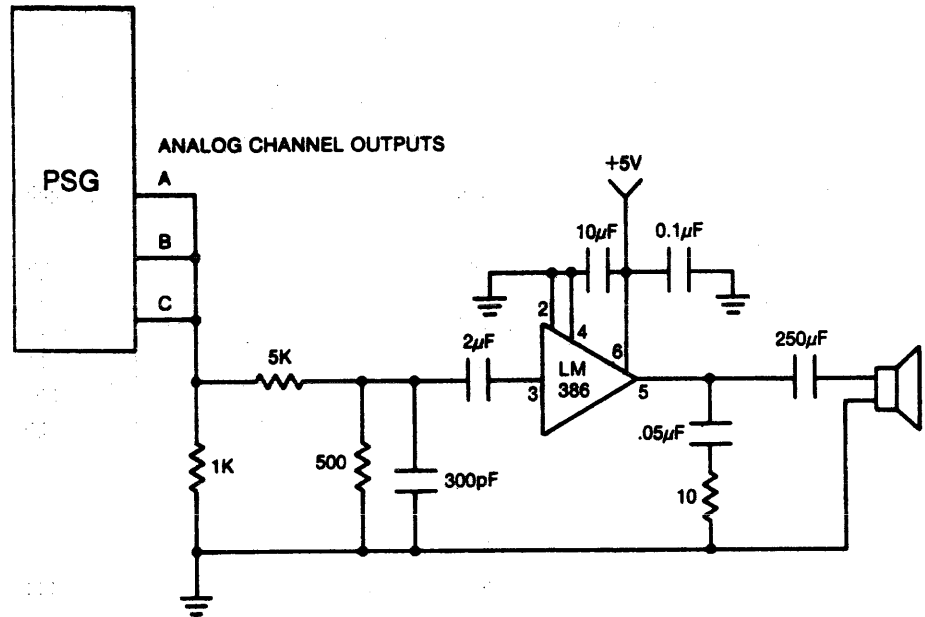


4.3 Audio Output Interface

Fig. 16 illustrates the audio output connections to a commercially available LM386 audio amplifier. It shows channels A, B, and C summed together to enable complex waveforms to be composed and amplified through a single external amplifier. These channels may be individually amplified through separate channels for more exotic sound systems.

Each output channel is individually controlled by separate amplitude registers (R10, R11, R12) and an enable register (R7) in the PSG.

Fig. 16 AUDIO OUTPUT INTERFACE



4.4 External Memory Access

The ROM or PROM shown connected to the PSG in Fig. 17 illustrates an option for providing additional data information for processor support. The two I/O registers within the PSG are used in this case to address the memory via I/O Port A (8 Bits) and read data from the memory via I/O Port B (8 Bits).

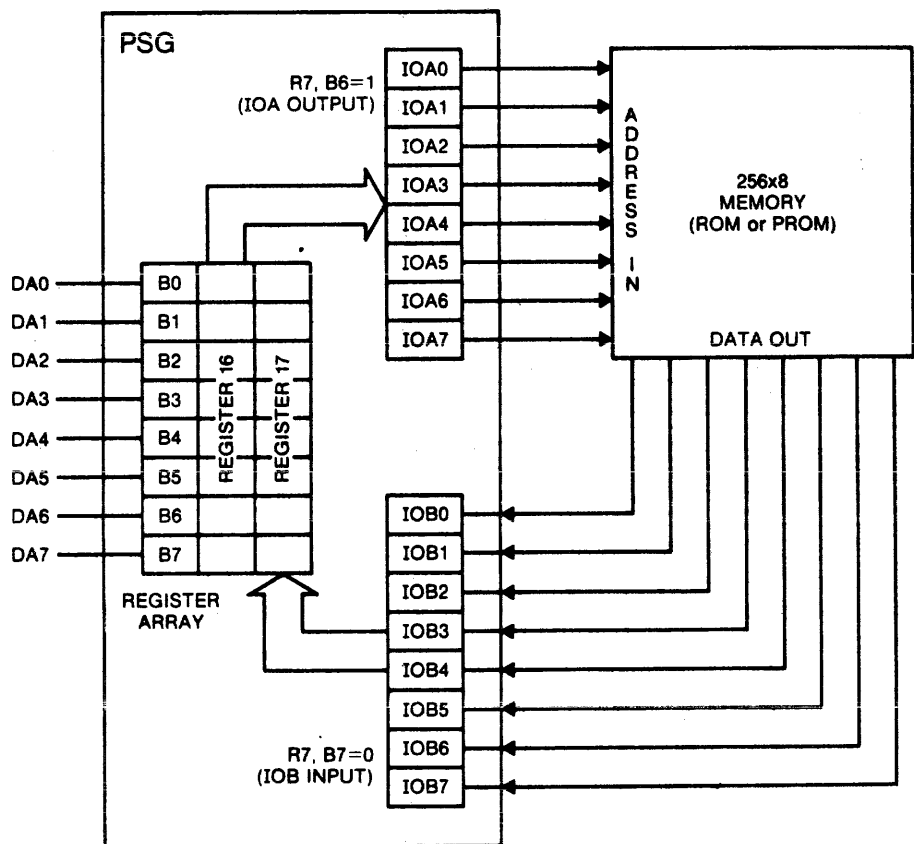
An example of the bus control sequence to address and read an external memory connected to I/O ports A and B would be as follows (Assume Port A addresses and Port B reads):

Bus Control	Bus Codes			Explanation of Bus Data (DA7--DA0)
	B DIR	BC2	BC1	
Latch address	1	1	1	00001111: Latch R7 to program I/O Ports
Write to PSG	1	1	0	01000000: Set B7, B6 to 0, 1 respectively
Latch address	1	1	1	00001110: Latch R16 to address memory
Write to PSG	1	1	0	00000001: Address data to memory
Latch address	1	1	1	00001111: Latch R17 to read memory
Read from PSG	0	1	1	XXXXXXXX: Memory data contained in R17

NOTE: BC2 in the above Bus Codes may be permanently tied to +5V thus requiring only two bus control lines for all control operations (refer to Section 2.3 for a complete explanation).

Also, RAM or EAROM may be used in place of the ROM or PROM shown by altering the program to use PORT B as an I/O. Port B then will be able to write data as an output and read data as an input.

Fig. 17 EXTERNAL MEMORY ACCESS



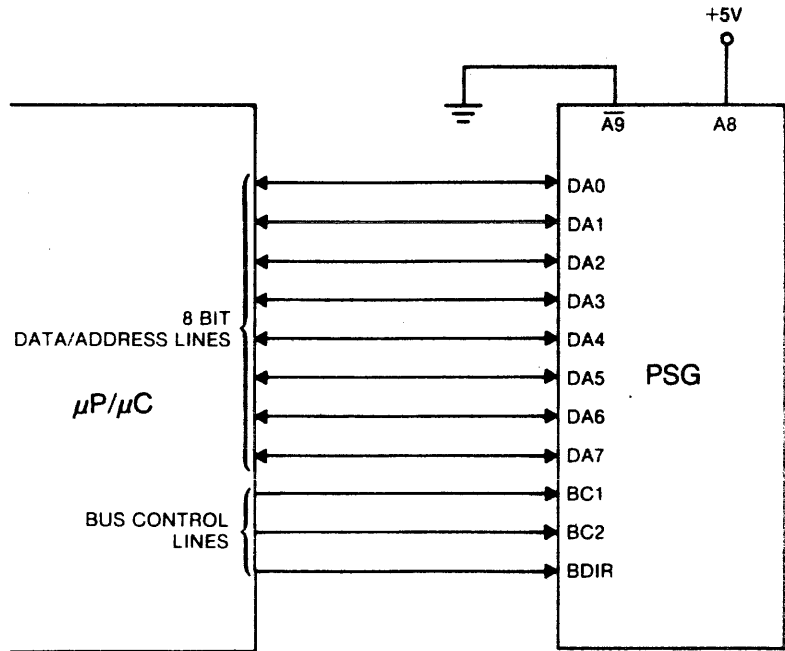
4.5 Microprocessor/ Microcomputer Interface

In Fig. 18, the lines identified DA7--DA0 are the input/output bus bits 7--0. This 8 bit bus is used to pass all data and address information between the AY-3-8910/8912 and the system processor.

BC1, BC2 and BDIR are bus control signals generated by the processor to direct all bus operations. These operations are identified as Latch Address, Write to PSG, Read from PSG, and Inactive.

The following Sections detail specific interfaces to several popular microprocessors/microcomputers.

Fig. 18 MICROPROCESSOR/MICROCOMPUTER INTERFACE



4.6 Interfacing to the PIC 1650

Fig. 19 shows the schematic of an AY-3-8910 demonstrator circuit. This configuration uses a PIC 1650 as the main controller in the circuit. The PIC 1650 is used to scan the keyboard, fetch data from the PROMs, write data to the AY-3-8910 and provide the timing for the AY-3-8910.

The interfacing is direct since the PIC 1650 and the AY-3-8910 operate with compatible supplies and input/output voltages.

This particular schematic illustrates how a microcomputer with additional memory can produce a stand-alone music and sound effects circuit. The circuit as shown operates with manual keyboard selections.

As Fig. 19 shows, the design for the interface connects directly to the output pins of the 1650 and the BC1, BC2, BDIR pins. The software then has the responsibility of manipulating these signals to signal the PSG to perform the proper address latch, read or write operations.

The program routine in this section illustrates code which is used in a hand-held demonstrator unit. This demonstration unit illustrates the range of PSG capabilities, including music, sound effects and I/O control. Note that the generalized routines perform the address latching before every read for convenience.

The "READ ROM" routine illustrates use of the generalized read and write routines to access the outside world through the PSG to read and write.

4.6.1 WRITE DATA ROUTINE

```
80.          ;WRITE FROM 1650 TO 8910
81.          ;ADDRESS OF 8910 REG IN 'ADDRESS'
82.          ;DATA TO WRITE IN 'DATA'
83. 024 0066 WRIT1 MOVWF  ADDRESS ;
84. 025 1026 WRITE MOVF  ADDRESS,W ;GET REGISTER NO.
85. 026 0045      MOVWF IOA      ;SET ADDRESS
86. 027 1006      MOVF  IOB,W    ;GET PRESENT BC1, BC2, BDIR ETC.
87. 030 7370      ANDLW 370
88. 031 6404      IORLW 4        ;SET BAR
89. 032 0046      MOVWF IOB      ;SEND BAR
90. 033 7370      ANDLW 370
91. 034 0046      MOVWF IOB      ;SEND NACT
92. 035 1027      MOVF  DATA,W
93. 036 0045      MOVWF IOA      ;PUT DATA ON D/A PINS OF 8910
94. 037 1006      MOVF  IOB,W
95. 040 7370      ANDLW 370
96. 041 6406      IORLW 6
97. 042 0046      MOVWF IOB      ;SEND DWS
98. 043 7370      ANDLW 370      ;SET UP NACT
99. 044 0046      MOVWF IOB      ;SEND NACT
100. 045 4000     RET            ;RETURN TO CALLING ROUTINE
```

4.6 Interfacing to the PIC 1650 (cont.)

4.6.2 READ DATA ROUTINE

```

51.          ;ADDRESS OF READ IN REGISTER 'ADDRESS'
52.          ;AFTER READ, INPUT DATA IN REGISTER 'DATA'
53.          ;ENTRANCE READ1 ASSUMES THAT REGISTER NUM IN W
54.
55. 000 0066 READ1 MOVWF ADDRESS ;BYPASS ADDRESS STORE
56. 001 1026 READ  MOVF  ADDRESS,W ;GET REGISTER NO.
57. 002 0045      MOVWF IOA      ;MOVE TO 8910 D/A PINS
58. 003 1006      MOVF  IOB,W     ;GET PRESENT BC1,BC2,BDIR ETC.
59. 004 6404      IORLW 4        ;SET BAR
60. 005 0046      MOVWF IOB      ;SEND BAR
61. 006 7370      ANDLW 370
62. 007 0046      MOVWF IOB      ;SEND NACT
63. 010 6377      MOVLW 377
64. 011 0045      MOVWF IOA      ;SET FOR INPUT
65. 012 1006      MOVF  IOB,W
66. 013 7370      ANDLW 370
67. 014 6403      IORLW 3        ;SET DTB
68. 015 0046      MOVWF IOB      ;SEND DTB
69. 016 1005      MOVF  IOA,W
70. 017 0067      MOVWF DATA    ;SAVE DATA
71. 020 1006      MOVF  IOB,W
72. 021 7370      ANDLW 370
73. 022 0046      MOVWF IOB      ;SEND NACT
74. 023 4000      RET            ;RETURN TO CALLING ROUTINE

```

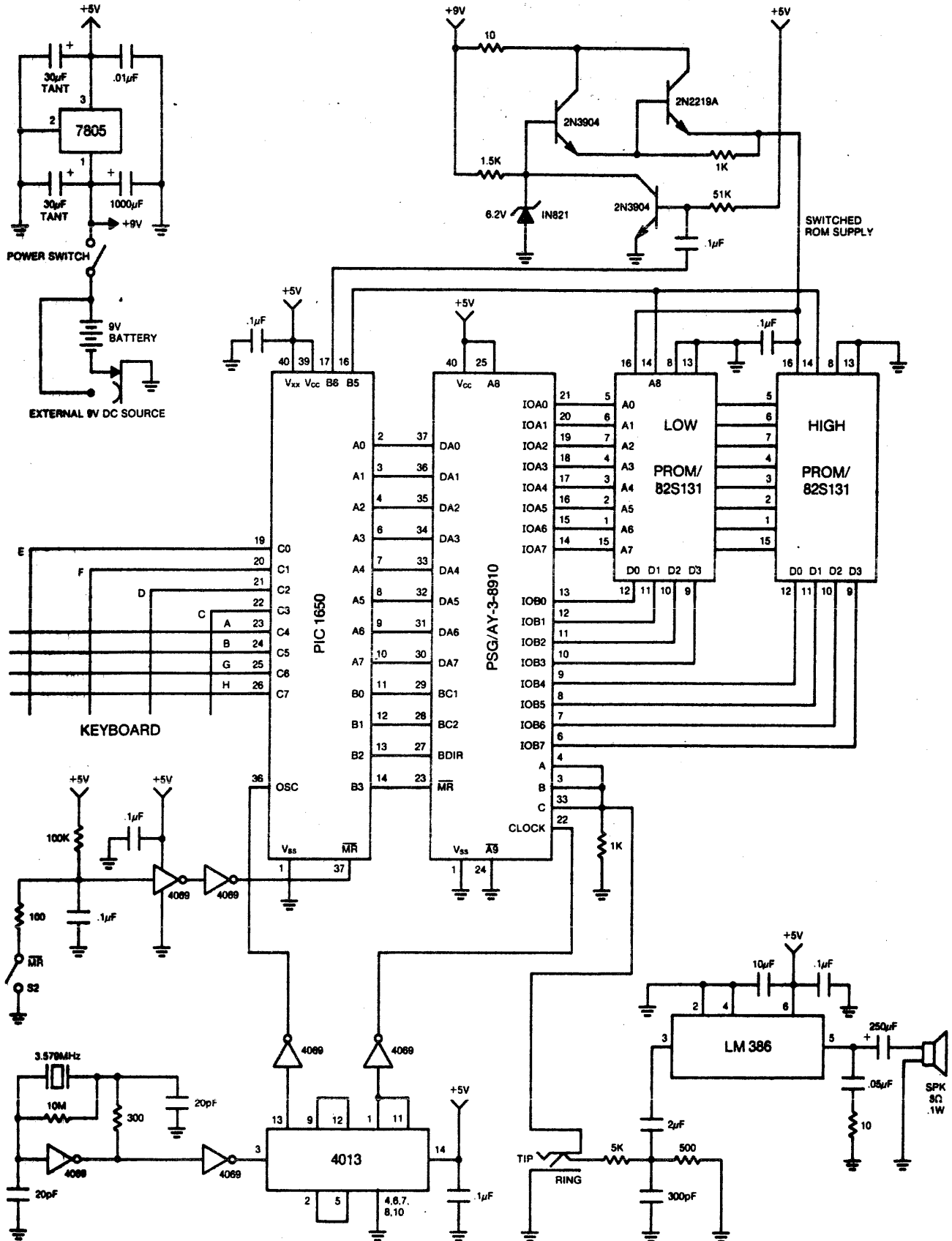
4.6.3 READ ROM ROUTINE

```

106.         ;ADDRESS OF ROM IN W AT ENTRANCE NEXROM
107.         ;ADDRESS OF ROM IN ROMAD AT ENTRANCE ROMRD
108.
109.         ;INCREMENTS ROMAD AFTER READ, IF ROM ADDRESS
110.         ;CROSSES 256 BORDER, MAKE UPPER BANK SELECT=1
111.
112.         ;USES 8910 REG 16 FOR ADDRESS
113.         ;8910 REG 17 FOR INPUT DATA
114. 046 1030 NEXROM MOVF  ROMAD,W
115. 047 0067 ROMRD  MOVWF DATA ;PUT ADDRESS
116. 050 6016      MOVLW 16     ;I/O A ADDRESS
117. 051 0066      MOVWF ADDRESS
118. 052 2306      BCF  IOB,6    ;TURN ON ROM
119. 053 4425      CALL WRITE    ;SEND TO IOA
120. 054 1266      INCF  ADDRESS ;TO IOB ADDRESS
121. 055 4401      CALL  READ    ;GET DATA
122. 056 2706      BSF  IOB,6    ;TURN OFF ROM
123. 057 1770      INCFSZ ROMAD  ;TO NEXT LOC
124. 060 4000      RET
125. 061 2646      BSF  IOB,5    ;SET HIGH SELECT
126. 062 4000      RET            ;RETURN TO CALLING ROUTINE

```

Fig. 19 PIC 1650/AY-3-8910 SYSTEM EXAMPLE



4.7 Interfacing to the CP1600/1610

As shown in Fig. 20, the wiring is direct between the AY-3-8910 and a CP1600/1610 microprocessor. The levels are compatible thus eliminating any need for level converters. Even the terminology between the IC's remains constant to provide simple-to-follow connections.

The CP1600/1610 acts as a controller in this configuration fetching data from ROM's contained elsewhere in the system. The CP1600/1610 also acts as the bus controller developing the necessary timing for the AY-3-8910.

4.7.1 WRITE DATA ROUTINE

The program necessary to write to a selected register is as follows:

```
MVI value, R0; move in value to be written  
MVO R0, Reg; write to register
```

The routine to load all registers with the same value is as follows:

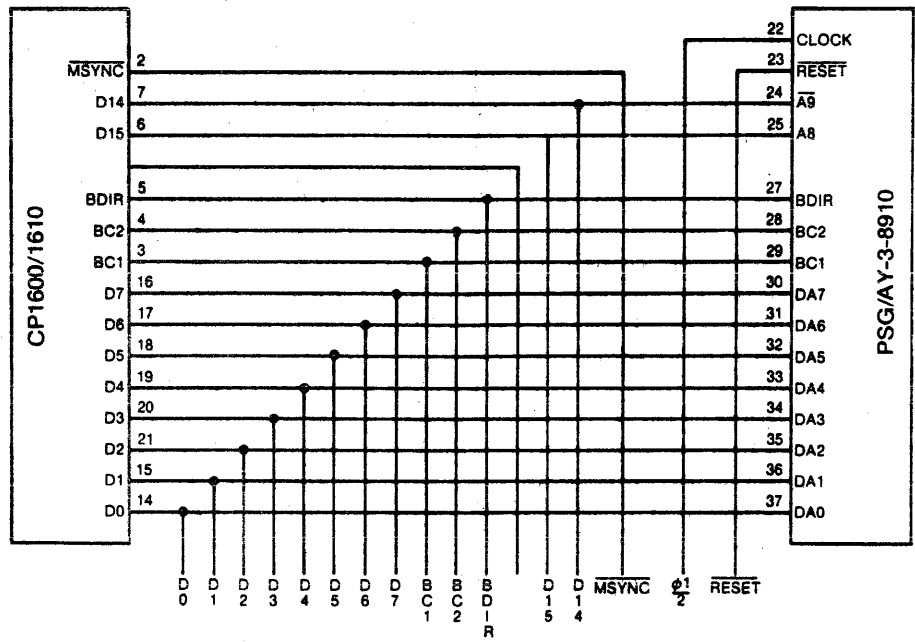
```
MVII Reg 0, R4  
CLRR R0  
Here MVO@ R0, R4  
CMPI Reg 0 + 17, R4  
BLT Here
```

4.7.2 READ DATA ROUTINE

The routine to read from a selected register is as follows:

```
MVI Reg, R0; get data from reg in R0  
MVO R0, value; store in memory
```

Fig. 20 CP1600/1610/AY-3-8910 INTERFACE



4.8 Interfacing to the M6800

An M6800 microprocessor can be interfaced with an AY-3-8910/8912 through the addition of an M6820 PIA chip. The I/O ports designated as PA0 to PA7 are used as the 8 bit bus lines and I/O ports PB0 to PB2 are used as the bus control lines. The software routines shown are used to control the latch address, write data, and read data functions for the AY-3-8910/8912.

4.8.1 LATCH ADDRESS ROUTINE

```
;AT ENTRY, B HAS ADDRESS VALUE  
;  
LATCH CLRA  
  STAA 8005 ;GET D DIR A  
  LDAA #FF  
  STAA 8004 ;OUTPUTS  
  LDAA #4  
  STAA 8005 ;GET PERIPHERAL A  
  STAB 8004 ;FORM ADDR  
  STAA 8006  
  CLRA  
  STAA 8006 ;LATCH ADDRESS  
  RTS ;RETURN
```

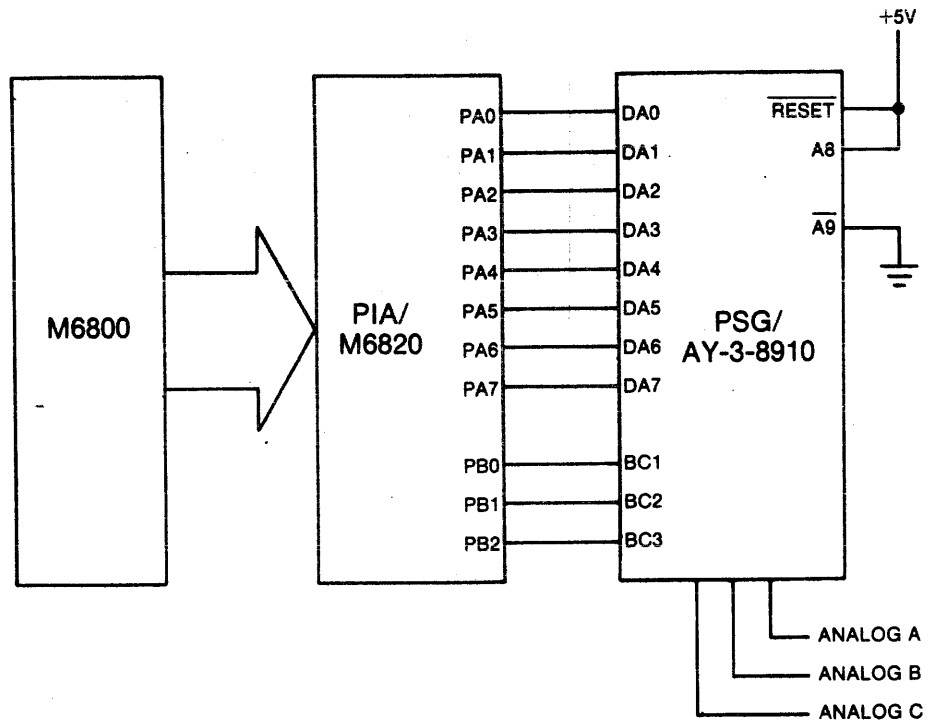
4.8.2 WRITE DATA ROUTINE

```
;AT ENTRY, B HAD DATA VALUE  
;  
WRITE STAB 8004 ;FORM DATA  
  LDAA #6 ;DWS  
  STAA 8006  
  CLRA  
  STAA 8006 ;WRITE DATA  
  RTS ;RETURN
```

4.8.3 READ DATA ROUTINE

```
;AFTER READ, B HAS READ DATA  
;  
READ STA A 8005 ;GET D DIR  
  STA A 8004 ;INPUTS  
  LDAA #4  
  STA A 8005 ;GET PERIPHERAL  
  DECA  
  STA A 8006 ;READ MODE  
  LDA B 8004 ;READ DATA  
  CLRA  
  STA A 8006 ;REMOVE READ MODE  
  RTS ;RETURN
```

Fig. 21 M6800/AY-3-8910 INTERFACE



4.9 Interfacing to the 8080 S100 Bus

The sample S100 bus design provides for reading and writing the PSG using only an 8080 "IN" or "OUT" instruction to the proper address. Another feature of the design is the provision for multiple PSG devices to be connected to a single bus. The system described is presently running two PSG's, one to each of two stereo channels.

As can be seen from the read and write routines in the illustrative program, the program overhead necessary to communicate with the PSG is minimal.

4.9.1 LATCH ADDRESS ROUTINE

```
PORTADDR EQU 80H ;ADDRESS TRANSFER PORT ADDRESS
PORTDATA EQU 81H ;DATA TRANSFER PORT ADDRESS
;
;THIS ROUTINE WILL TRANSFER THE CONTENTS OF
;8080 REGISTER C TO THE PSG ADDRESS REGISTER
PSGBAR    MOV    A,C ;GET C IN A FOR OUT
          OUT    PORTBAR ;SEND TO ADDRESS PORT
          RET
```

4.9.2 WRITE DATA ROUTINE

```
;
;ROUTINE TO WRITE THE CONTENTS OF 8080 REGISTER B
;TO THE PSG REGISTER SPECIFIED BY 8080 REGISTER C
;
PSGWRITE  CALL    PSGBAR ;GET ADDRESS LATCHED
          MOV    A,B ;GET VALUE IN A FOR TRANSFER
          OUT    PORTDATA ;PUT TO PSG REGISTER
          RET
```

4.9.3 READ DATA ROUTINE

```
;
;ROUTINE TO READ THE PSG REGISTER SPECIFIED
;BY THE 8080 REGISTER C AND RETURN THE DATA
;IN 8080 REGISTER B
;
PSGREAD   CALL    PSGBAR
          IN     PORTDATA ;GET REGISTER DATA
          MOV    B,A GET IN TRANSFER REGISTER
          RET
```

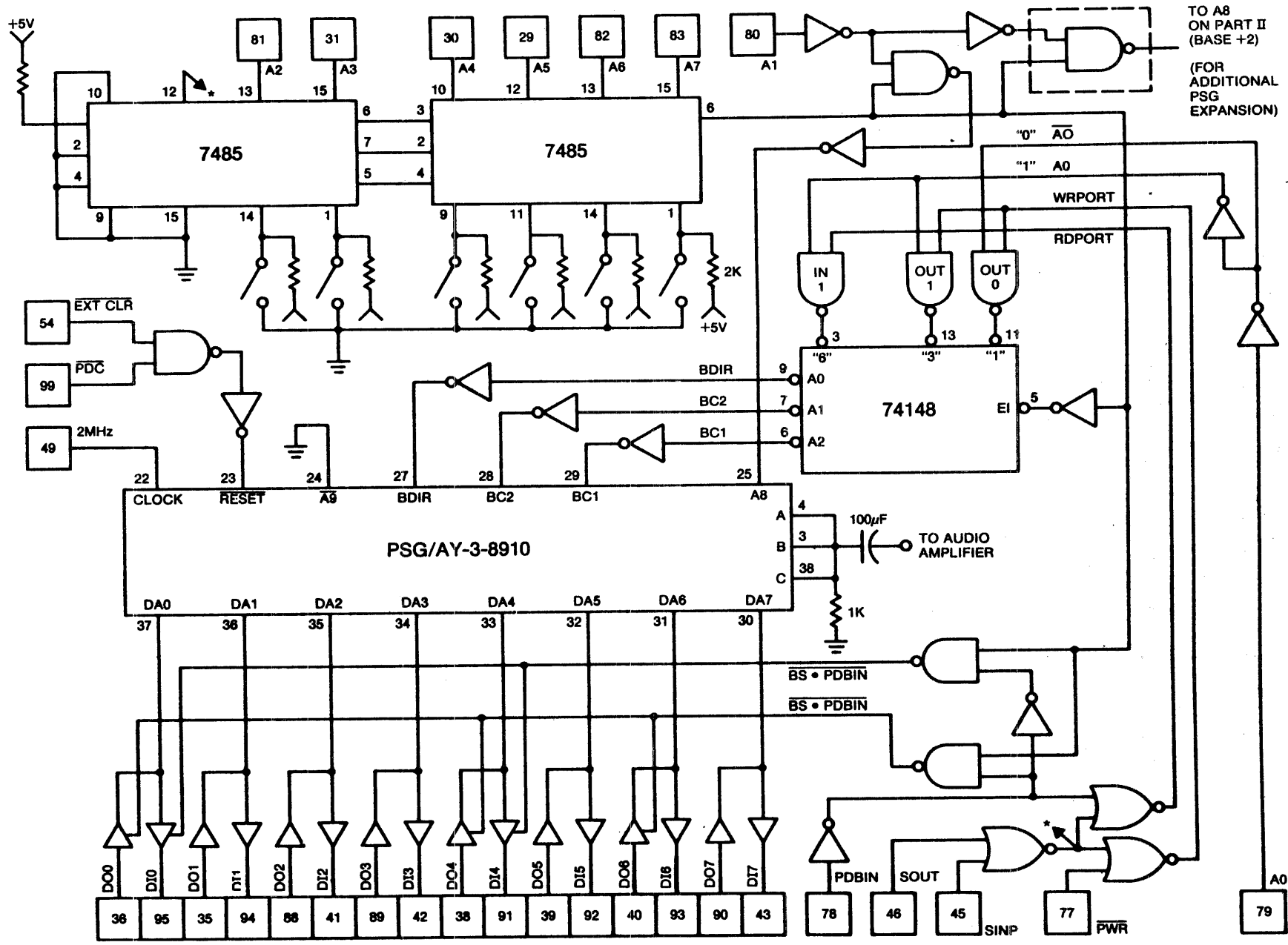


Fig. 22 8080 S100 BUS/AY-3-8910 INTERFACE

5 MUSIC GENERATION

The production of music involves the creation of series of frequencies which are pleasing to the human ear (setting critical evaluation aside). This involves essentially mathematical relationships, making the application ideal for digital devices. For example, the shifting up or down in octaves is a multiplication or division by a power of 2, which is a simple shift operation for most microprocessors.

Another factor in music generation is "communication". The composer must be able to convey his tune ideas so that a musician or group of musicians can reproduce the composer's ideas—often on widely differing instruments. This concept involves "tuning" the instruments to a standard set of frequencies and following a set rhythm pattern. The tuning frequency most widely used is based on the third octave note "A" of 440Hz, the "Equal Tempered Chromatic Scale".

Although it is easy to construct recognizable tunes using only one note at a time, the simultaneous sounding of more than one note to produce chords and counterpoint vastly increases the quality of the sound. This feature is easily achieved in the PSG since three channels are provided, each independently programmable.

5.1 Note Generation

Since notes are formed by sustaining a particular frequency for a preset period of time at a varying amplitude, the PSG performs this function with a series of simple register loads. The method used in many cases is to obtain register load values for first octave notes and to shift to the correct octave at playtime.

The chart in Fig. 23 lists a full 8 octaves of notes from a low of C1 (32.703Hz) to a high of B8 (7902.080Hz). Assuming an input clock frequency of 1.78977MHz (one half the standard "color" crystal frequency of 3.579545MHz), and applying the formulas of Section 3.1 for calculating Tone Period register load values, results in the register values shown. The nature of the PSG divider scheme produces a high degree of accuracy for low frequencies, less for high frequencies. This can be seen in the chart in the comparison of "ideal frequencies" and "actual frequencies", with the ideal frequencies being those of the Equal Tempered Chromatic Scale, and the actual frequencies being the "best fit" values from the formula calculation.

NOTE	OCTAVE	IDEAL FREQUENCY	ACTUAL FREQUENCY	12-BIT REGISTER VALUE IN OCTAL				NOTE	OCTAVE	IDEAL FREQUENCY	ACTUAL FREQUENCY	12-BIT REGISTER VALUE IN OCTAL			
C	1	32.703	32.698	6	5	3	5	C	6	523.248	522.714	0	3	2	6
C#	1	34.648	34.653	6	2	3	4	C#	5	554.368	553.766	0	3	1	2
D	1	36.708	36.712	5	7	4	7	D	5	587.328	588.741	0	2	7	6
D#	1	38.891	38.895	5	4	7	4	D#	5	622.256	621.449	0	2	6	4
E	1	41.203	41.201	5	2	3	3	E	5	659.248	658.005	0	2	5	2
F	1	43.654	43.662	5	0	0	2	F	5	698.464	699.130	0	2	4	0
F#	1	46.249	46.243	4	5	6	3	F#	5	739.984	740.800	0	2	2	7
G	1	48.999	48.997	4	3	5	3	G	5	783.984	782.243	0	2	1	7
G#	1	51.913	51.908	4	1	5	3	G#	5	830.608	828.598	0	2	0	7
A	1	55.000	54.995	3	7	6	2	A	5	880.000	880.794	0	1	7	7
A#	1	58.270	58.261	3	6	0	0	A#	5	932.320	932.173	0	1	7	0
B	1	61.735	61.733	3	4	2	4	B	5	987.760	989.918	0	1	6	1
C	2	65.406	65.416	3	2	5	6	C	6	1046.496	1045.428	0	1	5	3
C#	2	69.296	69.307	3	1	1	6	C#	6	1108.736	1107.532	0	1	4	5
D	2	73.416	73.399	2	7	6	4	D	6	1174.656	1177.482	0	1	3	7
D#	2	77.782	77.789	2	6	3	6	D#	6	1244.512	1242.898	0	1	3	2
E	2	82.406	82.432	2	5	1	5	E	6	1318.496	1316.009	0	1	2	5
F	2	87.308	87.323	2	4	0	1	F	6	1396.928	1398.260	0	1	2	0
F#	2	92.498	92.523	2	2	7	1	F#	6	1479.968	1471.852	0	1	1	4
G	2	97.998	98.037	2	1	6	5	G	6	1567.968	1575.504	0	1	0	7
G#	2	103.826	103.863	2	0	6	5	G#	6	1661.216	1669.564	0	1	0	3
A	2	110.000	109.991	1	7	7	1	A	6	1760.000	1747.825	0	1	0	0
A#	2	116.540	116.522	1	7	0	0	A#	6	1864.640	1864.346	0	0	7	4
B	2	123.470	123.467	1	6	1	2	B	6	1975.520	1962.470	0	0	7	1
C	3	130.812	130.831	1	5	2	7	C	7	2092.992	2110.581	0	0	6	5
C#	3	138.592	138.613	1	4	4	7	C#	7	2217.472	2237.216	0	0	6	2
D	3	146.832	146.799	1	3	7	2	D	7	2349.312	2330.433	0	0	6	0
D#	3	155.564	155.578	1	3	1	7	D#	7	2489.024	2485.795	0	0	5	5
E	3	164.812	164.743	1	2	4	7	E	7	2636.992	2663.352	0	0	5	2
F	3	174.616	174.510	1	2	0	1	F	7	2793.856	2796.520	0	0	5	0
F#	3	184.996	184.894	1	1	3	5	F#	7	2959.936	2943.705	0	0	4	6
G	3	195.996	195.903	1	0	7	3	G	7	3135.936	3107.244	0	0	4	4
G#	3	207.652	207.534	1	0	3	3	G#	7	3322.432	3290.023	0	0	4	2
A	3	220.000	220.198	0	7	7	4	A	7	3520.000	3495.649	0	0	4	0
A#	3	233.080	233.043	0	7	4	0	A#	7	3729.280	3728.693	0	0	3	6
B	3	246.940	246.933	0	7	0	5	B	7	3951.040	3995.028	0	0	3	4
C	4	261.624	261.357	0	6	5	4	C	8	4185.984	4142.992	0	0	3	3
C#	4	277.184	276.883	0	6	2	4	C#	8	4434.944	4474.431	0	0	3	1
D	4	293.664	293.598	0	5	7	5	D	8	4698.624	4660.866	0	0	3	0
D#	4	311.128	310.724	0	5	5	0	D#	8	4978.048	5084.581	0	0	2	6
E	4	329.624	329.973	0	5	2	3	E	8	5273.984	5326.704	0	0	2	5
F	4	349.232	349.565	0	5	0	0	F	8	5587.712	5593.039	0	0	2	4
F#	4	369.992	370.400	0	4	5	6	F#	8	5919.872	5887.410	0	0	2	3
G	4	391.992	392.494	0	4	3	5	G	8	6271.872	6214.488	0	0	2	2
G#	4	415.304	415.839	0	4	1	5	G#	8	6644.864	6580.046	0	0	2	1
A	4	440.000	440.397	0	3	7	6	A	8	7040.000	6991.299	0	0	2	0
A#	4	466.160	466.087	0	3	6	0	A#	8	7458.560	7457.385	0	0	1	7
B	4	493.880	494.959	0	3	4	2	B	8	7902.080	7990.056	0	0	1	6

Fig. 23 EQUAL TEMPERED CHROMATIC SCALE ($f_{\text{CLOCK}}=1.78977\text{MHz}$)

5.2 Tune Entry/ Playback

One of the methods of entering a composition into a computer memory would be to utilize a keyboard to pass number and alphabetic information concerning the composer's wishes. An alternate method would be to scan a positional series of switches (like a piano keyboard) to determine note, volume and duration data.

Since flexibility in tune entry is desired, it is important to allow the composer to specify certain constants of entry such as octave, pitch or tempo, and have these entries normalized to a known value.

5.3 Tune Variations

One of the significant features of a microcomputer based music player is the ability to modify the tune once it has been recorded. Among the simpler variations are:

5.3.1 OCTAVE SHIFT

If an octave constant is added to the octave of the recorded note prior to storing the value in the PSG register, dynamic pitch changes can be obtained. The programming effect would be to shift one bit left for each lower octave and one bit right for each higher octave. For example, the effect will be that a tune written to play on a piano will sound like bells if a multiple octave up modification is performed.

5.3.2 KEY

One measure of the virtuosity of a musician is his ability to modify the "key" or suboctave shift of a composition. The logical description of key transposition is to shift each note up or down by a predetermined number of notes from the original. For example, a piece written in C and played in C# would have all C notes shifted to C#, C# shifted to D, etc. (Note that the case must be considered where B of one octave is shifted to C of the next higher octave.) All of these operations require that the one of twelve note identification must be retained in the recorded representation.

5.3.3 TEMPO

The duration of each recorded note is best expressed in terms of "ticks" of an overall "tempo clock". At playtime, the total duration can be obtained by programatically multiplying the individual note to "slow down" or "speed up" the tune without changing the crucial time relationship between the notes. This can be accomplished by imbedding the note timing loops within the tempo timing loops for simple operation.

5.3.4 CHORDS

There are certain combinations of notes which when played simultaneously produce pleasant combinations. These "chords" can be easily formed from a base note by performing octave and key changes on two notes, which are played with the main note. These relationships are illustrated in Fig. 24, which lists the various note constants which will produce musical chords. A chord with a particular quality may be formed by playing its root, a 3rd Minor or Major, and other notes from the chord chart. For example, a C Major chord is formed from C(+2), E(+2), and G(+2).

Fig. 24 CHORD SELECTION CHART

Chord Selection	Root	3rd Minor	3rd Major	4th	5th	6th	7th
C	C (+2)	D# (+2)	E (+2)	F (+2)	G (+2)	A (+2)	A# (+2)
C#	C# (+2)	E (+2)	F (+2)	F# (+2)	G# (+2)	A# (+2)	B (+2)
D	D (+2)	F (+2)	F# (+2)	G (+2)	A (+2)	B (+2)	C (+1)
D#	D# (+2)	F# (+2)	G (+2)	G# (+2)	A# (+2)	C (+1)	C# (+1)
E	E (+2)	G (+2)	G# (+2)	A (+2)	B (+2)	C# (+1)	D (+1)
F	F (+2)	G# (+2)	A (+2)	A# (+2)	C (+1)	D (+1)	D# (+1)
F#	F# (+4)	A (+4)	A# (+4)	B (+4)	C# (+2)	D# (+2)	E (+2)
G	G (+4)	A# (+4)	B (+4)	C (+2)	D (+2)	E (+2)	F (+2)
G#	G# (+4)	B (+4)	C (+2)	C# (+2)	D# (+2)	F (+2)	F# (+2)
A	A (+4)	C (+2)	C# (+2)	D (+2)	E (+2)	F# (+2)	G (+2)
A#	A# (+4)	C# (+2)	D (+2)	D# (+2)	F (+2)	G (+2)	G# (+2)
B	B (+4)	D (+2)	D# (+2)	E (+2)	F# (+2)	G# (+2)	A (+2)

5.4 Sound Variation

5.4.1 RELATIVE CHANNEL VOLUME

The independently programmable amplitude control for each channel allows up to 16 levels if using the processor controlled amplitude mode (bit 4 of registers 10, 11 or 12=0). In the case of a decaying or steady note, when a note is played or "fired", a frequency may be set up in the coarse and fine tune registers and then an amplitude value placed in the respective register 10, 11 or 12. The value which is placed to play the tune can be an independent variable, allowing channels to play their respective melody lines with varying force.

5.4.2 DECAY

The main difference between a "piano" sound and an "organ" sound is the speed with which the note loses volume. If all of the notes can be decayed at a uniform rate, the automatic envelope generator can be set to produce a decaying waveform. Each of the three channels can have the same decay constant but differing playing times to simulate the same instrument with differing note-strike times.

5.4.3 OTHER EFFECTS

The addition of variable noise to any or all of the channels can produce modification effects such "breathing" with a wind instrument. Or noise can be used alone to produce a drum rhythm. The fact that the noise dominant frequencies are variable allows "synthesizer" type effects with simple processor interaction.

Other pleasing effects include vibrato and tremolo, the cyclical variation of the frequency and volume. Because an intelligent microprocessor is controlling the effect, they can be all keyed to the tune itself or to other external stimuli.

5.5 Applications

While many applications of the PSG in music generation are apparent, for instance in the area of toys and games, other applications are possible even in the area of high accuracy sophisticated musical instruments such as high-end electronic organs. With tone frequencies generated from another source to meet the exacting requirements of organ operation, the PSG can be used as a complex envelope generator. The PSG is also effective for generating bass notes and rhythms with percussion instruments, taking advantage of the PSG's high accuracy in producing low frequency notes. The following paragraphs detail examples of these applications.

5.5.1 ORGAN ENVELOPE GENERATION

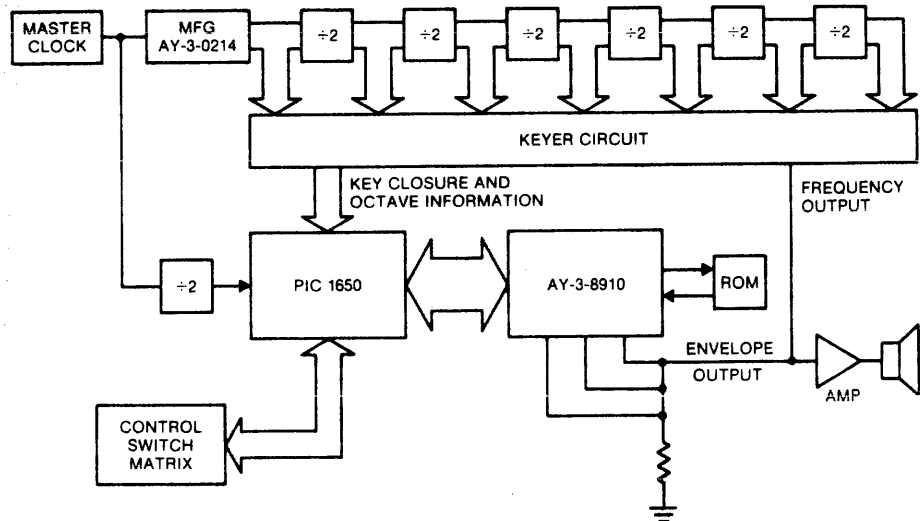
The envelope generation diagram shown in Fig. 25 illustrates how an AY-3-8910 can be configured to produce envelopes for organ voicing. All functions are controlled by a microcomputer.

The basis of this system consists of a master frequency generator with a string of dividers. This produces all frequencies for the keyboard. The microcomputer and the AY-3-8910 are actually used to replace the usual components of voicing filters that would ordinarily be used in an electronic organ.

The microcomputer shown is a GI PIC 1650 controlled by inputs from the keyboard keyer circuit and a control switch matrix. The keyer inputs octave and key closure information to develop the envelope amplitude and duration for the note to be played. The control switch matrix can be used to control sustain and add other special effects. The ROM shown connected to the AY-3-8910 is optional depending on the amount of data necessary for the microcomputer.

The system shown here may also consist of multiple AY-3-8910's, all controlled by a single microcomputer. It represents an economical solution to developing voicing control with a minimum of components.

Fig. 25 ORGAN ENVELOPE GENERATION



5.5 Applications (cont.)

5.5.2 ORGAN RHYTHM GENERATION

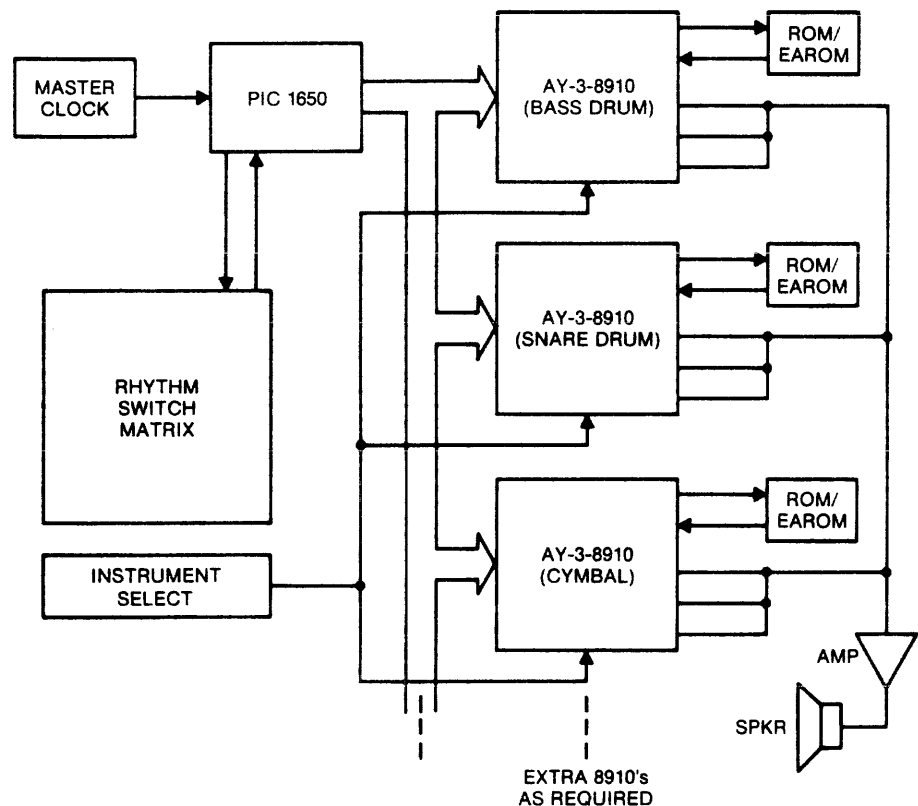
The rhythm generation diagram (Fig. 26) illustrates a simplified version of how a microcomputer can be implemented with the AY-3-8910 to provide a percussion instrument section for an electronic organ.

The microcomputer used in this case could be a GI PIC 1650 which can be internally programmed to drive a series of AY-3-8910's, all hardwired to an I/O port of the PIC. Each AY-3-8910 provides a separate output envelope and frequency of the instrument it is to synthesize.

The Rhythm Switch Matrix is used to select any preprogrammed rhythm pattern and tempo from the PIC. The Instrument Select switches allow manual in/out selection of the 8910's via the A8 and A9 address lines providing additional instrument sound variations. These switches are intended to be user-selected and mounted in a convenient position on the instrument.

In addition, optional ROMs could be added to the PSG I/O ports, saving microcomputer ports, to provide extra rhythm length or number of patterns. These ROMs could also be replaced by EAROMs to provide user rhythm programming from a modified Rhythm Switch Matrix. The programmable rhythm feature could be used to add new or original user rhythms to update the instrument.

Fig. 26 ORGAN RHYTHM GENERATION



6 SOUND EFFECTS GENERATION

One of the main uses of the PSG is to produce non-musical sound effects to accompany visual action or as a feature in itself. The following sections outline techniques and provide actual examples of some popular effects. All examples are based on a 1.78977MHz PSG clock.

6.1 Tone Only Effects

Many effects are possible using only the tone generation capability of the PSG without adding noise and without using the PSG's envelope generation capability. Examples of this type of effect would include telephone tone frequencies (two distinct frequencies produced simultaneously) or the European Siren effect listed in Fig. 27 (two distinct frequencies sequentially produced).

Fig. 27 EUROPEAN SIREN SOUND EFFECT CHART

Register #	Octal Load Value	Explanation	
Any not specified	000	—	
R0	376 }	Set Channel A Tone period to 2.27ms (440Hz).	
R1	000 }		
R7	076		Enable Tone only on Channel A only.
R10	017		Select maximum amplitude on Channel A.
		<i>(Wait approximately 350ms before continuing)</i>	
R0	126 }	Set Channel A Tone period to 5.346ms (187Hz).	
R1	001 }		
		<i>(Wait approximately 350ms before continuing)</i>	
R10	000	Turn off Channel A to end sound effect.	

6.2 Noise Only Effects

Some of the more commonly required sounds require only the use of noise and the envelope generator (or processor control of channel envelope if other channels are using the envelope generator).

Examples of this, which can be seen in Figs. 28 and 29, are gunshot and explosion. In both cases pure noise is used with a decaying envelope. In the examples shown the only changes are in the length of the envelope as modified by the coarse tune register and in the noise period. Note that a significantly lower explosion can be obtained by using all three channels operating with the same parameters.

Fig. 28 GUNSHOT SOUND EFFECT CHART

Register #	Octal Load Value	Explanation
Any not specified	000	—
R6	017	Set Noise period to mid-value.
R7	007	Enable Noise only on Channels A,B,C.
R10	020	Select full amplitude range under direct control of Envelope Generator.
R11	020	
R12	020	
R14	020	Set Envelope period to 0.586 seconds.
R15	000	Select Envelope "decay", one cycle only.

Fig. 29 EXPLOSION SOUND EFFECT CHART

Register #	Octal Load Value	Explanation
Any not specified	000	—
R6	000	Set Noise period to max. value.
R7	007	Enable Noise only, on Channels A,B,C.
R10	020	Select full amplitude range under direct control of Envelope Generator.
R11	020	
R12	020	
R14	070	Set Envelope period to 2.05 seconds.
R15	000	Select Envelope "decay", one cycle only.

6.3 Frequency Sweep Effects

The Laser, Whistling Bomb, Wolf Whistle, and Race Car sounds in Figs. 30 thru 33 all utilize frequency sweeping effects. In all cases they involve the increasing or decreasing of the values in the tone period registers with variable start, end, and time between frequency changes. For example, the sweep speed of the Laser is much more rapid than the high gear accelerate in the race car, yet both use the same computer routine with differing parameters.

Other easily achievable results include "doppler" and noise sweep effects. The sweeping of the noise clocking register (R6) produces a "doppler" effect which seems well suited for "space war" type games.

Fig. 30 LASER SOUND EFFECT CHART

Register #	Octal Load Value	Explanation
Any not specified	000	—
R7	076	Enable Tone only on Channel A only.
R10	017	Select maximum amplitude on Channel A.
R0	060 (start)	Sweep effect for Channel A Tone period via a processor loop with approximately 3ms wait time between each step from 060 to 160 (0.429ms/2330Hz to 1.0ms/1000Hz).
R0	160 (end)	
R10	000	Turn off Channel A to end sound effect.

Fig. 31 WHISTLING BOMB SOUND EFFECT CHART

Register #	Octal Load Value	Explanation
Any not specified	000	—
R7	076	Enable Tone only on Channel A only.
R10	017	Select maximum amplitude on Channel A.
R0	060 (start)	Sweep effect for Channel A Tone period via a processor loop with approximately 25ms wait time between each step from 060 to 300 (0.429ms/2330Hz to 1.72ms/582Hz).
R0	300 (end)	

After above loop is completed, follow with sequence in Fig. 28.

6.4 Multi-Channel Effects

Because of the independent architecture of the PSG, many rather complex effects are possible without burdening the processor. For example, the Wolf Whistle effect in Fig. 32 shows two channels in use to add constant breath hissing noise to the three concentrated frequency sweeps of the whistle. Once the noise is put on the channel, the processor only need be concerned with the frequency sweep operation.

Fig. 32 WOLF WHISTLE SOUND EFFECT CHART

Register #	Octal Load Value	Explanation
Any not specified	000	—
R6	001	Set Noise period to minimum value.
R7	056	Enable Tone on Channel A, Noise on Channel B.
R10	017	Select maximum amplitude on Channel A.
R11	011	Select lower amplitude on Channel B.
R0	100 (start)	Sweep effect for Channel A Tone period via a processor loop with approximately 12ms wait time between each step from 100 to 040 (0.572ms/1748Hz to 0.286ms/3496Hz). (Wait approximately 150ms before continuing)
R0	040 (end)	
R0	100 (start)	A processor loop with approximately 25ms wait time between each step from 100 to 060 (0.572ms/1748Hz to 0.429ms/2331Hz).
R0	060 (end)	
R0	060 (start)	A processor loop with approximately 6ms wait time between each step from 060 to 150 (0.429ms/2331Hz to 0.930ms/1075Hz).
R0	150 (end)	
R10	000	Turn off Channels A and B to end effect.
R11	000	

Fig. 33 RACE CAR SOUND EFFECT CHART

Register #	Octal Load Value	Explanation
Any not specified	000	—
R3	017	Set Channel B Tone period to 34.33ms (29Hz).
R7	074	Enable Tones only on Channels A and B.
R10	017	Select maximum amplitude on Channel A.
R11	012	Select lower amplitude on Channel B.
*R1/R0	013/000 (start)	Sweep effect for Channel A Tone period via a processor loop with approximately 3ms wait time between each step from 013/000 to 004/000 (25.17ms/39.7Hz to 9.15ms/109.3Hz).
*R1/R0	004/000 (end)	
R1/R0	011/000 (start)	A processor loop with approximately 3ms wait time between each step from 011/000 to 003/000 (20.6ms/48.5Hz to 6.87ms/145.6Hz).
R1/R0	003/000 (end)	
R1/R0	006/000 (start)	A processor loop with approximately 6ms wait time between each step from 006/000 to 001/000 (13.73ms/72.8Hz to 2.29ms/436.7Hz).
R1/R0	001/000 (end)	
R10	000	Turn off Channels A and B to end effect.
R11	000	

* Decrement R1/R0 as a whole number, e.g. start at 013/000, then 012/377, then 012/376, etc.

7 ELECTRICAL SPECIFICATIONS

7.1 Maximum Ratings

Storage Temperature -55°C to $+150^{\circ}\text{C}$
 Operating Temperature 0°C to $+40^{\circ}\text{C}$
 V_{CC} and all other input and output voltages with respect to V_{SS} -0.3V to $+8.0\text{V}$

Exceeding these ratings could cause permanent damage to these devices. Functional operation at these conditions is not implied—operating conditions are specified below.

7.2 Standard Conditions

$V_{CC} = +5\text{V} \pm 5\%$
 $V_{SS} = \text{GND}$
 Operating temperature: 0°C to $+40^{\circ}\text{C}$

7.3 DC Characteristics

Characteristic	Sym	Min.	Typ.*	Max.	Units	Conditions
All Inputs						
Logic "0"	V_{IL}	0	—	0.6	V	
Logic "1"	V_{IH}	2.4	—	V_{CC}	V	
All Outputs (except Analog Channel Outputs)						
Logic "0"	V_{OL}	0	—	0.5	V	$I_{OL} = 1.6\text{ mA}$, 20pF
Logic "1"	V_{OH}	2.4	—	V_{CC}	V	$I_{OH} = 100\mu\text{A}$, 20pF
Analog Channel Outputs	V_o	0	—	60	dB	Test circuit: Fig. 34
Power Supply Current	I_{CC}	—	45	75	mA	

*Typical values are at $+25^{\circ}\text{C}$ and nominal voltages.

Fig. 34 ANALOG CHANNEL OUTPUT TEST CIRCUIT

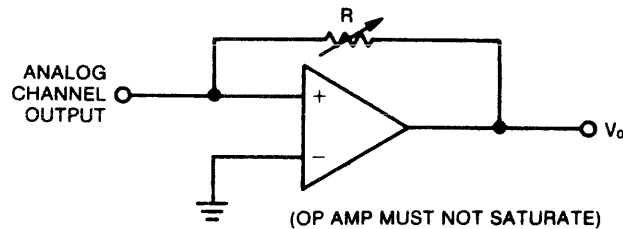
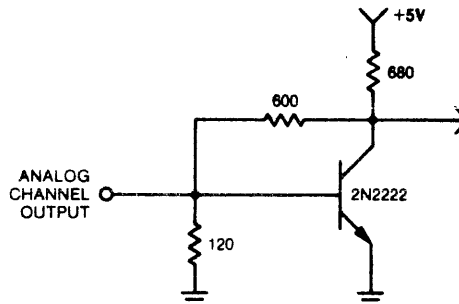


Fig. 35 CURRENT TO VOLTAGE CONVERTER



7.4 AC Characteristics

Characteristic	Sym	Min.	Typ.*	Max.	Units	Conditions
Clock input						
Frequency	f_c	1.0	—	2.0	MHz	} Fig. 36
Rise time	t_r	—	—	50	ns	
Fall time	t_f	—	—	50	ns	
Duty Cycle	—	25	50	75	%	
Bus Signals (BDIR, BC2, BC1)						
Associative Delay Time	t_{BD}	—	—	50	ns	} Fig. 37
Reset						
Reset Pulse Width	t_{RW}	500	—	—	ns	} Fig. 38
Reset to Bus Control Delay Time	t_{RB}	100	—	—	ns	
A9, A8, DA7--DA0 (Address Mode)						
Address Setup Time	t_{AS}	400	—	—	ns	} Fig. 39
Address Hold Time	t_{AH}	100	—	—	ns	
DA7--DA0 (Write Mode)						
Write Data Pulse Width	t_{DW}	500	—	10,000	ns	} Fig. 40
Write Data Setup Time	t_{DS}	50	—	—	ns	
Write Data Hold Time	t_{DH}	100	—	—	ns	
DA7--DA0 (Read Mode)						
Read Data Access Time	t_{DA}	—	250	500	ns	} Fig. 40
DA7--DA0 (Inactive Mode)						
Tristate Delay Time	t_{TS}	—	100	200	ns	

* Typical values are at 25°C and nominal voltages.

Fig. 36 CLOCK AND BUS SIGNAL TIMING

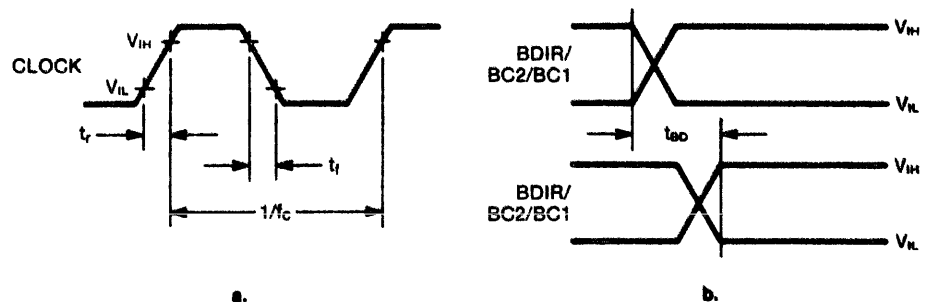


Fig. 37 RESET TIMING

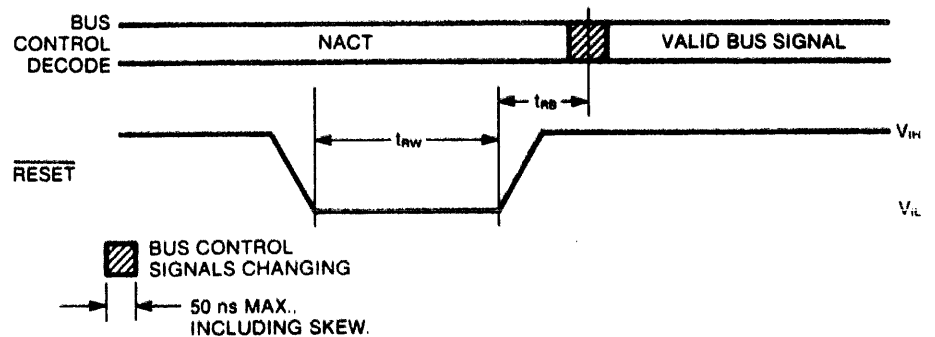


Fig. 38 LATCH ADDRESS TIMING

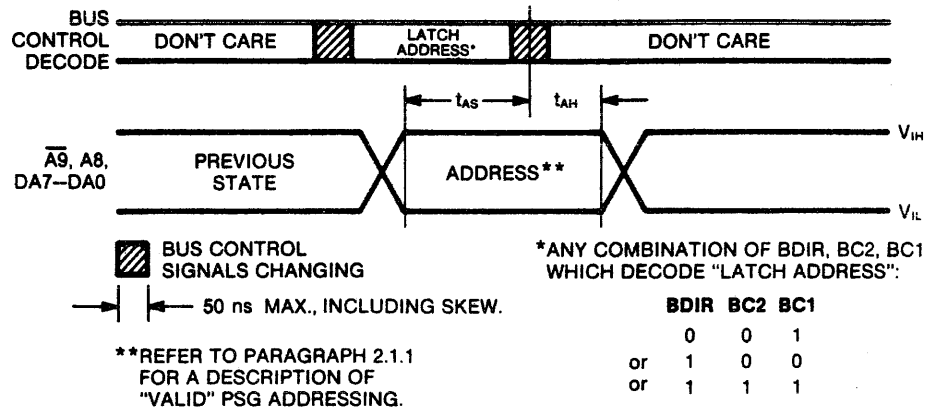


Fig. 39 WRITE DATA TIMING

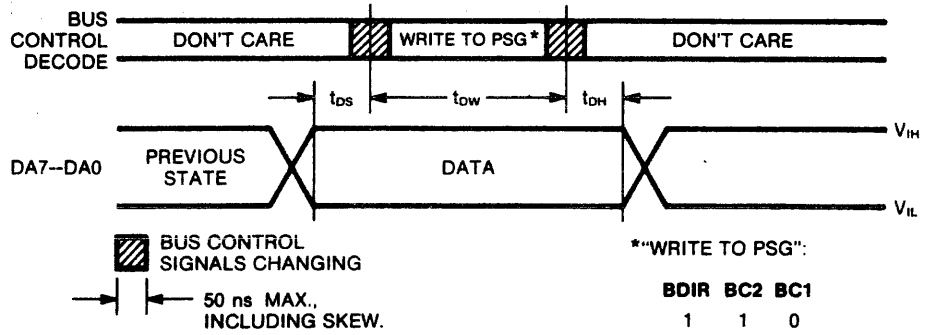
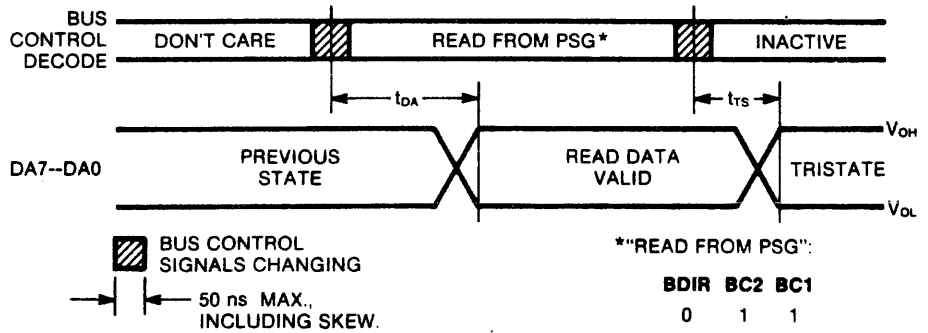
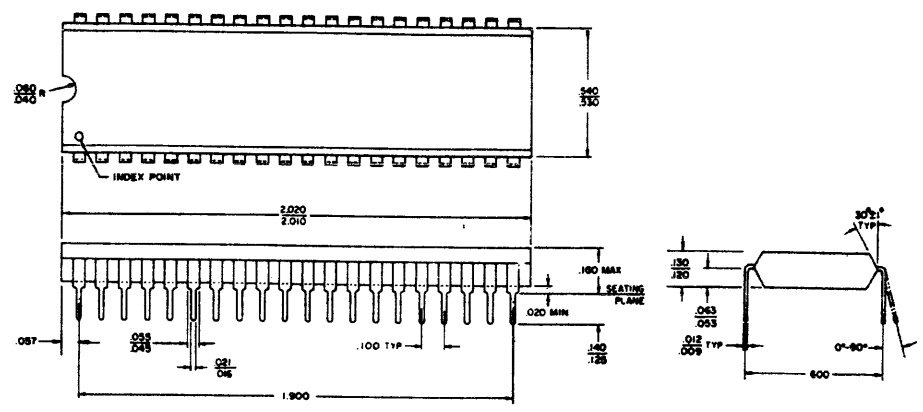


Fig. 40 READ DATA TIMING

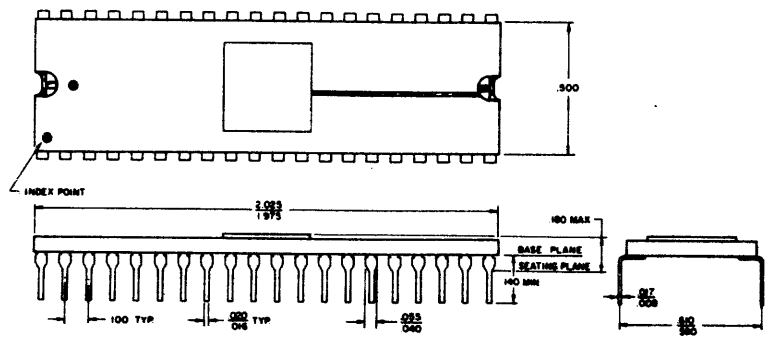


7.5 Package Outlines

Fig. 41 40 LEAD DUAL IN LINE PACKAGES (for AY-3-8910)

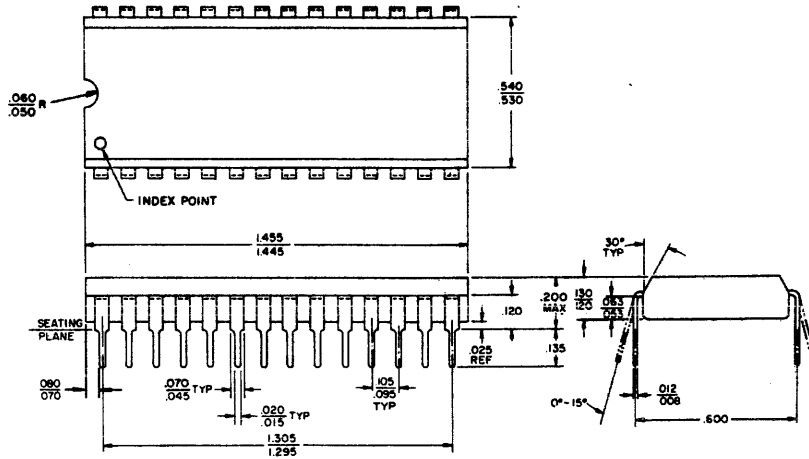


PLASTIC

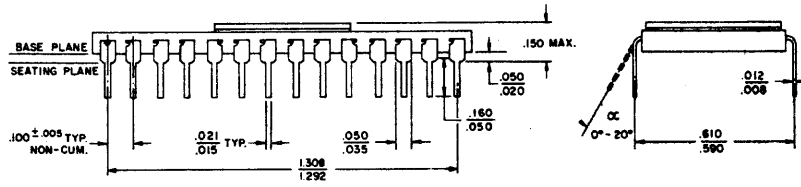
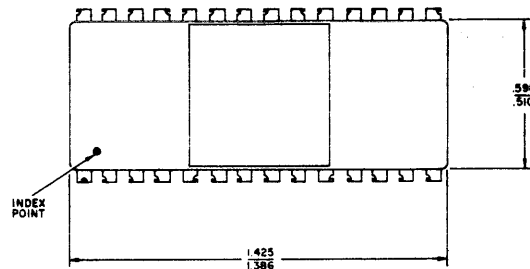


CERAMIC

Fig. 42 28 LEAD DUAL IN LINE PACKAGES (for AY-3-8912)



PLASTIC



CERAMIC

NOTES

Appendix D

ADC0808/ADC0809 Data Sheet
(ADC)

The following material is copyrighted by National Semiconductor Corporation. It is reprinted here with the permission of National Semiconductor. This data sheet may not be reproduced for any purpose in whole or part without the expressed written consent of National Semiconductor.

ADC0808, ADC0809 8-Bit μ P Compatible A/D Converters With 8-Channel Multiplexer

General Description

The ADC0808, ADC0809 data acquisition component is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8-bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8-single-ended analog signals.

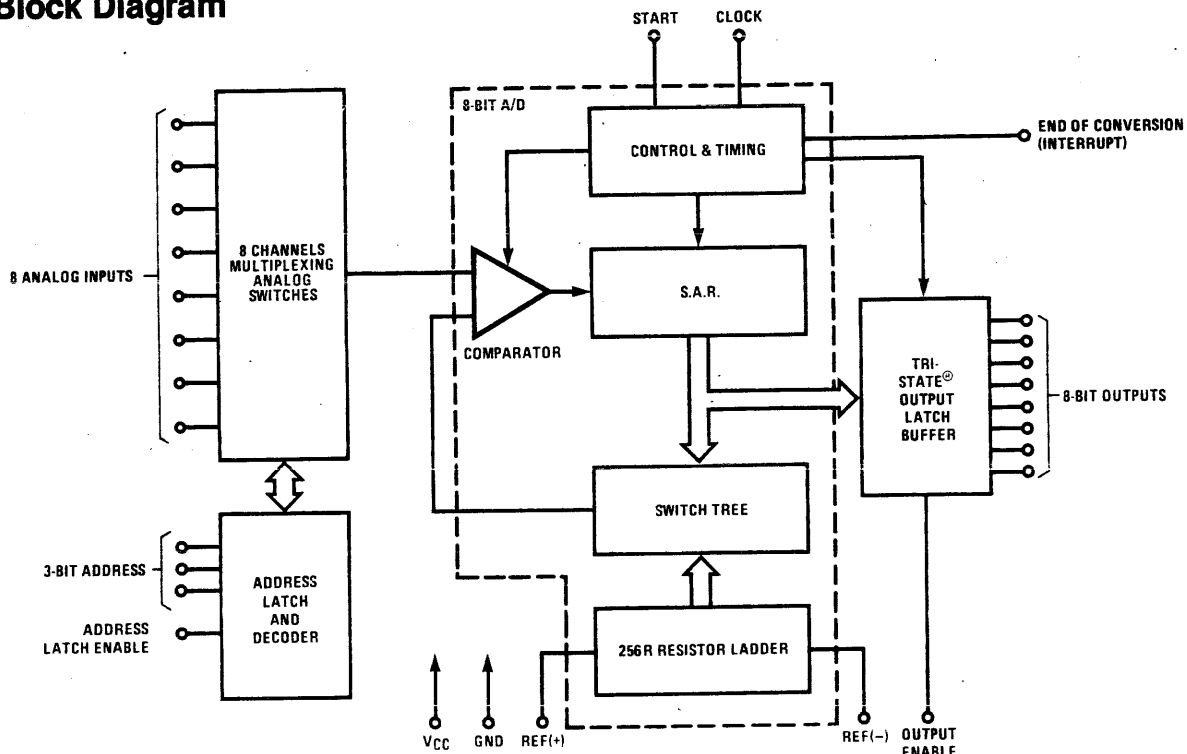
The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL TRI-STATE[®] outputs.

The design of the ADC0808, ADC0809 has been optimized by incorporating the most desirable aspects of several A/D conversion techniques. The ADC0808, ADC0809 offers high speed, high accuracy, minimal temperature dependence, excellent long-term accuracy and repeatability, and consumes minimal power. These features make this device ideally suited to applications from process and machine control to consumer and automotive applications. For 16-channel multiplexer with common output (sample/hold port) see ADC0816 data sheet.

Features

- Resolution — 8-bits
- Total unadjusted error — $\pm 1/2$ LSB and ± 1 LSB
- No missing codes
- Conversion time — 100 μ s
- Single supply — 5 V_{DC}
- Operates ratiometrically or with 5 V_{DC} or analog span adjusted voltage reference
- 8-channel multiplexer with latched control logic
- Easy interface to all microprocessors, or operates "stand alone"
- Outputs meet T²L voltage level specifications
- 0V to 5V analog input voltage range with single 5V supply
- No zero or full-scale adjust required
- Standard hermetic or molded 28-pin DIP package
- Temperature range -40°C to $+85^{\circ}\text{C}$ or -55°C to $+125^{\circ}\text{C}$
- Low power consumption — 15 mW
- Latched TRI-STATE[®] output

Block Diagram



Absolute Maximum Ratings (Notes 1 and 2)

Supply Voltage (V_{CC}) (Note 3)	6.5V
Voltage at Any Pin Except Control Inputs	-0.3V to ($V_{CC} + 0.3V$)
Voltage at Control Inputs (START, OE, CLOCK, ALE, ADD A, ADD B, ADD C)	-0.3V to +15V
Storage Temperature Range	-65°C to +150°C
Package Dissipation at $T_A = 25^\circ\text{C}$	875 mW
Lead Temperature (Soldering, 10 seconds)	300°C

Operating Ratings (Notes 1 and 2)

Temperature Range (Note 1) ADC0808CJ	$T_{MIN} \leq T_A \leq T_{MAX}$ -55°C $\leq T_A \leq$ +125°C
ADC0808CCJ, ADC0808CCN, ADC0809CCN	-40°C $\leq T_A \leq$ +85°C
Range of V_{CC} (Note 1)	4.5 V_{DC} to 6.0 V_{DC}

Electrical Characteristics

Converter Specifications: $V_{CC} = 5 V_{DC} = V_{REF(+)}$, $V_{REF(-)} = GND$, $T_{MIN} \leq T_A \leq T_{MAX}$ and $f_{CLK} = 640 \text{ kHz}$ unless otherwise stated.

Parameter	Conditions	Min	Typ	Max	Units
ADC0808 Total Unadjusted Error (Note 5)	25°C T_{MIN} to T_{MAX}			$\pm 1/2$ $\pm 3/4$	LSB LSB
ADC0809 Total Unadjusted Error (Note 5)	0°C to 70°C T_{MIN} to T_{MAX}			± 1 $\pm 1 1/4$	LSB LSB
Input Resistance	From Ref(+) to Ref(-)	1.0	2.5		k Ω
Analog Input Voltage Range	(Note 4) V(+) or V(-)	GND-0.10		$V_{CC} + 0.10$	V_{DC}
$V_{REF(+)}$ Voltage, Top of Ladder	Measured at Ref(+)		V_{CC}	$V_{CC} + 0.1$	V
$\frac{V_{REF(+)} + V_{REF(-)}}{2}$ Voltage, Center of Ladder		$V_{CC}/2 - 0.1$	$V_{CC}/2$	$V_{CC}/2 + 0.1$	V
$V_{REF(-)}$ Voltage, Bottom of Ladder	Measured at Ref(-)	-0.1	0		V
Comparator Input Current	$f_c = 640 \text{ kHz}$, (Note 6)	-2	± 0.5	2	μA

Electrical Characteristics

Digital Levels and DC Specifications: ADC0808CJ $4.5V \leq V_{CC} \leq 5.5V$, $-55^\circ\text{C} \leq T_A \leq +125^\circ\text{C}$ unless otherwise noted
ADC0808CCJ, ADC0808CCN, and ADC0809CCN $4.75 \leq V_{CC} \leq 5.25V$, $-40^\circ\text{C} \leq T_A \leq +85^\circ\text{C}$ unless otherwise noted

Parameter	Conditions	Min	Typ	Max	Units
ANALOG MULTIPLEXER					
$I_{OFF(+)}$ OFF Channel Leakage Current	$V_{CC} = 5V$, $V_{IN} = 5V$, $T_A = 25^\circ\text{C}$ T_{MIN} to T_{MAX}		10	200 1.0	nA μA
$I_{OFF(-)}$ OFF Channel Leakage Current	$V_{CC} = 5V$, $V_{IN} = 0$, $T_A = 25^\circ\text{C}$ T_{MIN} to T_{MAX}	-200 -1.0	-10		nA μA
CONTROL INPUTS					
$V_{IN(1)}$ Logical "1" Input Voltage		$V_{CC} - 1.5$			V
$V_{IN(0)}$ Logical "0" Input Voltage				1.5	V
$I_{IN(1)}$ Logical "1" Input Current (The Control Inputs)	$V_{IN} = 15V$			1.0	μA
$I_{IN(0)}$ Logical "0" Input Current (The Control Inputs)	$V_{IN} = 0$	-1.0			μA
I_{CC} Supply Current	$f_{CLK} = 640 \text{ kHz}$		0.3	3.0	mA

Electrical Characteristics (Continued)

Digital Levels and DC Specifications: ADC0808CJ $4.5V \leq V_{CC} \leq 5.5V$, $-55^{\circ}C \leq T_A \leq +125^{\circ}C$ unless otherwise noted
 ADC0808CCJ, ADC0808CCN, and ADC0809CCN $4.75 \leq V_{CC} \leq 5.25V$, $-40^{\circ}C \leq T_A \leq +85^{\circ}C$ unless otherwise noted

Parameter	Conditions	Min	Typ	Max	Units
DATA OUTPUTS AND EOC (INTERRUPT)					
$V_{OUT(1)}$	Logical "1" Output Voltage	$I_O = -360 \mu A$	$V_{CC}-0.4$		V
$V_{OUT(0)}$	Logical "0" Output Voltage	$I_O = 1.6 \text{ mA}$		0.45	V
$V_{OUT(EO)}$	Logical "0" Output Voltage EOC	$I_O = 1.2 \text{ mA}$		0.45	V
I_{OUT}	TRI-STATE Output Current	$V_O = 5V$ $V_O = 0$	-3	3	μA μA

Electrical Characteristics

Timing Specifications: $V_{CC} = V_{REF(+)} = 5V$, $V_{REF(-)} = GND$, $t_r = t_f = 20 \text{ ns}$ and $T_A = 25^{\circ}C$ unless otherwise noted.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
t_{WS}	Minimum Start Pulse Width	(Figure 5)		100	200	ns
t_{WALE}	Minimum ALE Pulse Width	(Figure 5)		100	200	ns
t_s	Minimum Address Set-Up Time	(Figure 5)		25	50	ns
t_H	Minimum Address Hold Time	(Figure 5)		25	50	ns
t_D	Analog MUX Delay Time From ALE	$R_S = 0\Omega$ (Figure 5)		1	2.5	μs
t_{H1}, t_{H0}	OE Control to Q Logic State	$C_L = 50 \text{ pF}$, $R_L = 10k$ (Figure 8)		125	250	ns
t_{1H}, t_{0H}	OE Control to Hi-Z	$C_L = 10 \text{ pF}$, $R_L = 10k$ (Figure 8)		125	250	ns
t_c	Conversion Time	$f_c = 640 \text{ kHz}$, (Figure 5) (Note 7)	90	100	116	μs
f_c	Clock Frequency		10	640	1280	kHz
t_{EOC}	EOC Delay Time	(Figure 5)	0		$8 + 2 \mu s$	Clock Periods
C_{IN}	Input Capacitance	At Control Inputs		10	15	pF
C_{OUT}	TRI-STATE® Output Capacitance	At TRI-STATE® Outputs, (Note 12)		10	15	pF

Note 1: Absolute maximum ratings are those values beyond which the life of the device may be impaired.

Note 2: All voltages are measured with respect to GND, unless otherwise specified.

Note 3: A zener diode exists, internally, from V_{CC} to GND and has a typical breakdown voltage of $7 V_{DC}$.

Note 4: Two on-chip diodes are tied to each analog input which will forward conduct for analog input voltages one diode drop below ground or one diode drop greater than the V_{CC} supply. The spec allows 100 mV forward bias of either diode. This means that as long as the analog V_{IN} does not exceed the supply voltage by more than 100 mV, the output code will be correct. To achieve an absolute 0 V_{DC} to 5 V_{DC} input voltage range will therefore require a minimum supply voltage of $4.900 V_{DC}$ over temperature variations, initial tolerance and loading.

Note 5: Total unadjusted error includes offset, full-scale, linearity, and multiplexer errors. See Figure 3. None of these A/Ds requires a zero or full-scale adjust. However, if an all zero code is desired for an analog input other than 0.0V, or if a narrow full-scale span exists (for example: 0.5V to 4.5V full-scale) the reference voltages can be adjusted to achieve this. See Figure 13.

Note 6: Comparator input current is a bias current into or out of the chopper stabilized comparator. The bias current varies directly with clock frequency and has little temperature dependence (Figure 6). See paragraph 4.0.

Note 7: The outputs of the data register are updated one clock cycle before the rising edge of EOC.

Functional Description

Multiplexer: The device contains an 8-channel single-ended analog signal multiplexer. A particular input channel is selected by using the address decoder. Table I shows the input states for the address lines to select any channel. The address is latched into the decoder on the low-to-high transition of the address latch enable signal.

TABLE I

SELECTED ANALOG CHANNEL	ADDRESS LINE		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

CONVERTER CHARACTERISTICS

The Converter

The heart of this single chip data acquisition system is its 8-bit analog-to-digital converter. The converter is designed

to give fast, accurate, and repeatable conversions over a wide range of temperatures. The converter is partitioned into 3 major sections: the 256R ladder network, the successive approximation register, and the comparator. The converter's digital outputs are positive true.

The 256R ladder network approach (*Figure 1*) was chosen over the conventional R/2R ladder because of its inherent monotonicity, which guarantees no missing digital codes. Monotonicity is particularly important in closed loop feedback control systems. A non-monotonic relationship can cause oscillations that will be catastrophic for the system. Additionally, the 256R network does not cause load variations on the reference voltage.

The bottom resistor and the top resistor of the ladder network in *Figure 1* are not the same value as the remainder of the network. The difference in these resistors causes the output characteristic to be symmetrical with the zero and full-scale points of the transfer curve. The first output transition occurs when the analog signal has reached + 1/2 LSB and succeeding output transitions occur every 1 LSB later up to full-scale.

The successive approximation register (SAR) performs 8 iterations to approximate the input voltage. For any SAR type converter, n-iterations are required for an n-bit converter. *Figure 2* shows a typical example of a 3-bit converter. In the ADC0808, ADC0809, the approximation technique is extended to 8 bits using the 256R network.

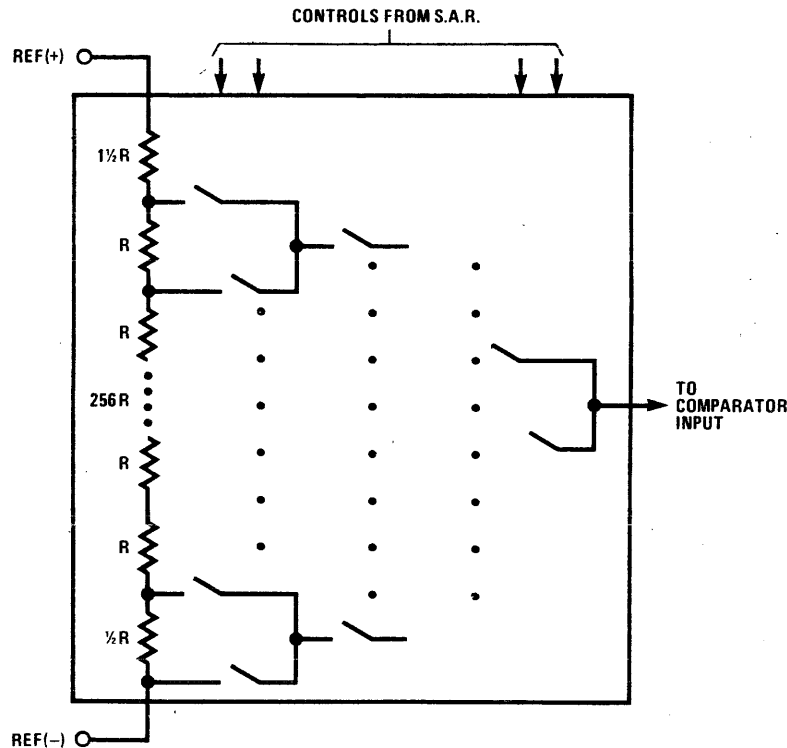


FIGURE 1. Resistor Ladder and Switch Tree

Functional Description (Continued)

The A/D converter's successive approximation register (SAR) is reset on the positive edge of the start conversion (SC) pulse. The conversion is begun on the falling edge of the start conversion pulse. A conversion in process will be interrupted by receipt of a new start conversion pulse. Continuous conversion may be accomplished by tying the end-of-conversion (EOC) output to the SC input. If used in this mode, an external start conversion pulse should be applied after power up. End-of-conversion will go low between 0 and 8 clock pulses after the rising edge of start conversion.

The most important section of the A/D converter is the comparator. It is this section which is responsible for the ultimate accuracy of the entire converter. It is also the

comparator drift which has the greatest influence on the repeatability of the device. A chopper-stabilized comparator provides the most effective method of satisfying all the converter requirements.

The chopper-stabilized comparator converts the DC input signal into an AC signal. This signal is then fed through a high gain AC amplifier and has the DC level restored. This technique limits the drift component of the amplifier since the drift is a DC component which is not passed by the AC amplifier. This makes the entire A/D converter extremely insensitive to temperature, long term drift and input offset errors.

Figure 4 shows a typical error curve for the ADC0808 as measured using the procedures outlined in AN-179.

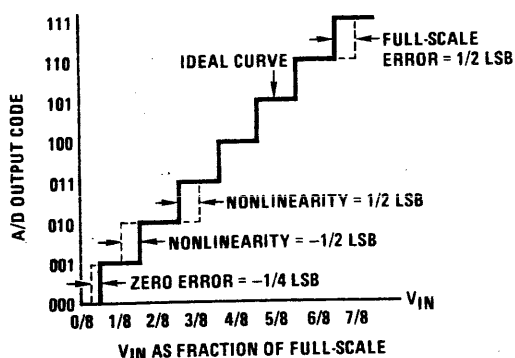


FIGURE 2. 3-Bit A/D Transfer Curve

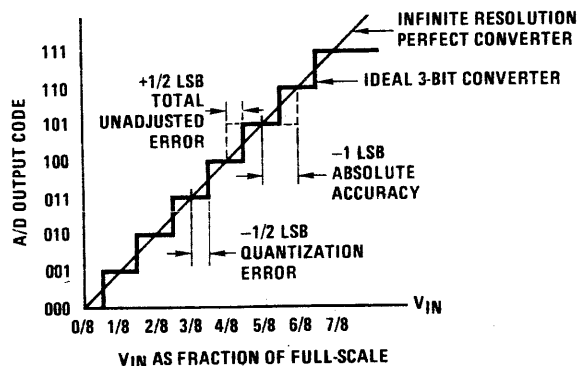


FIGURE 3. 3-Bit A/D Absolute Accuracy Curve

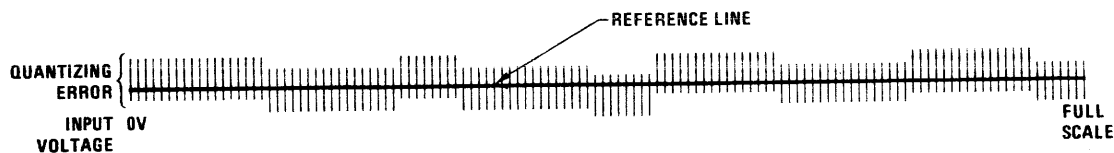
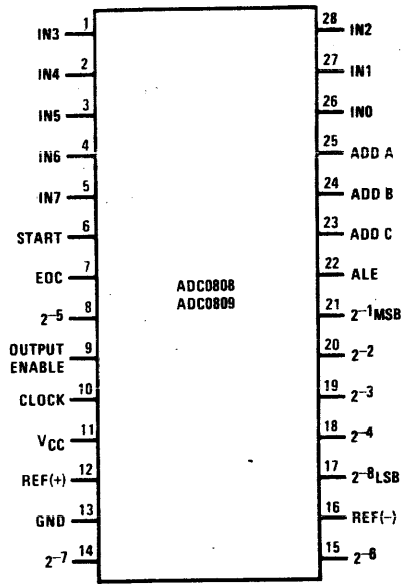


FIGURE 4. Typical Error Curve

Connection Diagram

Dual-In-Line Package



TOP VIEW

Timing Diagram

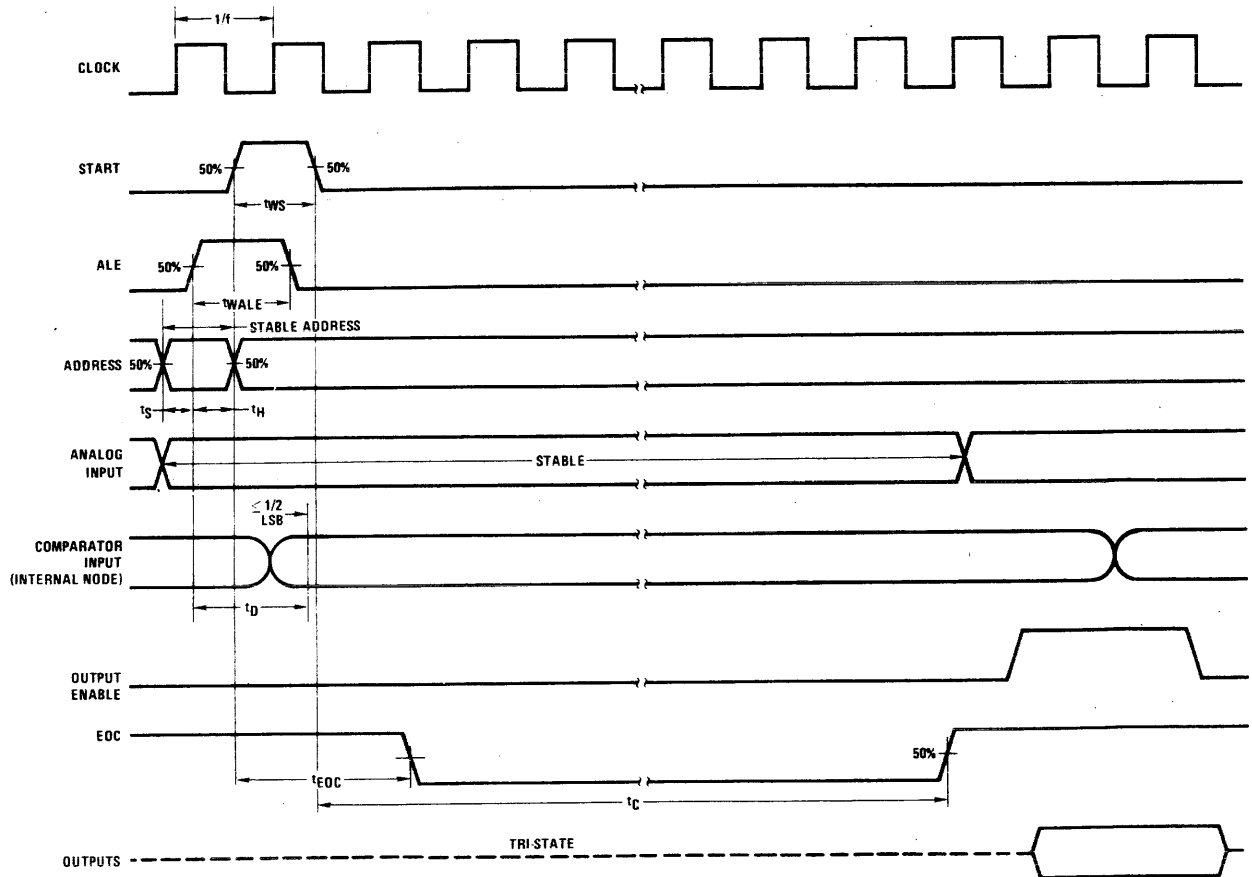


FIGURE 5

Typical Performance Characteristics

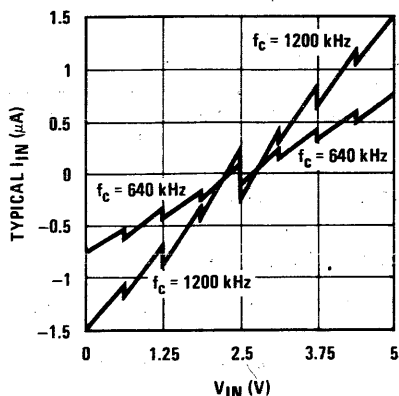


FIGURE 6. Comparator I_{IN} vs V_{IN}
($V_{CC} = V_{REF} = 5V$)

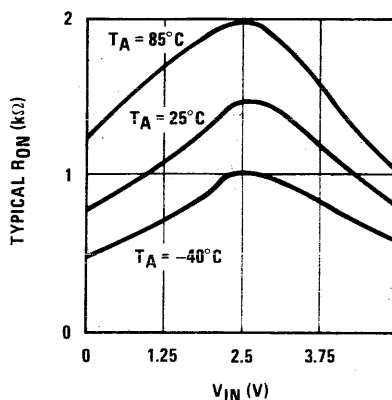


FIGURE 7. Multiplexer R_{ON} vs V_{IN}
($V_{CC} = V_{REF} = 5V$)

TRI-STATE® Test Circuits and Timing Diagrams

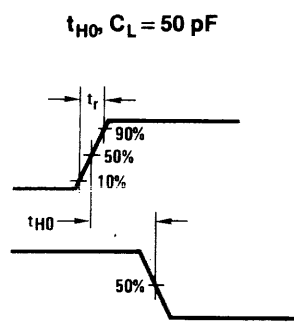
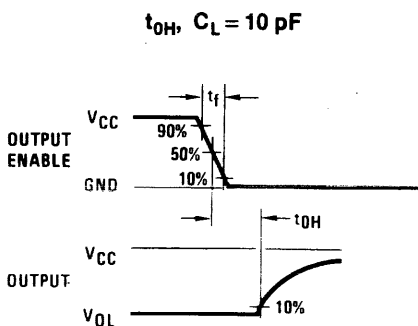
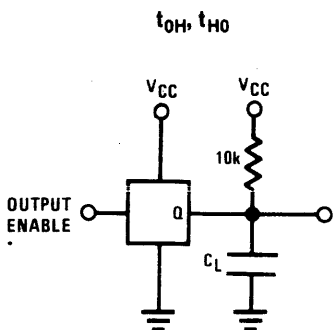
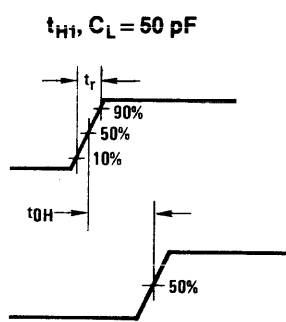
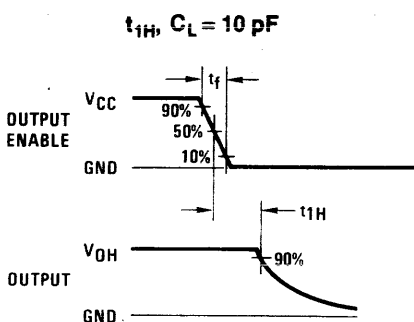
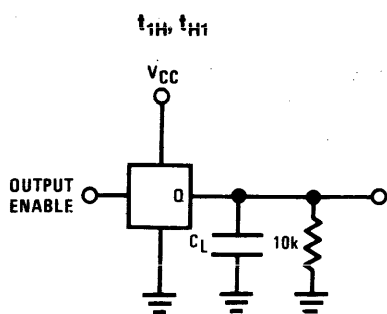


FIGURE 8

Applications Information

OPERATION

1.0 Ratiometric Conversion

The ADC0808, ADC0809 is designed as a complete Data Acquisition System (DAS) for ratiometric conversion systems. In ratiometric systems, the physical variable being measured is expressed as a percentage of full-scale which is not necessarily related to an absolute standard. The voltage input to the ADC0808 is expressed by the equation

$$\frac{V_{IN} - V_Z}{V_{fs} - V_Z} = \frac{D_X}{D_{MAX} - D_{MIN}} \quad (1)$$

V_{IN} = Input voltage into the ADC0808

V_{fs} = Full-scale voltage

V_Z = Zero voltage

D_X = Data point being measured

D_{MAX} = Maximum data limit

D_{MIN} = Minimum data limit

A good example of a ratiometric transducer is a potentiometer used as a position sensor. The position of the wiper is directly proportional to the output voltage which is a ratio of the full-scale voltage across it. Since the data is represented as a proportion of full-scale, reference requirements are greatly reduced, eliminating a large source of error and cost for many applications. A major advantage of the ADC0808, ADC0809 is that the input voltage range is equal to the supply range so the transducers can be connected directly across the supply and their outputs connected directly into the multiplexer inputs, (Figure 9).

Ratiometric transducers such as potentiometers, strain gauges, thermistor bridges, pressure transducers, etc., are suitable for measuring proportional relationships; however, many types of measurements must be referred to an absolute standard such as voltage or current. This means a system reference must be used which relates the full-scale voltage to the standard volt. For example, if $V_{CC} = V_{REF} = 5.12V$, then the full-scale range is divided into 256 standard steps. The smallest standard step is 1 LSB which is then 20 mV.

2.0 Resistor Ladder Limitations

The voltages from the resistor ladder are compared to the selected input 8 times in a conversion. These voltages are coupled to the comparator via an analog switch tree which is referenced to the supply. The voltages at the top, center and bottom of the ladder must be controlled to maintain proper operation.

The top of the ladder, Ref(+), should not be more positive than the supply, and the bottom of the ladder, Ref(-), should not be more negative than ground. The center of the ladder voltage must also be near the center of the supply because the analog switch tree changes from N-channel switches to P-channel switches. These limitations are automatically satisfied in ratiometric systems and can be easily met in ground referenced systems.

Figure 10 shows a ground referenced system with a separate supply and reference. In this system, the supply must be trimmed to match the reference voltage. For instance, if a 5.12V is used, the supply should be adjusted to the same voltage within 0.1V.

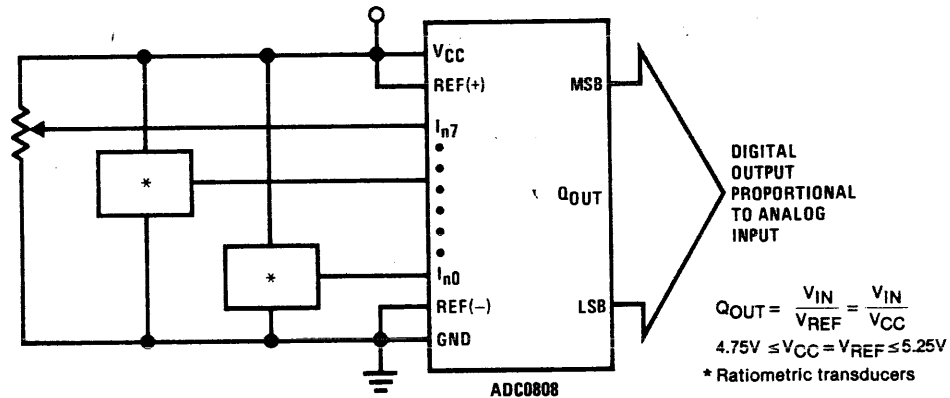


FIGURE 9. Ratiometric Conversion System

Applications Information (Continued)

The ADC0808 needs less than a milliamp of supply current so developing the supply from the reference is readily accomplished. In *Figure 11* a ground referenced system is shown which generates the supply from the reference. The buffer shown can be an op amp of sufficient drive to supply the milliamp of supply current and the desired bus drive, or if a capacitive bus is driven by the outputs a large capacitor will supply the transient supply current as seen in *Figure 12*. The LM301 is overcompensated to insure stability when loaded by the 10 μ F output capacitor.

The top and bottom ladder voltages cannot exceed V_{CC} and ground, respectively, but they can be symmetrically less than V_{CC} and greater than ground. The center of the ladder voltage should always be near the center of the supply. The sensitivity of the converter can be increased, (i.e., size of the LSB steps decreased) by using a symmetrical reference system. In *Figure 13*, a 2.5V reference is symmetrically centered about $V_{CC}/2$ since the same current flows in identical resistors. This system with a 2.5V reference allows the LSB bit to be half the size of a 5V reference system.

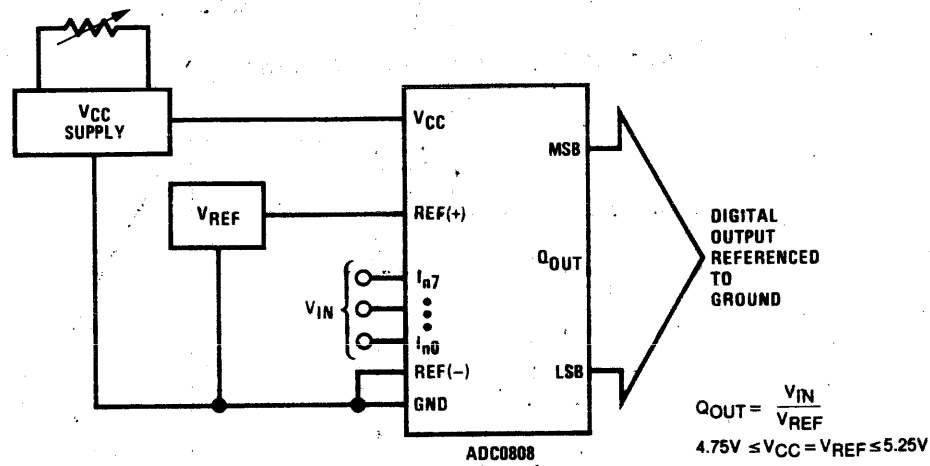


FIGURE 10. Ground Referenced Conversion System Using Trimmed Supply

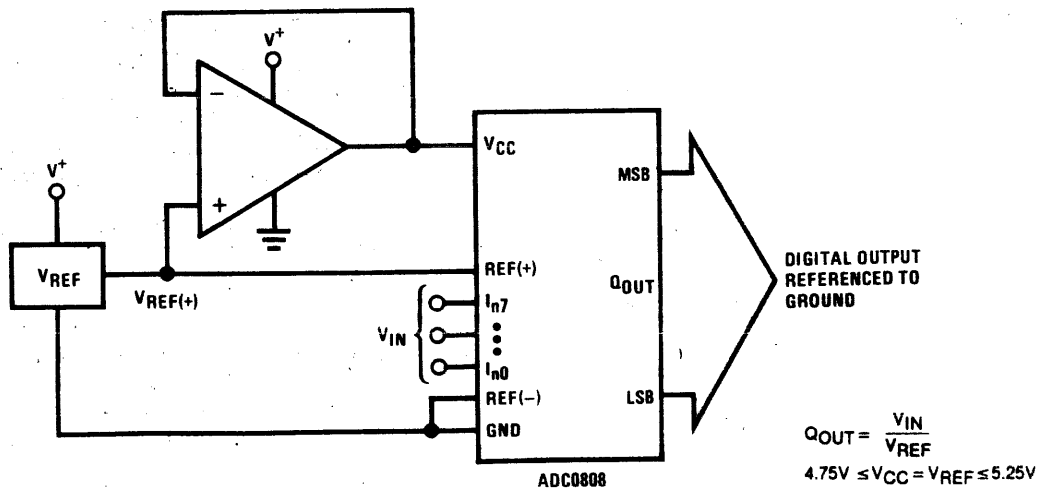


FIGURE 11. Ground Referenced Conversion System with Reference Generating V_{CC} Supply

Applications Information (Continued)

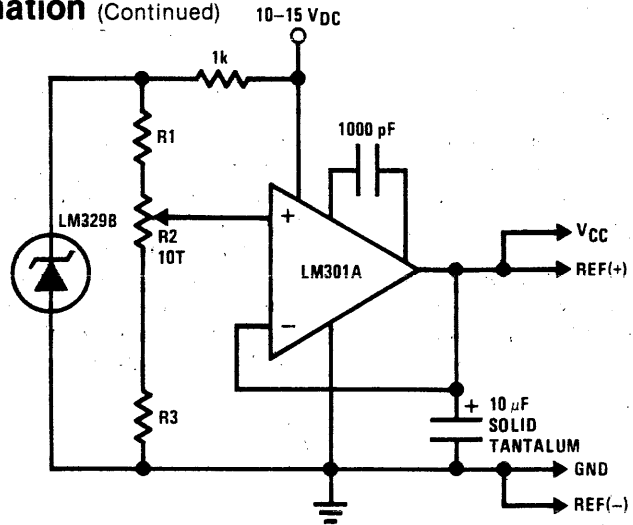


FIGURE 12. Typical Reference and Supply Circuit

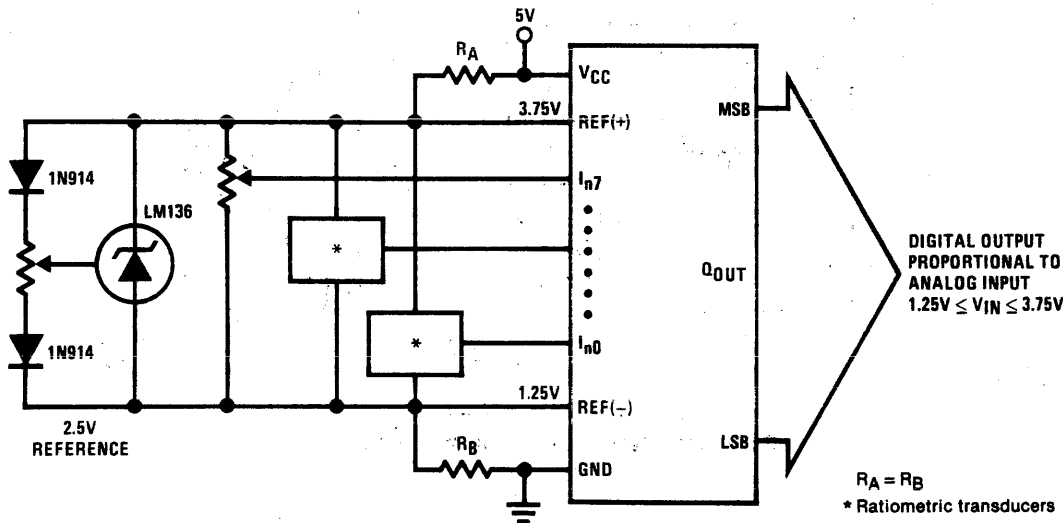


FIGURE 13. Symmetrically Centered Reference

3.0 Converter Equations

The transition between adjacent codes N and N + 1 is given by:

$$V_{IN} = \left\{ (V_{REF(+)} - V_{REF(-)}) \left[\frac{N}{256} + \frac{1}{512} \right] \pm V_{TUE} \right\} + V_{REF(-)} \quad (2)$$

The center of an output code N is given by:

$$V_{IN} = \left\{ (V_{REF(+)} - V_{REF(-)}) \left[\frac{N}{256} \right] \pm V_{TUE} \right\} + V_{REF(-)} \quad (3)$$

The output code N for an arbitrary input are the integers within the range:

$$N = \frac{V_{IN} - V_{REF(-)}}{V_{REF(+)} - V_{REF(-)}} \times 256 \pm \text{Absolute Accuracy} \quad (4)$$

- where: V_{IN} = Voltage at comparator input
- $V_{REF(+)}$ = Voltage at Ref (+)
- $V_{REF(-)}$ = Voltage at Ref (-)
- V_{TUE} = Total unadjusted error voltage (typically $V_{REF(+)} \div 512$)

4.0 Analog Comparator Inputs

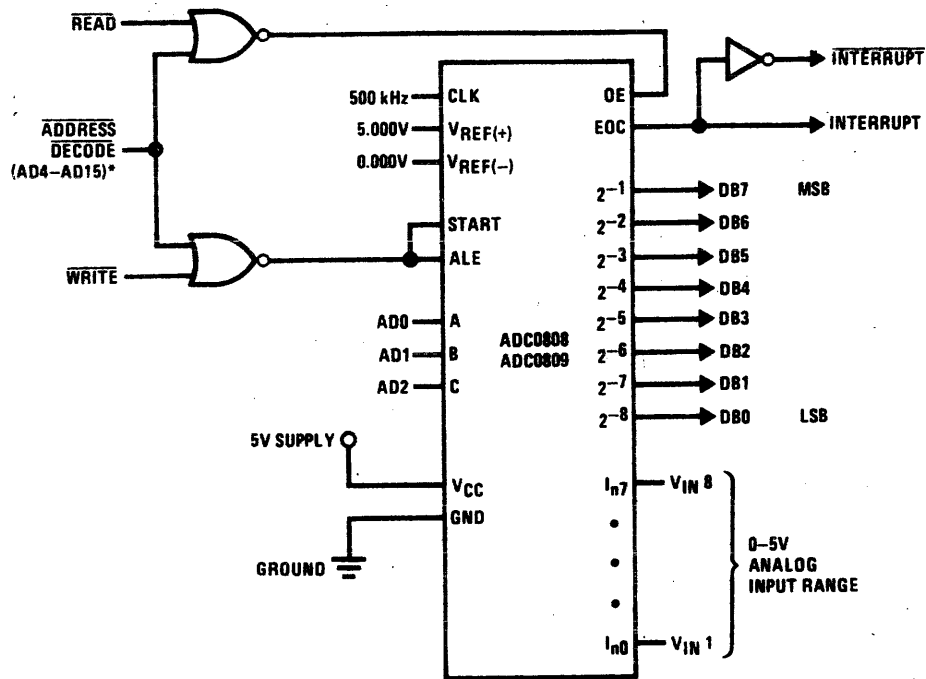
The dynamic comparator input current is caused by the periodic switching of on-chip stray capacitances. These are connected alternately to the output of the resistor ladder/switch tree network and to the comparator input as part of the operation of the chopper stabilized comparator.

The average value of the comparator input current varies directly with clock frequency and with V_{IN} as shown in Figure 6.

If no filter capacitors are used at the analog inputs and the signal source impedances are low, the comparator input current should not introduce converter errors, as the transient created by the capacitance discharge will die out before the comparator output is strobed.

If input filter capacitors are desired for noise reduction and signal conditioning they will tend to average out the dynamic comparator input current. It will then take on the characteristics of a DC bias current whose effect can be predicted conventionally.

Typical Application



* Address latches needed for 8085 and SC/MP interfacing the ADC0808 to a microprocessor

MICROPROCESSOR INTERFACE TABLE

PROCESSOR	READ	WRITE	INTERRUPT (COMMENT)
8080	MEMR	MEMW	INTR (Thru RST Circuit)
8085	RD	WR	INTR (Thru RST Circuit)
Z-80	RD	WR	INT (Thru RST Circuit, Mode 0)
SC/MP	NRDS	NWDS	SA (Thru Sense A)
6800	VMA-2-R/W	VMA-2-R/W	IRQA or IRQB (Thru PIA)

Ordering Information

TEMPERATURE RANGE		- 40°C to +85°C		- 55°C to +125°C
Error	± 1/2 Bit Unadjusted	ADC0808CCN	ADC0808CCJ	ADC0808CJ
	± 1 Bit Unadjusted	ADC0809CCN		
Package Outline		N28A Molded DIP	J28A Hermetic DIP	J28A Hermetic DIP

Appendix E

8253 Data Sheet
(CTC)

The following material is copyrighted by Intel Corporation. It is reprinted here with the permission of Intel. This data sheet may not be reproduced for any purpose in whole or part without the expressed written consent of Intel.



8253/8253-5 PROGRAMMABLE INTERVAL TIMER

- MCS-85™ Compatible 8253-5
- 3 Independent 16-Bit Counters
- DC to 2 MHz
- Programmable Counter Modes
- Count Binary or BCD
- Single +5V Supply
- 24-Pin Dual In-Line Package

The Intel® 8253 is a programmable counter/timer chip designed for use as an Intel microcomputer peripheral. It uses nMOS technology with a single +5V supply and is packaged in a 24-pin plastic DIP.

It is organized as 3 independent 16-bit counters, each with a count rate of up to 2 MHz. All modes of operation are software programmable.

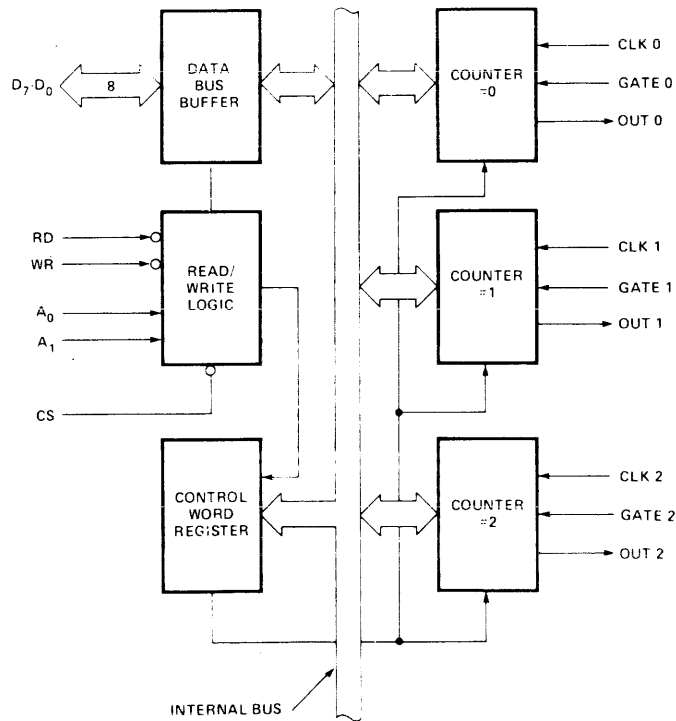


Figure 1. Block Diagram

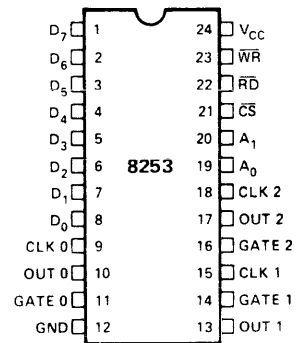


Figure 2. Pin Configuration

FUNCTIONAL DESCRIPTION

General

The 8253 is a programmable interval timer/counter specifically designed for use with the Intel™ Micro-computer systems. Its function is that of a general purpose, multi-timing element that can be treated as an array of I/O ports in the system software.

The 8253 solves one of the most common problems in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in systems software, the programmer configures the 8253 to match his requirements, initializes one of the counters of the 8253 with the desired quantity, then upon command the 8253 will count out the delay and interrupt the CPU when it has completed its tasks. It is easy to see that the software overhead is minimal and that multiple delays can easily be maintained by assignment of priority levels.

Other counter/timer functions that are non-delay in nature but also common to most microcomputers can be implemented with the 8253.

- Programmable Rate Generator
- Event Counter
- Binary Rate Multiplier
- Real Time Clock
- Digital One-Shot
- Complex Motor Controller

Data Bus Buffer

This 3-state, bi-directional, 8-bit buffer is used to interface the 8253 to the system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput CPU instructions. The Data Bus Buffer has three basic functions.

1. Programming the MODES of the 8253.
2. Loading the count registers.
3. Reading the count values.

Read/Write Logic

The Read/Write Logic accepts inputs from the system bus and in turn generates control signals for overall device operation. It is enabled or disabled by CS so that no operation can occur to change the function unless the device has been selected by the system logic.

\overline{RD} (Read)

A "low" on this input informs the 8253 that the CPU is inputting data in the form of a counters value.

\overline{WR} (Write)

A "low" on this input informs the 8253 that the CPU is outputting data in the form of mode information or loading counters.

A0, A1

These inputs are normally connected to the address bus. Their function is to select one of the three counters to be operated on and to address the control word register for mode selection.

\overline{CS} (Chip Select)

A "low" on this input enables the 8253. No reading or writing will occur unless the device is selected. The \overline{CS} input has no effect upon the actual operation of the counters.

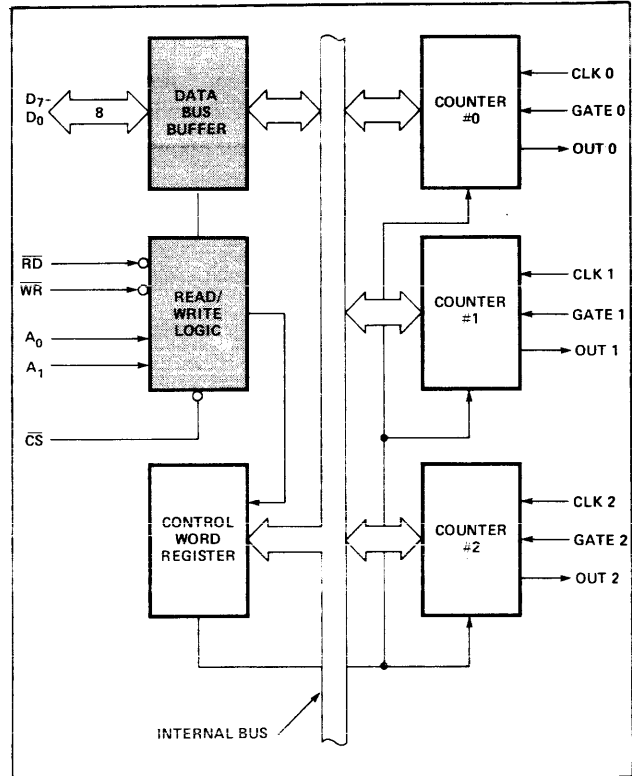


Figure 3. Block Diagram Showing Data Bus Buffer and Read/Write Logic Functions

\overline{CS}	\overline{RD}	\overline{WR}	A ₁	A ₀	
0	1	0	0	0	Load Counter No. 0
0	1	0	0	1	Load Counter No. 1
0	1	0	1	0	Load Counter No. 2
0	1	0	1	1	Write Mode Word
0	0	1	0	0	Read Counter No. 0
0	0	1	0	1	Read Counter No. 1
0	0	1	1	0	Read Counter No. 2
0	0	1	1	1	No-Operation 3-State
1	X	X	X	X	Disable 3-State
0	1	1	X	X	No-Operation 3-State

Control Word Register

The Control Word Register is selected when A0, A1 are 11. It then accepts information from the data bus buffer and stores it in a register. The information stored in this register controls the operational MODE of each counter, selection of binary or BCD counting and the loading of each count register.

The Control Word Register can only be written into; no read operation of its contents is available.

Counter #0, Counter #1, Counter #2

These three functional blocks are identical in operation so only a single Counter will be described. Each Counter consists of a single, 16-bit, pre-settable, DOWN counter. The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of MODES stored in the Control Word Register.

The counters are fully independent and each can have separate Mode configuration and counting operation, binary or BCD. Also, there are special features in the control word that handle the loading of the count value so that software overhead can be minimized for these functions.

The reading of the contents of each counter is available to the programmer with simple READ operations for event counting applications and special commands and logic are included in the 8253 so that the contents of each counter can be read "on the fly" without having to inhibit the clock input.

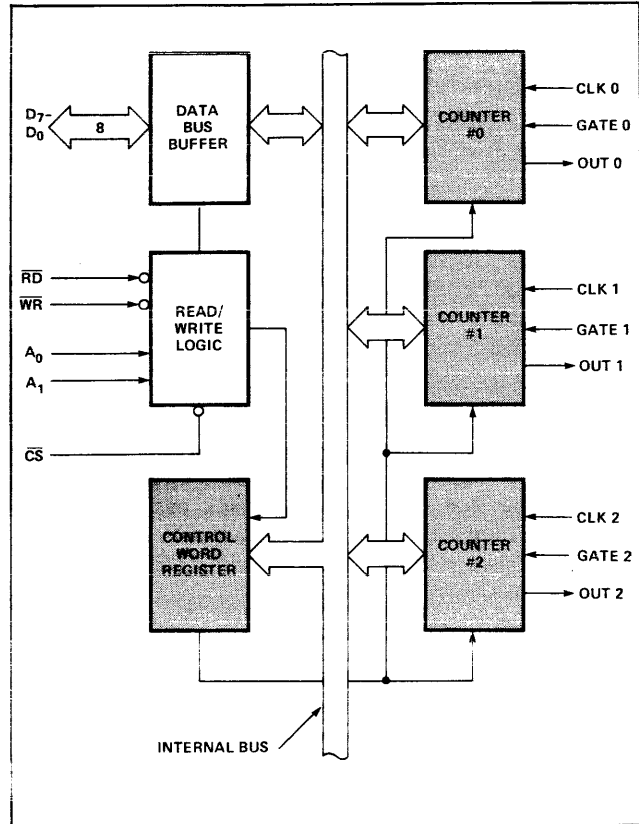


Figure 4. Block Diagram Showing Control Word Register and Counter Functions

8253 SYSTEM INTERFACE

The 8253 is a component of the Intel™ Microcomputer Systems and interfaces in the same manner as all other peripherals of the family. It is treated by the systems software as an array of peripheral I/O ports; three are counters and the fourth is a control register for MODE programming.

Basically, the select inputs A0, A1 connect to the A0, A1 address bus signals of the CPU. The CS can be derived directly from the address bus using a linear select method. Or it can be connected to the output of a decoder, such as an Intel® 8205 for larger systems.

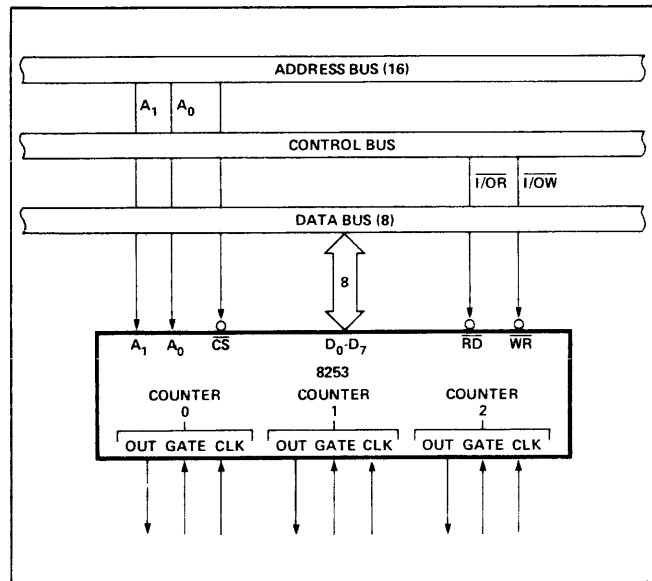


Figure 5. 8253 System Interface

OPERATIONAL DESCRIPTION

General

The complete functional definition of the 8253 is programmed by the systems software. A set of control words must be sent out by the CPU to initialize each counter of the 8253 with the desired MODE and quantity information. Prior to initialization, the MODE, count, and output of all counters is undefined. These control words program the MODE, Loading sequence and selection of binary or BCD counting.

Once programmed, the 8253 is ready to perform whatever timing tasks it is assigned to accomplish.

The actual counting operation of each counter is completely independent and additional logic is provided on-chip so that the usual problems associated with efficient monitoring and management of external, asynchronous events or rates to the microcomputer system have been eliminated.

Programming the 8253

All of the MODES for each counter are programmed by the systems software by simple I/O operations.

Each counter of the 8253 is individually programmed by writing a control word into the Control Word Register. (A0, A1 = 11)

Control Word Format

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

Definition of Control

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

RL — Read/Load:

RL1	RL0	
0	0	Counter Latching operation (see READ/WRITE Procedure Section)
1	0	Read/Load most significant byte only.
0	1	Read/Load least significant byte only.
1	1	Read/Load least significant byte first, then most significant byte.

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Counter Loading

The count register is not loaded until the count value is written (one or two bytes, depending on the mode selected by the RL bits), followed by a rising edge and a falling edge of the clock. Any read of the counter prior to that falling clock edge may yield invalid data.

MODE Definition

MODE 0: Interrupt on Terminal Count. The output will be initially low after the mode set operation. After the count is loaded into the selected count register, the output will remain low and the counter will count. When terminal count is reached the output will go high and remain high until the selected count register is reloaded with the mode or a new count is loaded. The counter continues to decrement after terminal count has been reached.

Rewriting a counter register during counting results in the following:

- (1) Write 1st byte stops the current counting.
- (2) Write 2nd byte starts the new count.

MODE 1: Programmable One-Shot. The output will go low on the count following the rising edge of the gate input.

The output will go high on the terminal count. If a new count value is loaded while the output is low it will not affect the duration of the one-shot pulse until the succeeding trigger. The current count can be read at any time without affecting the one-shot pulse.

The one-shot is retriggerable, hence the output will remain low for the full count after any rising edge of the gate input.

MODE 2: Rate Generator. Divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. If the count register is reloaded between output pulses the present period will not be affected, but the subsequent period will reflect the new value.

The gate input, when low, will force the output high. When the gate input goes high, the counter will start from the initial count. Thus, the gate input can be used to synchronize the counter.

When this mode is set, the output will remain high until after the count register is loaded. The output then can also be synchronized by software.

MODE 3: Square Wave Rate Generator. Similar to MODE 2 except that the output will remain high until one half the count has been completed (for even numbers) and go low for the other half of the count. This is accomplished by decrementing the counter by two on the falling edge of each clock pulse. When the counter reaches terminal count, the state of the output is changed and the counter is reloaded with the full count and the whole process is repeated.

If the count is odd and the output is high, the first clock pulse (after the count is loaded) decrements the count by 1. Subsequent clock pulses decrement the clock by 2. After timeout, the output goes low and the full count is reloaded. The first clock pulse (following the reload) decrements the counter by 3. Subsequent clock pulses decrement the count by 2 until timeout. Then the whole process is repeated. In this way, if the count is odd, the output will be high for $(N + 1)/2$ counts and low for $(N - 1)/2$ counts.

MODE 4: Software Triggered Strobe. After the mode is set, the output will be high. When the count is loaded, the counter will begin counting. On terminal count, the output will go low for one input clock period, then will go high again.

If the count register is reloaded between output pulses, counting will continue from the new value. The count will be inhibited while the gate input is low. Reloading the counter register will restart counting beginning with the new number.

MODE 5: Hardware Triggered Strobe. The counter will start counting after the rising edge of the trigger input and will go low for one clock period when the terminal count is reached. The counter is retriggerable. The output will not go low until the full count after the rising edge of any trigger.

Modes	Signal Status	Low Or Going Low	Rising	High
0		Disables counting	---	Enables counting
1		---	1) Initiates counting 2) Resets output after next clock	---
2		1) Disables counting 2) Sets output immediately high	1) Reloads counter 2) Initiates counting	Enables counting
3		1) Disables counting 2) Sets output immediately high	Initiates counting	Enables counting
4		Disables counting	---	Enables counting
5		---	Initiates counting	---

Figure 6. Gate Pin Operations Summary

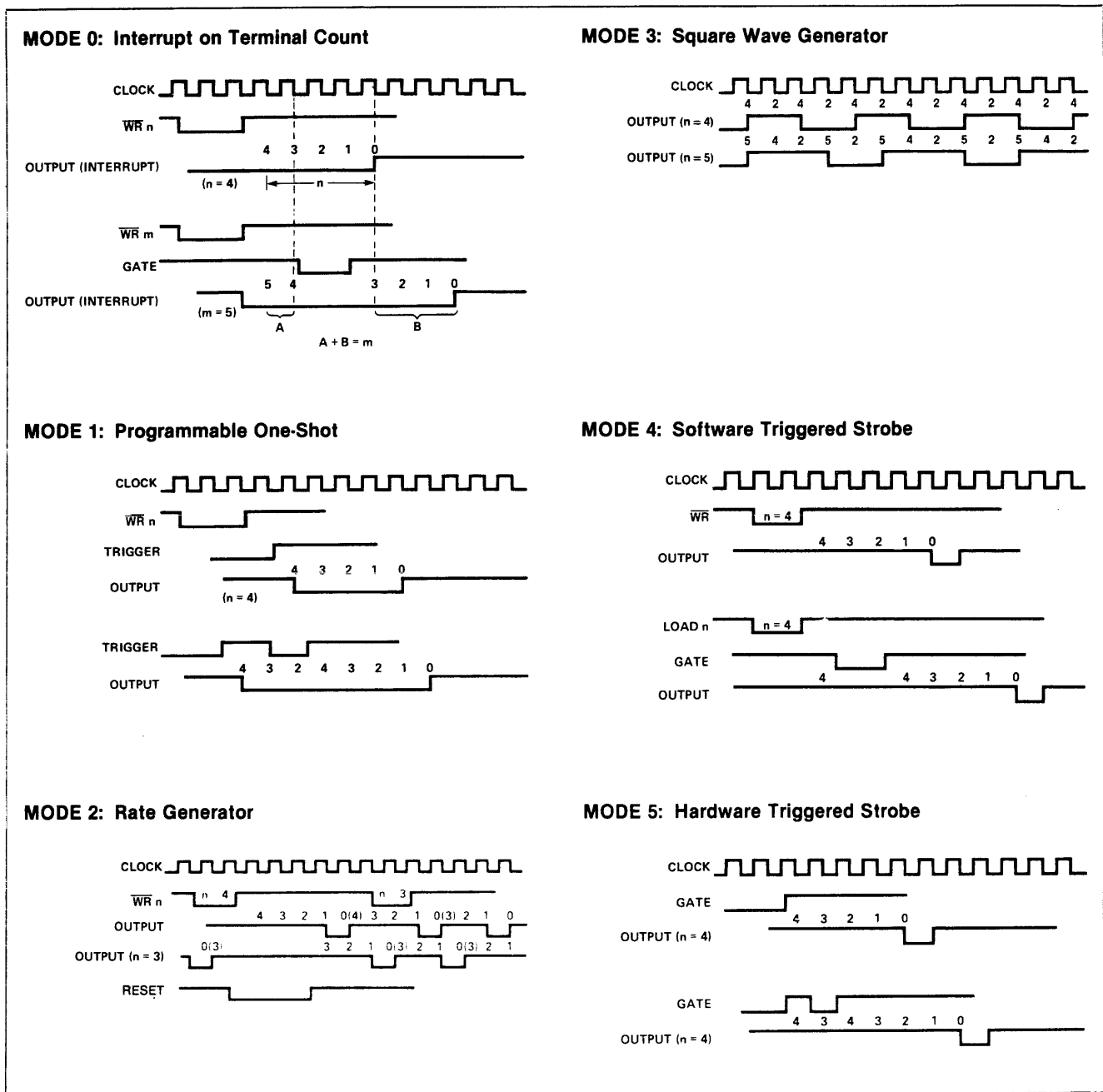


Figure 7. 8253 Timing Diagrams

8253 READ/WRITE PROCEDURE

Write Operations

The systems software must program each counter of the 8253 with the mode and quantity desired. The programmer must write out to the 8253 a MODE control word and the programmed number of count register bytes (1 or 2) prior to actually using the selected counter.

The actual order of the programming is quite flexible. Writing out of the MODE control word can be in any sequence of counter selection, e.g., counter #0 does not have to be first or counter #2 last. Each counter's MODE control word register has a separate address so that its loading is completely sequence independent. (SC0, SC1)

The loading of the Count Register with the actual count value, however, must be done in exactly the sequence programmed in the MODE control word (RL0, RL1). This loading of the counter's count register is still sequence independent like the MODE control word loading, but when a selected count register is to be loaded it must be loaded with the number of bytes programmed in the MODE control word (RL0, RL1). The one or two bytes to be loaded in the count register do not have to follow the associated MODE control word. They can be programmed at any time following the MODE control word loading as long as the correct number of bytes is loaded in order.

All counters are down counters. Thus, the value loaded into the count register will actually be decremented. Loading all zeroes into a count register will result in the maximum count (2^{16} for Binary or 10^4 for BCD). In MODE 0 the new count will not restart until the load has been completed. It will accept one of two bytes depending on how the MODE control words (RL0, RL1) are programmed. Then proceed with the restart operation.

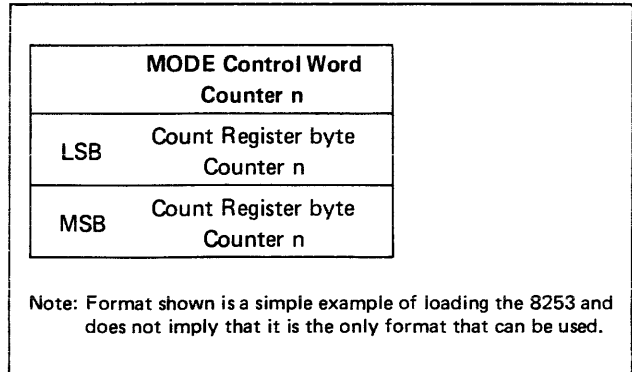


Figure 8. Programming Format

		A1	A0
No. 1	MODE Control Word Counter 0	1	1
No. 2	MODE Control Word Counter 1	1	1
No. 3	MODE Control Word Counter 2	1	1
No. 4	LSB Count Register Byte Counter 1	0	1
No. 5	MSB Count Register Byte Counter 1	0	1
No. 6	LSB Count Register Byte Counter 2	1	0
No. 7	MSB Count Register Byte Counter 2	1	0
No. 8	LSB Count Register Byte Counter 0	0	0
No. 9	MSB Count Register Byte Counter 0	0	0

Note: The exclusive addresses of each counter's count register make the task of programming the 8253 a very simple matter, and maximum effective use of the device will result if this feature is fully utilized.

Figure 9. Alternate Programming Formats

Read Operations

In most counter applications it becomes necessary to read the value of the count in progress and make a computational decision based on this quantity. Event counters are probably the most common application that uses this function. The 8253 contains logic that will allow the programmer to easily read the contents of any of the three counters without disturbing the actual count in progress.

There are two methods that the programmer can use to read the value of the counters. The first method involves the use of simple I/O read operations of the selected counter. By controlling the A0, A1 inputs to the 8253 the programmer can select the counter to be read (remember that no read operation of the mode register is allowed A0, A1-11). The only requirement with this method is that in order to assure a stable count reading the actual operation of the selected counter must be inhibited either by controlling the Gate input or by external logic that inhibits the clock input. The contents of the counter selected will be available as follows:

- first I/O Read contains the least significant byte (LSB).
- second I/O Read contains the most significant byte (MSB).

Due to the internal logic of the 8253 it is absolutely necessary to complete the entire reading procedure. If two bytes are programmed to be read then two bytes must be read before any loading WR command can be sent to the same counter.

Read Operation Chart

A1	A0	RD	
0	0	0	Read Counter No. 0
0	1	0	Read Counter No. 1
1	0	0	Read Counter No. 2
1	1	0	Illegal

Reading While Counting

In order for the programmer to read the contents of any counter without effecting or disturbing the counting operation the 8253 has special internal logic that can be accessed using simple WR commands to the MODE register. Basically, when the programmer wishes to read the contents of a selected counter "on the fly" he loads the MODE register with a special code which latches the present count value into a storage register so that its contents contain an accurate, stable quantity. The programmer then issues a normal read command to the selected counter and the contents of the latched register is available.

MODE Register for Latching Count

A0, A1 = 11

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	X	X	X	X

- SC1,SC0 — specify counter to be latched.
- D5,D4 — 00 designates counter latching operation.
- X — don't care.

The same limitation applies to this mode of reading the counter as the previous method. That is, it is mandatory to complete the entire read operation as programmed. This command has no effect on the counter's mode.

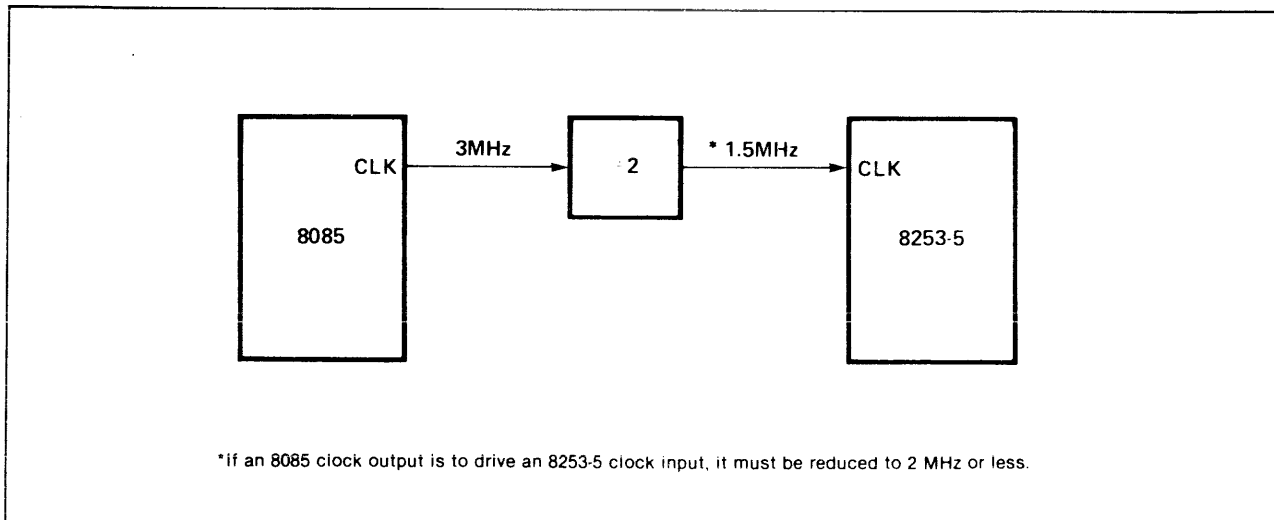


Figure 10. MCS-85™ Clock Interface*

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias 0°C to 70°C
 Storage Temperature -65°C to +150°C
 Voltage On Any Pin
 With Respect to Ground -0.5 V to +7 V
 Power Dissipation 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS ($T_A = 0^\circ\text{C to } 70^\circ\text{C}$, $V_{CC} = 5\text{V} \pm 10\%$)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.2	$V_{CC} + .5\text{V}$	V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH}	Output High Voltage	2.4		V	Note 2
I_{IL}	Input Load Current		± 10	μA	$V_{IN} = V_{CC}$ to 0V
I_{OFL}	Output Float Leakage		± 10	μA	$V_{OUT} = V_{CC}$ to 0V
I_{CC}	V_{CC} Supply Current		140	mA	

CAPACITANCE ($T_A = 25^\circ\text{C}$, $V_{CC} = \text{GND} = 0\text{V}$)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	$f_c = 1\text{ MHz}$
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS ($T_A = 0^\circ\text{C to } 70^\circ\text{C}$, $V_{CC} = 5.0\text{V} \pm 5\%$, $\text{GND} = 0\text{V}$)

Bus Parameters (Note 3)
READ CYCLE

Symbol	Parameter	8253		8253-5		Unit
		Min.	Max.	Min.	Max.	
t_{AR}	Address Stable Before $\overline{\text{READ}}$	50		30		ns
t_{RA}	Address Hold Time for $\overline{\text{READ}}$	5		5		ns
t_{RR}	$\overline{\text{READ}}$ Pulse Width	400		300		ns
t_{RD}	Data Delay From $\overline{\text{READ}}$ [4]		300		200	ns
t_{DF}	$\overline{\text{READ}}$ to Data Floating	25	125	25	100	ns
t_{RV}	Recovery Time Between READ and Any Other Control Signal	1		1		μs

A.C. CHARACTERISTICS (Continued)
WRITE CYCLE

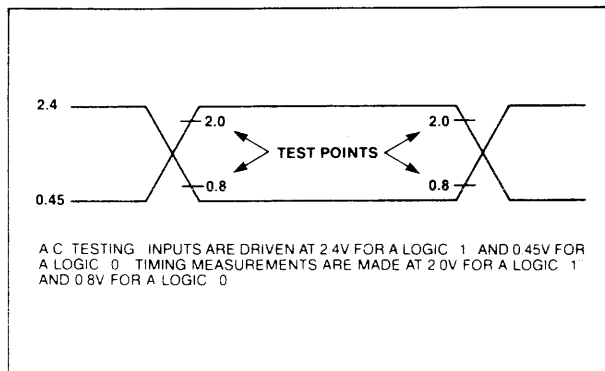
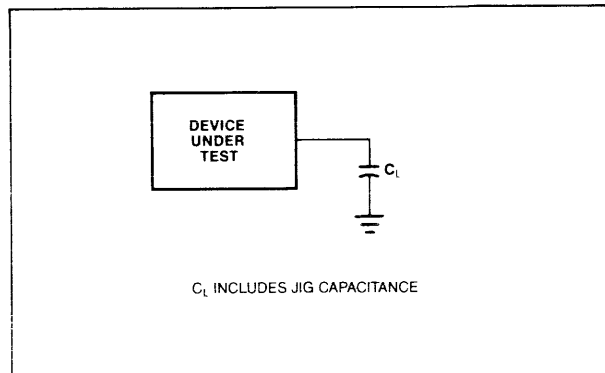
Symbol	Parameter	8253		8253-5		Unit
		Min.	Max.	Min.	Max.	
t_{AW}	Address Stable Before \overline{WRITE}	50		30		ns
t_{WA}	Address Hold Time for \overline{WRITE}	30		30		ns
t_{WW}	\overline{WRITE} Pulse Width	400		300		ns
t_{DW}	Data Set Up Time for \overline{WRITE}	300		250		ns
t_{WD}	Data Hold Time for \overline{WRITE}	40		30		ns
t_{RV}	Recovery Time Between \overline{WRITE} and Any Other Control Signal	1		1		μ s

CLOCK AND GATE TIMING

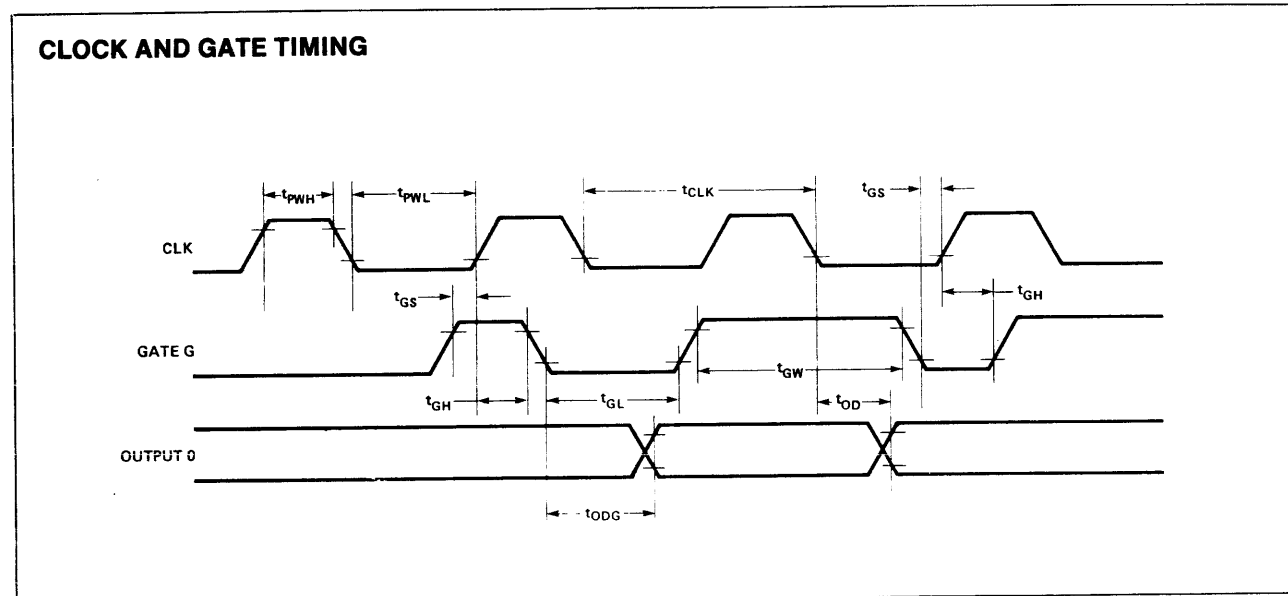
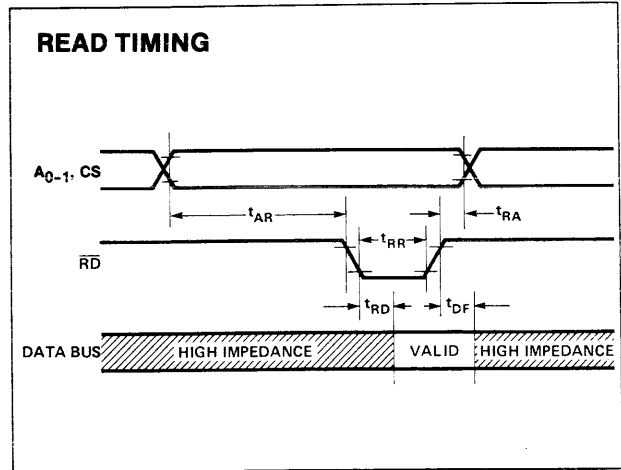
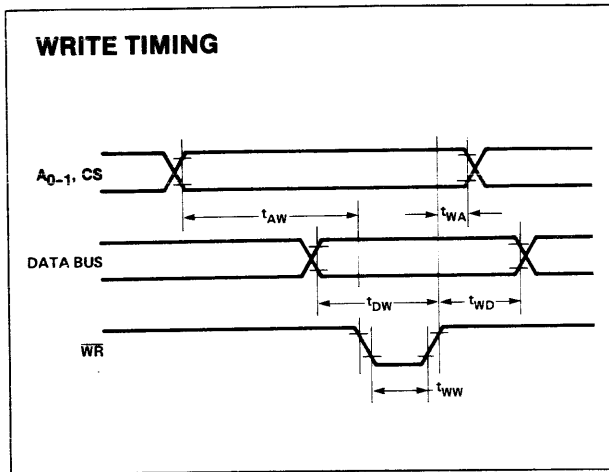
Symbol	Parameter	8253		8253-5		Unit
		Min.	Max.	Min.	Max.	
t_{CLK}	Clock Period	380	dc	380	dc	ns
t_{PWH}	High Pulse Width	230		230		ns
t_{PWL}	Low Pulse Width	150		150		ns
t_{GW}	Gate Width High	150		150		ns
t_{GL}	Gate Width Low	100		100		ns
t_{GS}	Gate Set Up Time to CLK \uparrow	100		100		ns
t_{GH}	Gate Hold Time After CLK \uparrow	50		50		ns
t_{OD}	Output Delay From CLK \downarrow [4]		400		400	ns
t_{ODG}	Output Delay From Gate \downarrow [4]		300		300	ns

NOTES:

- $I_{OL} = 2.2$ mA.
- $I_{OH} = -400$ μ A.
- AC timings measured at $V_{OH} = 2.2$, $V_{OL} = 0.8$.
- $C_L = 150$ pF.

A.C. TESTING INPUT, OUTPUT WAVEFORM

A.C. TESTING LOAD CIRCUIT


WAVEFORMS



Appendix F

8259A Data Sheet
(PIC)

The following material is copyrighted by Intel Corporation. It is reprinted here with the permission of Intel. This data sheet may not be reproduced for any purpose in whole or part without the expressed written consent of Intel.



8259A/8259A-2/8259A-8 PROGRAMMABLE INTERRUPT CONTROLLER

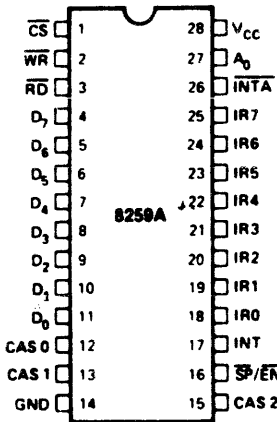
- 8086/8088 Compatible
- Programmable Interrupt Modes
- MCS-80/85™ Compatible
- Individual Request Mask Capability
- Eight-Level Priority Controller
- Single +5V Supply (No Clocks)
- Expandable to 64 Levels
- 28-Pin Dual-In-Line Package

The Intel® 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single +5V supply. Circuitry is static, requiring no clock input.

The 8259A is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements.

The 8259A is fully upward compatible with the Intel® 8259. Software originally written for the 8259 will operate the 8259A in all 8259 equivalent modes (MCS-80/85, Non-Buffered, Edge Triggered).

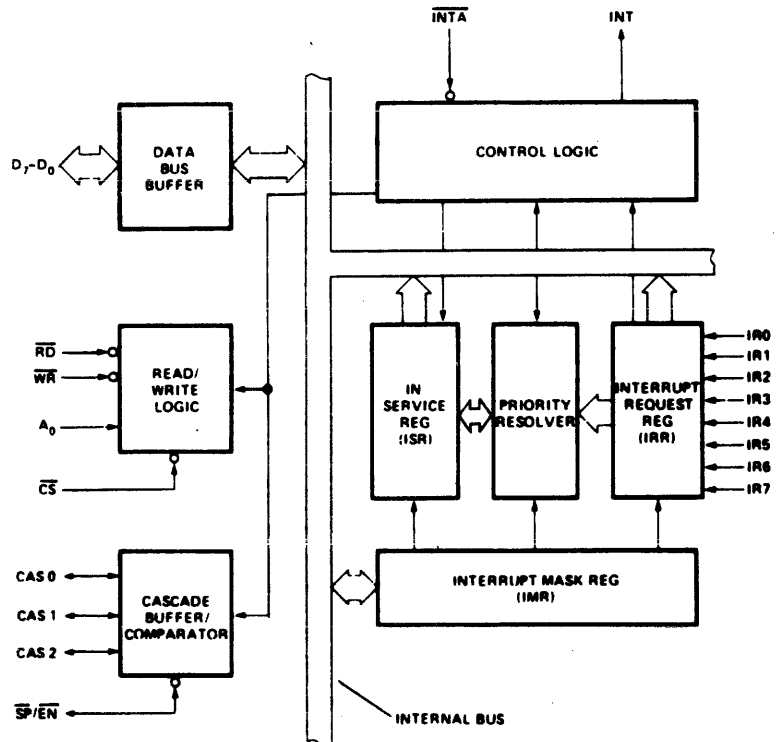
PIN CONFIGURATION



PIN NAMES

D ₇ -D ₀	DATA BUS (BI-DIRECTIONAL)
RD	READ INPUT
WR	WRITE INPUT
A ₀	COMMAND SELECT ADDRESS
CS	CHIP SELECT
CAS2-CAS0	CASCADE LINES
SP/EN	SLAVE PROGRAM/ENABLE BUFFER
INT	INTERRUPT OUTPUT
INTA	INTERRUPT ACKNOWLEDGE INPUT
IR0-IR7	INTERRUPT REQUEST INPUTS

BLOCK DIAGRAM



INTERRUPTS IN MICROCOMPUTER SYSTEMS

Microcomputer system design requires that I/O devices such as keyboards, displays, sensors and other components receive servicing in an efficient manner so that large amounts of the total system tasks can be assumed by the microcomputer with little or no effect on throughput.

The most common method of servicing such devices is the *Polled* approach. This is where the processor must test each device in sequence and in effect "ask" each one if it needs servicing. It is easy to see that a large portion of the main program is looping through this continuous polling cycle and that such a method would have a serious, detrimental effect on system throughput, thus limiting the tasks that could be assumed by the microcomputer and reducing the cost effectiveness of using such devices.

A more desirable method would be one that would allow the microprocessor to be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off.

This method is called *Interrupt*. It is easy to see that system throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness.

The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.

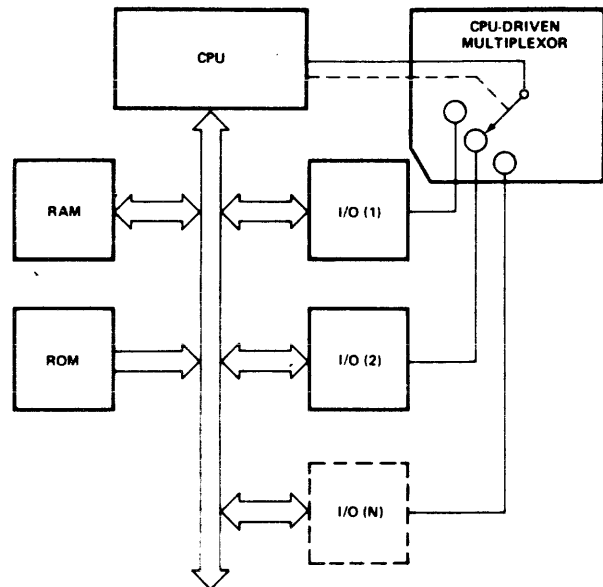
Each peripheral device or structure usually has a special program or "routine" that is associated with its specific functional or operational requirements; this is referred to as a "service routine". The PIC, after issuing an interrupt to the CPU, must somehow input information into the CPU that can "point" the Program Counter to the service routine associated with the requesting device. This "pointer" is an address in a vectoring table and will often be referred to, in this document, as vectoring data.

8259A BASIC FUNCTIONAL DESCRIPTION

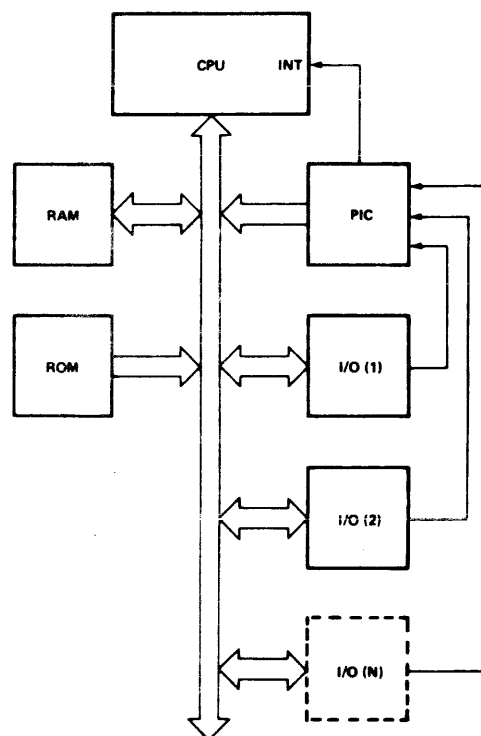
GENERAL

The 8259A is a device specifically designed for use in real time, interrupt driven microcomputer systems. It manages eight levels of requests and has built-in features for expandability to other 8259A's (up to 64 levels). It is programmed by the system's software as an I/O peripheral. A selection of priority modes is available to the programmer so that the manner in which the requests are processed by the 8259A can be configured to

match his system requirements. The priority modes can be changed or reconfigured dynamically at any time during the main program. This means that the complete interrupt structure can be defined as required, based on the total system environment.



Polled Method



Interrupt Method

INTERRUPT REQUEST REGISTER (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

PRIORITY RESOLVER

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during INTA pulse.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower priority.

INT (INTERRUPT)

This output goes directly to the CPU interrupt input. The VOH level on this line is designed to be fully compatible with the 8080A, 8085A, 8086 and 8088.

INTA (INTERRUPT ACKNOWLEDGE)

INTA pulses will cause the 8259A to release vectoring information onto the data bus. The format of this data depends on the system mode (μ PM) of the 8259A.

DATA BUS BUFFER

This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer.

READ/WRITE CONTROL LOGIC

The function of this block is to accept OUTPUT commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the Data Bus.

CS (CHIP SELECT)

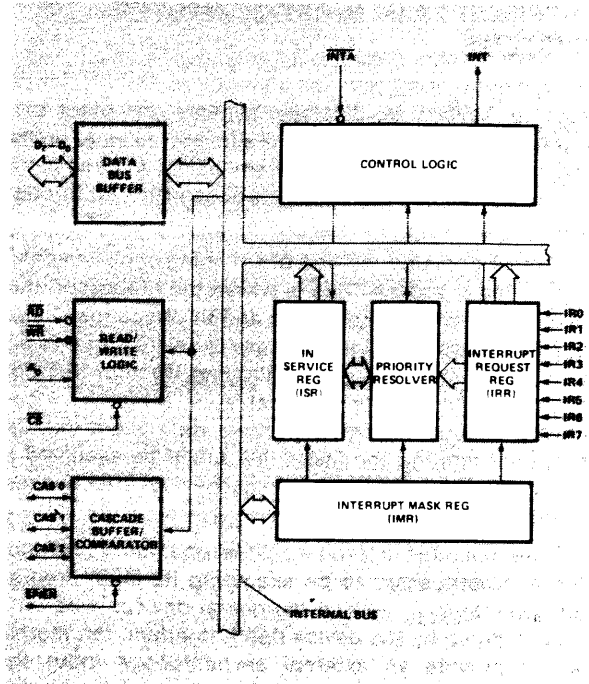
A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.

WR (WRITE)

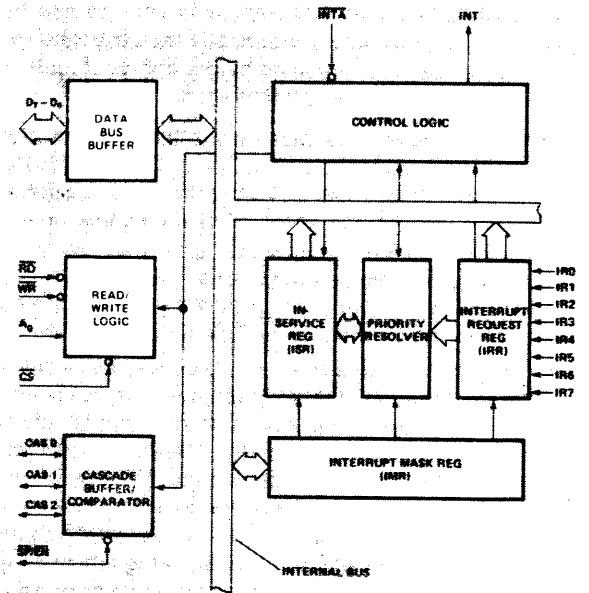
A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.

RD (READ)

A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.



8259A Block Diagram



8259A Block Diagram

A_0

This input signal is used in conjunction with \overline{WR} and \overline{RD} signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

8259A/8259A-2/8259A-8

THE CASCADE BUFFER/COMPARATOR

This function block stores and compares the IDs of all 8259A's used in the system. The associated three I/O pins (CAS0-2) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave. As a master, the 8259A sends the ID of the interrupting slave device onto the CAS0-2 lines. The slave thus selected will send its preprogrammed subroutine address onto the Data Bus during the next one or two consecutive \overline{INTA} pulses. (See section "Cascading the 8259A".)

INTERRUPT SEQUENCE

The powerful features of the 8259A in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used.

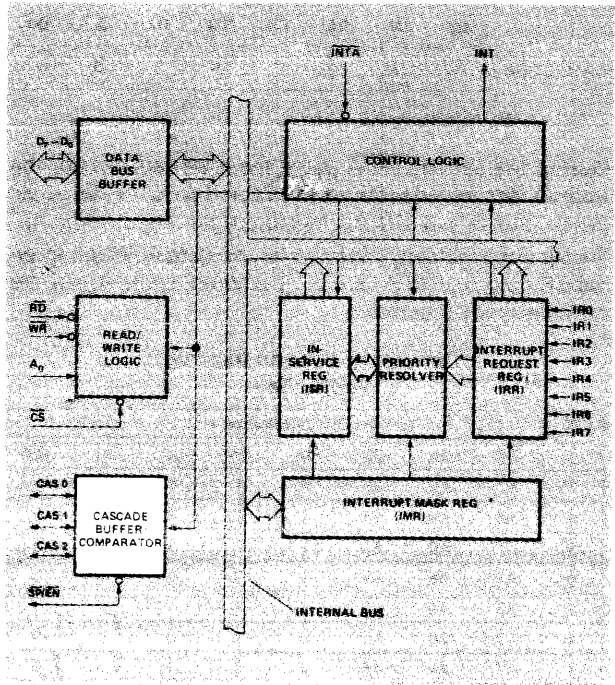
The events occur as follows in an MCS-80/85 system:

1. One or more of the INTERRUPT REQUEST lines (IR7-0) are raised high, setting the corresponding IRR bit(s).
2. The 8259A evaluates these requests, and sends an \overline{INT} to the CPU, if appropriate.
3. The CPU acknowledges the \overline{INT} and responds with an \overline{INTA} pulse.
4. Upon receiving an \overline{INTA} from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7-0 pins.
5. This CALL instruction will initiate two more \overline{INTA} pulses to be sent to the 8259A from the CPU group.
6. These two \overline{INTA} pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first \overline{INTA} pulse and the higher 8-bit address is released at the second \overline{INTA} pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third \overline{INTA} pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

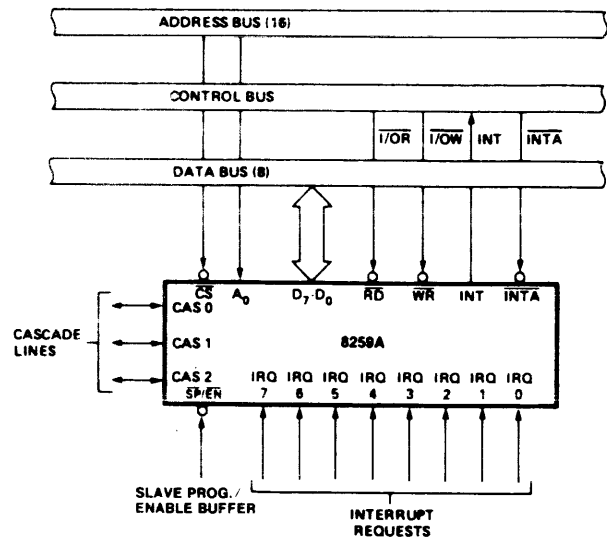
The events occurring in an 8086/8088 system are the same until step 4.

4. Upon receiving an \overline{INTA} from the CPU group, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive the Data Bus during this cycle.
5. The 8086/8088 CPU will initiate a second \overline{INTA} pulse. During this pulse, the 8259A releases an 8-bit pointer onto the Data Bus where it is read by the CPU.
6. This completes the interrupt cycle. In the AEOI mode the ISR bit is reset at the end of the second \overline{INTA} pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine.

If no interrupt request is present at step 4 of either sequence (i.e., the request was too short in duration) the 8259A will issue an interrupt level 7. Both the vectoring bytes and the CAS lines will look like an interrupt level 7 was requested.



8259A Block Diagram



8259A Interface to Standard System Bus

INTERRUPT SEQUENCE OUTPUTS

MCS-80/85 MODE

This sequence is timed by three \overline{INTA} pulses. During the first \overline{INTA} pulse the CALL opcode is enabled onto the data bus.

Content of First Interrupt Vector Byte

	D7	D6	D5	D4	D3	D2	D1	D0
CALL CODE	1	1	0	0	1	1	0	1

During the second \overline{INTA} pulse the lower address of the appropriate service routine is enabled onto the data bus. When Interval = 4 bits A₅-A₇ are programmed, while A₀-A₄ are automatically inserted by the 8259A. When Interval = 8 only A₆ and A₇ are programmed, while A₀-A₅ are automatically inserted.

Content of Second Interrupt Vector Byte

IR	Interval = 4							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	A5	1	1	1	0	0
6	A7	A6	A5	1	1	0	0	0
5	A7	A6	A5	1	0	1	0	0
4	A7	A6	A5	1	0	0	0	0
3	A7	A6	A5	0	1	1	0	0
2	A7	A6	A5	0	1	0	0	0
1	A7	A6	A5	0	0	1	0	0
0	A7	A6	A5	0	0	0	0	0

IR	Interval = 8							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	1	1	1	0	0	0
6	A7	A6	1	1	0	0	0	0
5	A7	A6	1	0	1	0	0	0
4	A7	A6	1	0	0	0	0	0
3	A7	A6	0	1	1	0	0	0
2	A7	A6	0	1	0	0	0	0
1	A7	A6	0	0	1	0	0	0
0	A7	A6	0	0	0	0	0	0

During the third \overline{INTA} pulse the higher address of the appropriate service routine, which was programmed as byte 2 of the initialization sequence (A₈-A₁₅), is enabled onto the bus.

Content of Third Interrupt Vector Byte

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

8086/8088 Mode

8086/8088 mode is similar to MCS80/85 mode except that only two Interrupt Acknowledge cycles are issued by the processor and no CALL opcode is sent to the processor. The first interrupt acknowledge cycle is similar to that of MCS-80/85 systems in that the 8259A uses it to internally freeze the state of the interrupts for priority resolution and as a master it issues the interrupt code on the cascade lines at the end of the \overline{INTA} pulse. On this first cycle it does not issue any data to the processor and leaves its data bus buffers disabled. On the second interrupt acknowledge cycle in 8086/8088 mode the master (or slave if so programmed) will send a byte of data to the processor with the acknowledged interrupt code composed as follows (note the state of the ADI mode control is ignored and A₅-A₁₁ are unused in 8086/8088 mode):

	D7	D6	D5	D4	D3	D2	D1	D0
IR7	T7	T6	T5	T4	T3	1	1	1
IR6	T7	T6	T5	T4	T3	1	1	0
IR5	T7	T6	T5	T4	T3	1	0	1
IR4	T7	T6	T5	T4	T3	1	0	0
IR3	T7	T6	T5	T4	T3	0	1	1
IR2	T7	T6	T5	T4	T3	0	1	0
IR1	T7	T6	T5	T4	T3	0	0	1
IR0	T7	T6	T5	T4	T3	0	0	0

8259A/8259A-2/8259A-8

PROGRAMMING THE 8259A

The 8259A accepts two types of command words generated by the CPU:

1. *Initialization Command Words (ICWs)*: Before normal operation can begin, each 8259A in the system must be brought to a starting point — by a sequence of 2 to 4 bytes timed by \overline{WR} pulses. This sequence is described in Figure 1.
2. *Operation Command Words (OCWs)*: These are the command words that are sent to the 8259A for various forms of operation, such as:

- Interrupt Masking
- End of Interrupt
- Priority Rotation
- Interrupt Status

The OCWs can be written into the 8259A anytime after initialization.

INITIALIZATION

GENERAL

Whenever a command is issued with $A_0=0$ and $D_4=1$, this is interpreted as Initialization Command Word 1 (ICW1). ICW1 starts the initialization sequence during which the following automatically occur.

- a. The edge sense circuit is reset, which means that following initialization, an interrupt request (IR) input must make a low-to-high transition to generate an interrupt.
- b. The Interrupt Mask Register is cleared.
- c. R7 input is assigned priority 7.
- d. The slave mode address is set to 7.
- e. Special Mask Mode is cleared and Status Read is set to IRR.
- f. If $IC4=0$, then all functions selected in ICW4 are set to zero. (Non-Buffered mode*, no Auto-EOI, MCS-80/85 system).

*Note: Master/Slave in ICW4 is only used in the buffered mode.

A_0	D_4	D_3	\overline{RD}	\overline{WR}	\overline{CS}	INPUT OPERATION (READ)
0			0	1	0	IRR, ISR or Interrupting Level → DATA BUS (Note 1)
1			0	1	0	IMR → DATA BUS
						OUTPUT OPERATION (WRITE)
0	0	0	1	0	0	DATA BUS → OCW2
0	0	1	1	0	0	DATA BUS → OCW3
0	1	X	1	0	0	DATA BUS → ICW1
1	X	X	1	0	0	DATA BUS → OCW1, ICW2, ICW3, ICW4 (Note 2)
						DISABLE FUNCTION
X	X	X	1	1	0	DATA BUS — 3-STATE (NO OPERATION)
X	X	X	X	X	1	DATA BUS — 3-STATE (NO OPERATION)

Notes: 1. Selection of IRR, ISR or Interrupting Level is based on the content of OCW3 written before the READ operation.

2. On-chip sequencer logic queues these commands into proper sequence.

8259A Basic Operation

INITIALIZATION COMMAND WORDS 1 AND 2 (ICW1, ICW2)

A₅-A₁₅: *Page starting address of service routines.* In an MCS 80/85 system, the 8 request levels will generate CALLs to 8 locations equally spaced in memory. These can be programmed to be spaced at intervals of 4 or 8 memory locations, thus the 8 routines will occupy a page of 32 or 64 bytes, respectively.

The address format is 2 bytes long (A₀-A₁₅). When the routine interval is 4, A₀-A₄ are automatically inserted by the 8259A, while A₅-A₁₅ are programmed externally. When the routine interval is 8, A₀-A₅ are automatically inserted by the 8259A, while A₆-A₁₅ are programmed externally.

The 8-byte interval will maintain compatibility with current software, while the 4-byte interval is best for a compact jump table.

In an MCS-86 system T₇-T₃ are inserted in the five most significant bits of the vectoring byte and the 8259A sets the three least significant bits according to the interrupt level. A₁₀-A₅ are ignored and ADI (Address Interval) has no effect.

LTIM: If LTIM = 1, then the 8259A will operate in the level interrupt mode. Edge detect logic on the interrupt inputs will be disabled.

ADI: CALL address interval. ADI = 1 then interval = 4; ADI = 0 then interval = 8.

SNGL: Single. Means that this is the only 8259A in the system. If SNGL = 1 no ICW3 will be issued.

IC4: If this bit is set — ICW4 has to be read. If ICW4 is not needed, set IC4 = 0.

INITIALIZATION COMMAND WORD 3 (ICW3)

This word is read only when there is more than one 8259A in the system and cascading is used, in which case SNGL = 0. It will load the 8-bit slave register. The functions of this register are:

- a. In the master mode (either when $\overline{SP} = 1$, or in buffered mode when M/S = 1 in ICW4) a "1" is set for each slave in the system. The master then will release byte 1 of the call sequence (for MCS-80/85 system) and will enable the corresponding slave to release bytes 2 and 3 (for 8086/8088 only byte 2) through the cascade lines.
- b. In the slave mode (either when $\overline{SP} = 0$, or if BUF = 1 and M/S = 0 in ICW4) bits 2-0 identify the slave. The slave compares its cascade input with these bits and if they are equal, bytes 2 and 3 of the call sequence (or just byte 2 for 8086/8088) are released by it on the Data Bus.

INITIALIZATION COMMAND WORD 4 (ICW4)

SFNM: If SFNM = 1 the special fully nested mode is programmed.

BUF: If BUF = 1 the buffered mode is programmed. In buffered mode $\overline{SP}/\overline{EN}$ becomes an enable output and the master/slave determination is by M/S.

M/S: If buffered mode is selected: M/S = 1 means the 8259A is programmed to be a master, M/S = 0 means the 8259A is programmed to be a slave. If BUF = 0, M/S has no function.

AEOI: If AEOI = 1 the automatic end of interrupt mode is programmed.

μPM: Microprocessor mode: μPM = 0 sets the 8259A for MCS-80/85 system operation, μPM = 1 sets the 8259A for MCS-86 system operation.

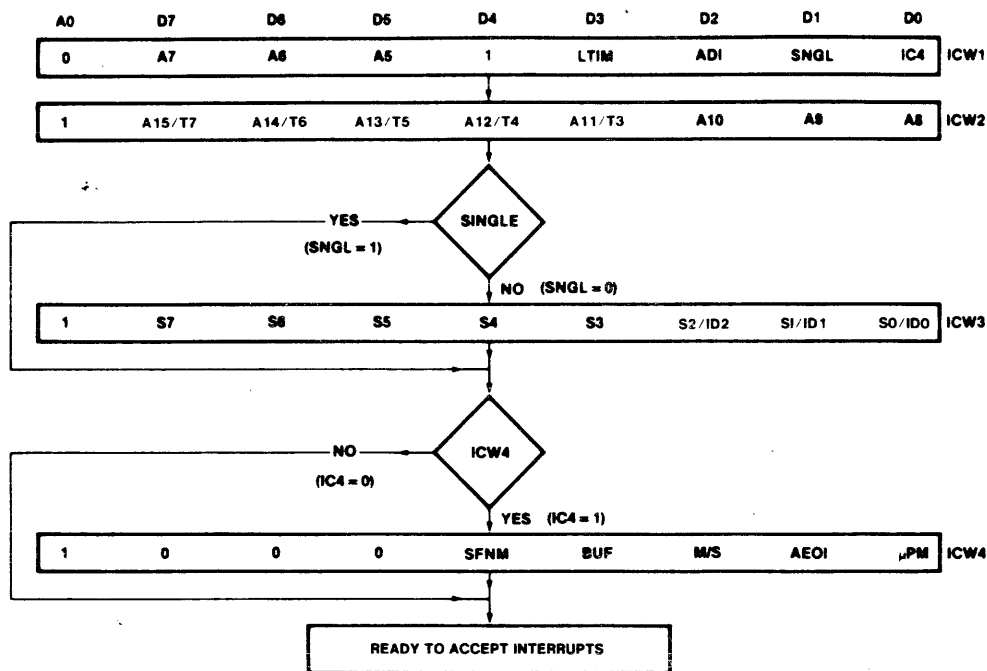
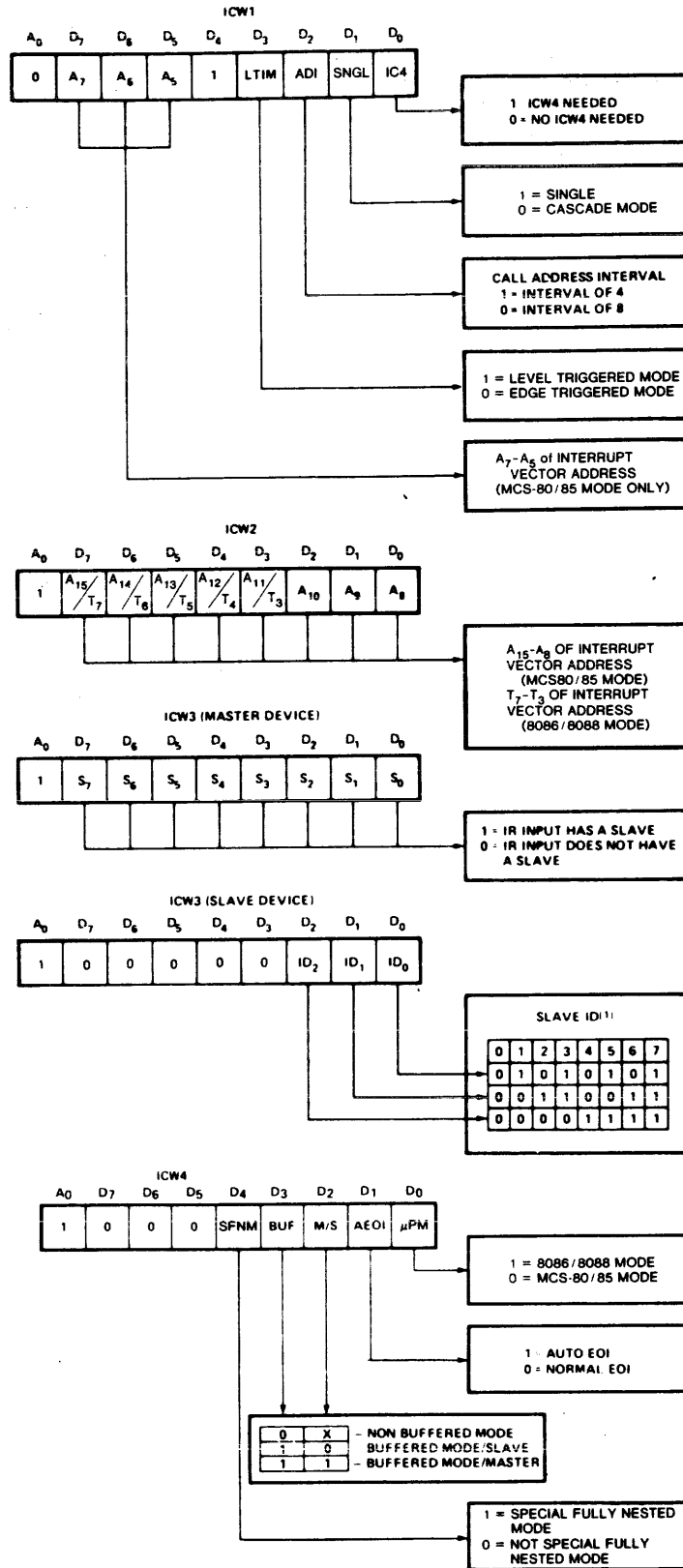


Figure 1. Initialization Sequence

8259A/8259A-2/8259A-8



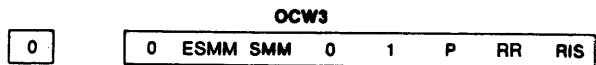
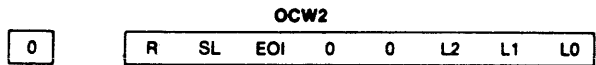
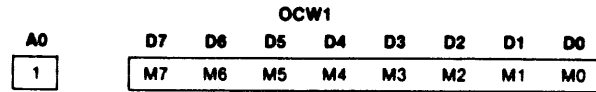
NOTE 1: SLAVE ID IS EQUAL TO THE CORRESPONDING MASTER IR INPUT.

Initialization Command Word Format

OPERATION COMMAND WORDS (OCWs)

After the Initialization Command Words (ICWs) are programmed into the 8259A, the chip is ready to accept Interrupt requests at its input lines. However, during the 8259A operation, a selection of algorithms can command the 8259A to operate in various modes through the Operation Command Words (OCWs).

OPERATION CONTROL WORDS (OCWs)



OPERATION CONTROL WORD 1 (OCW1)

OCW1 sets and clears the mask bits in the interrupt Mask Register (IMR). M₇–M₀ represent the eight mask bits. M = 1 indicates the channel is masked (inhibited), M = 0 indicates the channel is enabled.

OPERATION CONTROL WORD 2 (OCW2)

R, SL, EOI — These three bits control the Rotate and End if Interrupt modes and combinations of the two. A chart of these combinations can be found on the Operation Command Word Format.

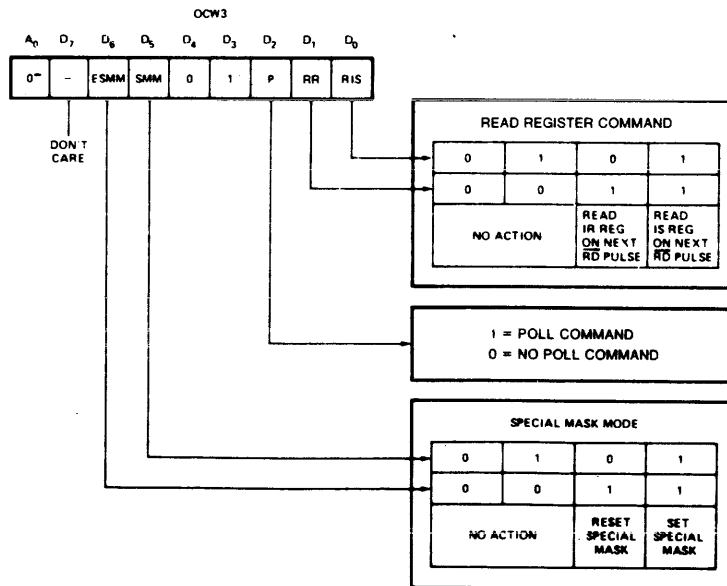
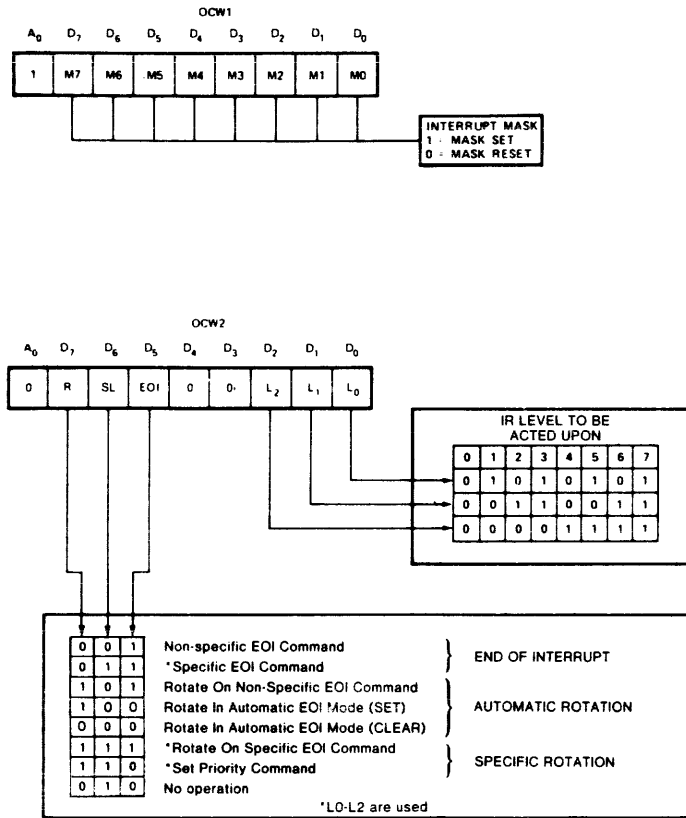
L₂, L₁, L₀ — These bits determine the interrupt level acted upon when the SEOI bit is active.

OPERATION CONTROL WORD 3 (OCW3)

ESMM — Enable Special Mask Mode. When this bit is set to 1 it enables the SMM bit to set or reset the Special Mask Mode. When ESMM = 0 the SMM bit becomes a "don't care".

SMM — Special Mask Mode. If ESMM = 1 and SMM = 1 the 8259A will enter Special Mask Mode. If ESMM = 1 and SMM = 0 the 8259A will revert to normal mask mode. When ESMM = 0, SMM has no effect.

8259A/8259A-2/8259A-8



Operation Command Word Format

INTERRUPT MASKS

Each Interrupt Request input can be masked individually by the Interrupt Mask Register (IMR) programmed through OCW1. Each bit in the IMR masks one interrupt channel if it is set (1). Bit 0 masks IR0, Bit 1 masks IR1 and so forth. Masking an IR channel does not affect the other channels operation.

SPECIAL MASK MODE

Some applications may require an interrupt service routine to dynamically alter the system priority structure during its execution under software control. For example, the routine may wish to inhibit lower priority requests for a portion of its execution but enable some of them for another portion.

The difficulty here is that if an Interrupt Request is acknowledged and an End of Interrupt command did not reset its IS bit (i.e., while executing a service routine), the 8259A would have inhibited all lower priority requests with no easy way for the routine to enable them

That is where the Special Mask Mode comes in. In the special Mask Mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level *and enables* interrupts from *all other* levels (lower as well as higher) that are not masked.

Thus, any interrupts may be selectively enabled by loading the mask register.

The special Mask Mode is set by OCW3 where: SMM = 1, SMM = 1, and cleared where SMM = 1, SMM = 0.

BUFFERED MODE

When the 8259A is used in a large system where bus driving buffers are required on the data bus and the cascading mode is used, there exists the problem of enabling buffers.

The buffered mode will structure the 8259A to send an enable signal on $\overline{SP/EN}$ to enable the buffers. In this mode, whenever the 8259A's data bus outputs are enabled, the $\overline{SP/EN}$ output becomes active.

This modification forces the use of software programming to determine whether the 8259A is a master or a slave. Bit 3 in ICW4 programs the buffered mode, and bit 2 in ICW4 determines whether it is a master or a slave.

FULLY NESTED MODE

This mode is entered after initialization unless another mode is programmed. The interrupt requests are ordered in priority form 0 through 7 (0 highest). When an interrupt is acknowledged the highest priority request is determined and its vector placed on the bus. Additionally, a bit of the Interrupt Service register (ISO-7) is set. This bit remains set until the microprocessor issues an End of Interrupt (EOI) command immediately before returning from the service routine, or if AEOI (Automatic End of Interrupt) bit is set, until the trailing edge of the last INTA. While the IS bit is set, all further interrupts of the same or lower priority are inhibited, while higher levels will generate an interrupt (which will be acknowledged only if the microprocessor internal Interrupt enable flip-flop has been re-enabled through software).

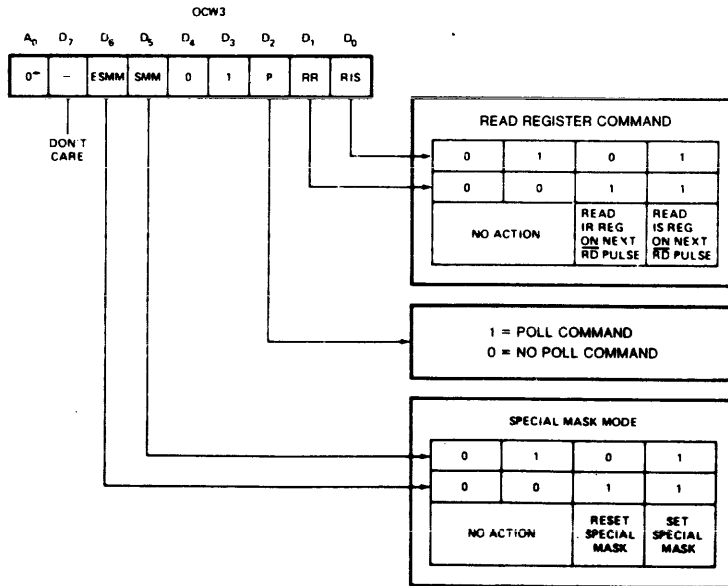
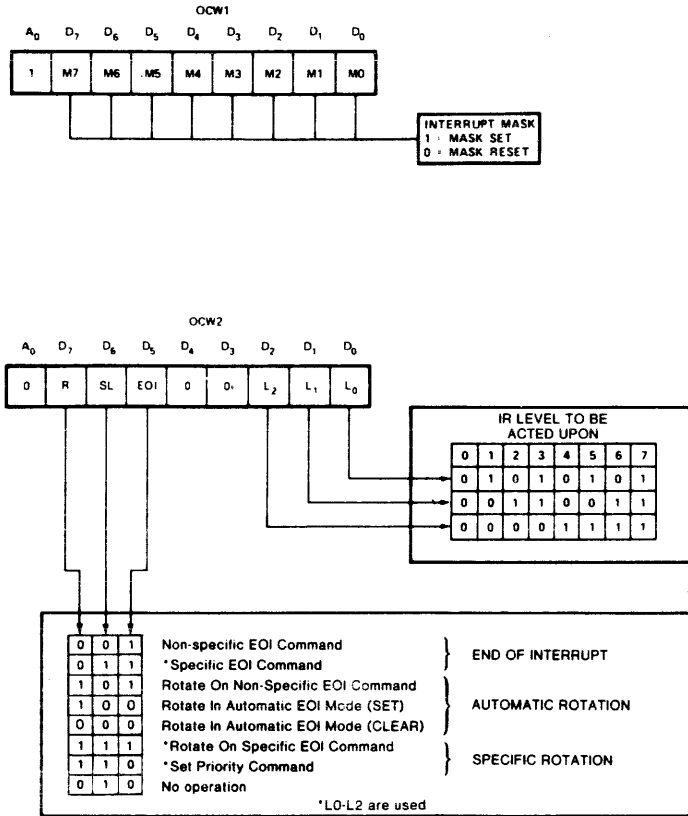
After the initialization sequence, IR0 has the highest priority and IR7 the lowest. Priorities can be changed, as will be explained, by priority rotation.

THE SPECIAL FULLY NESTED MODE

This mode will be used in the case of a big system where cascading is used, and the priority has to be conserved within each slave. In this case the special fully nested mode will be programmed to the master (using ICW4). This mode is similar to the normal fully nested mode with the following exceptions:

- a. When an interrupt request from a certain slave is in service this slave is not locked out from the master's priority logic and further interrupt requests from higher priority IR's within the slave will be recognized by the master and will initiate interrupts to the processor. (In the normal nested mode a slave is masked out when its request is in service and no higher requests from the same slave can be serviced.)
- b. When exiting the Interrupt Service routine the software has to check whether the interrupt serviced was the only one from that slave. This is done by sending a non-specific End of Interrupt (EOI) command to the slave and then reading its In-Service register and checking for zero. If it is empty, a non-specific EOI can be sent to the master too. If not, no EOI should be sent.

8259A/8259A-2/8259A-8



Operation Command Word Format

INTERRUPT MASKS

Each Interrupt Request input can be masked individually by the Interrupt Mask Register (IMR) programmed through OCW1. Each bit in the IMR masks one interrupt channel if it is set (1). Bit 0 masks IR0, Bit 1 masks IR1 and so forth. Masking an IR channel does not affect the other channels operation.

SPECIAL MASK MODE

Some applications may require an interrupt service routine to dynamically alter the system priority structure during its execution under software control. For example, the routine may wish to inhibit lower priority requests for a portion of its execution but enable some of them for another portion.

The difficulty here is that if an Interrupt Request is acknowledged and an End of Interrupt command did not reset its IS bit (i.e., while executing a service routine), the 8259A would have inhibited all lower priority requests with no easy way for the routine to enable them.

That is where the Special Mask Mode comes in. In the special Mask Mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level *and enables* interrupts from *all other* levels (lower as well as higher) that are not masked.

Thus, any interrupts may be selectively enabled by loading the mask register.

The special Mask Mode is set by OCW3 where: SMM = 1, SMM = 1, and cleared where SMM = 1, SMM = 0.

BUFFERED MODE

When the 8259A is used in a large system where bus driving buffers are required on the data bus and the cascading mode is used, there exists the problem of enabling buffers.

The buffered mode will structure the 8259A to send an enable signal on $\overline{SP/EN}$ to enable the buffers. In this mode, whenever the 8259A's data bus outputs are enabled, the $\overline{SP/EN}$ output becomes active.

This modification forces the use of software programming to determine whether the 8259A is a master or a slave. Bit 3 in ICW4 programs the buffered mode, and bit 2 in ICW4 determines whether it is a master or a slave.

FULLY NESTED MODE

This mode is entered after initialization unless another mode is programmed. The interrupt requests are ordered in priority form 0 through 7 (0 highest). When an interrupt is acknowledged the highest priority request is determined and its vector placed on the bus. Additionally, a bit of the Interrupt Service register (ISO-7) is set. This bit remains set until the microprocessor issues an End of Interrupt (EOI) command immediately before returning from the service routine, or if AEOI (Automatic End of Interrupt) bit is set, until the trailing edge of the last INTA. While the IS bit is set, all further interrupts of the same or lower priority are inhibited, while higher levels will generate an interrupt (which will be acknowledged only if the microprocessor internal Interrupt enable flip-flop has been re-enabled through software).

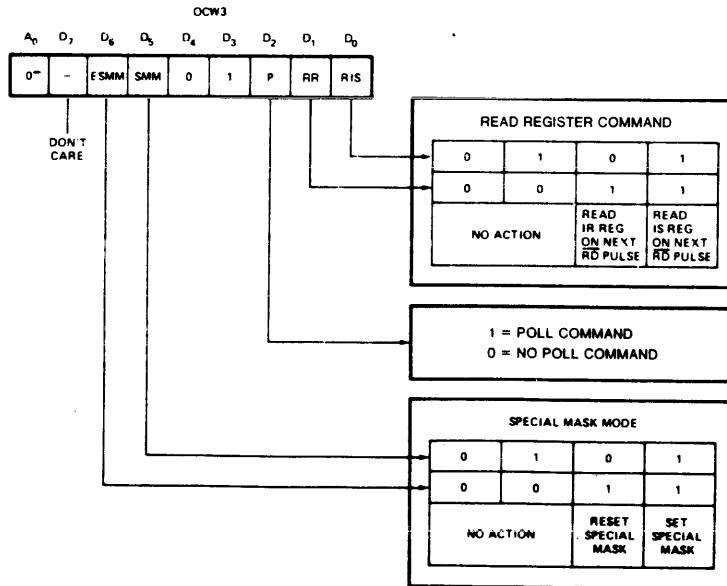
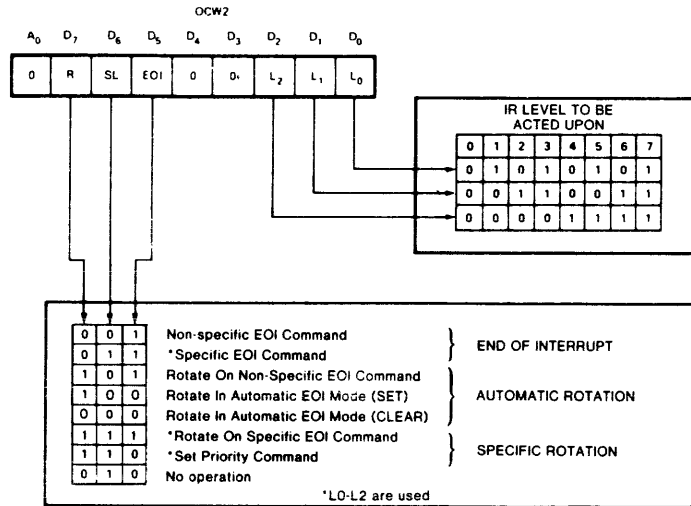
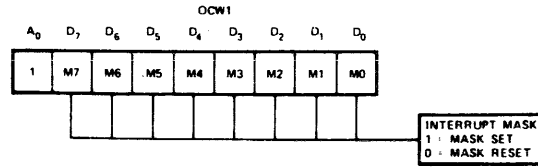
After the initialization sequence, IR0 has the highest priority and IR7 the lowest. Priorities can be changed, as will be explained, by priority rotation.

THE SPECIAL FULLY NESTED MODE

This mode will be used in the case of a big system where cascading is used, and the priority has to be conserved within each slave. In this case the special fully nested mode will be programmed to the master (using ICW4). This mode is similar to the normal fully nested mode with the following exceptions:

- a. When an interrupt request from a certain slave is in service this slave is not locked out from the master's priority logic and further interrupt requests from higher priority IR's within the slave will be recognized by the master and will initiate interrupts to the processor. (In the normal nested mode a slave is masked out when its request is in service and no higher requests from the same slave can be serviced.)
- b. When exiting the Interrupt Service routine the software has to check whether the interrupt serviced was the only one from that slave. This is done by sending a non-specific End of Interrupt (EOI) command to the slave and then reading its In-Service register and checking for zero. If it is empty, a non-specific EOI can be sent to the master too. If not, no EOI should be sent.

8259A/8259A-2/8259A-8



Operation Command Word Format

INTERRUPT MASKS

Each Interrupt Request input can be masked individually by the Interrupt Mask Register (IMR) programmed through OCW1. Each bit in the IMR masks one interrupt channel if it is set (1). Bit 0 masks IR0, Bit 1 masks IR1 and so forth. Masking an IR channel does not affect the other channels operation.

SPECIAL MASK MODE

Some applications may require an interrupt service routine to dynamically alter the system priority structure during its execution under software control. For example, the routine may wish to inhibit lower priority requests for a portion of its execution but enable some of them for another portion.

The difficulty here is that if an Interrupt Request is acknowledged and an End of Interrupt command did not reset its IS bit (i.e., while executing a service routine), the 8259A would have inhibited all lower priority requests with no easy way for the routine to enable them

That is where the Special Mask Mode comes in. In the special Mask Mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level *and enables* interrupts from *all other* levels (lower as well as higher) that are not masked.

Thus, any interrupts may be selectively enabled by loading the mask register.

The special Mask Mode is set by OCW3 where: SMM = 1, SMM = 1, and cleared where SMM = 1, SMM = 0.

BUFFERED MODE

When the 8259A is used in a large system where bus driving buffers are required on the data bus and the cascading mode is used, there exists the problem of enabling buffers.

The buffered mode will structure the 8259A to send an enable signal on $\overline{SP/EN}$ to enable the buffers. In this mode, whenever the 8259A's data bus outputs are enabled, the $\overline{SP/EN}$ output becomes active.

This modification forces the use of software programming to determine whether the 8259A is a master or a slave. Bit 3 in ICW4 programs the buffered mode, and bit 2 in ICW4 determines whether it is a master or a slave.

FULLY NESTED MODE

This mode is entered after initialization unless another mode is programmed. The interrupt requests are ordered in priority form 0 through 7 (0 highest). When an interrupt is acknowledged the highest priority request is determined and its vector placed on the bus. Additionally, a bit of the Interrupt Service register (ISO-7) is set. This bit remains set until the microprocessor issues an End of Interrupt (EOI) command immediately before returning from the service routine, or if AEOI (Automatic End of Interrupt) bit is set, until the trailing edge of the last INTA. While the IS bit is set, all further interrupts of the same or lower priority are inhibited, while higher levels will generate an interrupt (which will be acknowledged only if the microprocessor internal Interrupt enable flip-flop has been re-enabled through software).

After the initialization sequence, IR0 has the highest priority and IR7 the lowest. Priorities can be changed, as will be explained, by priority rotation.

THE SPECIAL FULLY NESTED MODE

This mode will be used in the case of a big system where cascading is used, and the priority has to be conserved within each slave. In this case the special fully nested mode will be programmed to the master (using ICW4). This mode is similar to the normal fully nested mode with the following exceptions:

- a. When an interrupt request from a certain slave is in service this slave is not locked out from the master's priority logic and further interrupt requests from higher priority IR's within the slave will be recognized by the master and will initiate interrupts to the processor. (In the normal nested mode a slave is masked out when its request is in service and no higher requests from the same slave can be serviced.)
- b. When exiting the Interrupt Service routine the software has to check whether the interrupt serviced was the only one from that slave. This is done by sending a non-specific End of Interrupt (EOI) command to the slave and then reading its In-Service register and checking for zero. If it is empty, a non-specific EOI can be sent to the master too. If not, no EOI should be sent.

POLL

In this mode the microprocessor internal Interrupt Enable flip-flop is reset, disabling its interrupt input. Service to devices is achieved by programmer initiative using a Poll command.

The Poll command is issued by setting P = "1" in OCW3. The 8259A treats the next \overline{RD} pulse to the 8259A (i.e., $\overline{RD} = 0$, $\overline{CS} = 0$) as an interrupt acknowledge, sets the appropriate IS bit if there is a request, and reads the priority level. Interrupt is frozen from \overline{WR} to \overline{RD} .

The word enabled onto the data bus during \overline{RD} is:

D7	D6	D5	D4	D3	D2	D1	D0
1	-	-	-	-	W2	W1	W0

W0-W2: Binary code of the highest priority level requesting service.

I: Equal to a "1" if there is an interrupt.

This mode is useful if there is a routine command common to several levels so that the INTA sequence is not needed (saves ROM space). Another application is to use the poll command to expand the number of priority levels to more than 64.

END OF INTERRUPT (EOI)

The In Service (IS) bit can be reset either automatically following the trailing edge of the last in sequence INTA pulse (when AEOI bit in ICW1 is set) or by a command word that must be issued to the 8259A before returning from a service routine (EOI command). An EOI command must be issued twice, once for the master and once for the corresponding slave if slaves are in use.

There are two forms of EOI command: Specific and Non-Specific. When the 8259A is operated in modes which preserve the fully nested structure, it can determine which IS bit to reset on EOI. When a Non-Specific EOI command is issued the 8259A will automatically reset the highest IS bit of those that are set, since in the nested mode the highest IS level was necessarily the last level acknowledged and serviced.

However, when a mode is used which may disturb the fully nested structure, the 8259A may no longer be able to determine the last level acknowledged. In this case a Specific End of Interrupt (SEOI) must be issued which includes as part of the command the IS level to be reset. EOI is issued whenever $EOI = 1$, in OCW2, where L0-L2 is the binary level of the IS bit to be reset. Note that although the Rotate command can be issued together with an EOI where $EOI = 1$, it is not necessarily tied to it.

It should be noted that an IS bit that is masked by an IMR bit will not be cleared by a non-specific EOI if the 8259A is in the Special Mask Mode.

AUTOMATIC END OF INTERRUPT (AEOI) MODE

If $AEOI = 1$ in ICW4, then the 8259A will operate in AEOI mode continuously until reprogrammed by ICW4. In this mode the 8259A will automatically perform a non-specific EOI operation at the trailing edge of the last interrupt acknowledge pulse (third pulse in MCS-80/85,

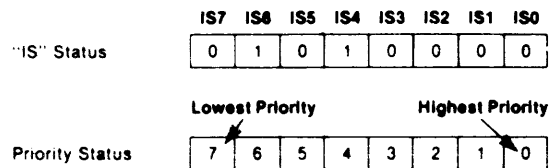
second in MCS-86). Note that from a system standpoint, this mode should be used only when a nested multilevel interrupt structure is not required within a single 8259A.

To achieve automatic rotation within AEOI, there is a special rotate flip-flop. It is set by OCW2 with $R = 1$, $SL = 0$. $EOI = 0$, and cleared with $R = 0$, $SEOI = 0$, $EOI = 0$.

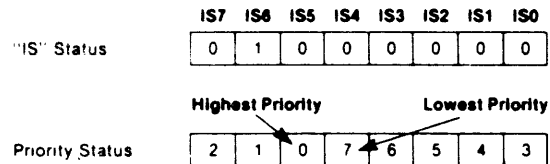
AUTOMATIC ROTATION (Equal Priority Devices)

In some applications there are a number of interrupting devices of equal priority. In this mode a device, after being serviced, receives the lowest priority, so a device requesting an interrupt will have to wait, in the worst case until each of 7 other devices are serviced at most once. For example, if the priority and "in service" status is:

Before Rotate (IR4 the highest priority requiring service)



After Rotate (IR4 was serviced, all other priorities rotated correspondingly)



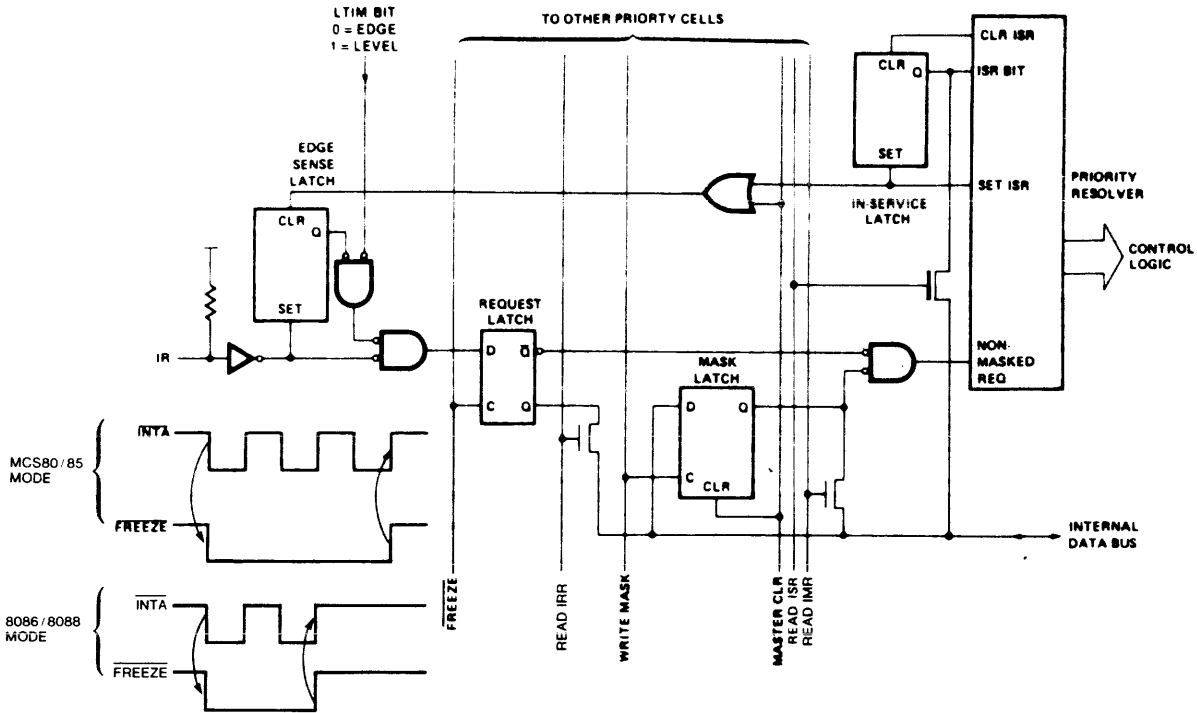
There are two ways to accomplish Automatic Rotation using OCW2, the Rotate on Non-Specific EOI Command ($R = 1$, $SL = 0$, $EOI = 1$) and the Rotate in Automatic EOI Mode which is set by ($R = 1$, $SL = 0$, $EOI = 0$) and cleared by ($R = 0$, $SL = 0$, $EOI = 0$).

SPECIFIC ROTATION (Specific Priority)

The programmer can change priorities by programming the bottom priority and thus fixing all other priorities; i.e., if IR5 is programmed as the bottom priority device, then IR6 will have the highest one.

The Set Priority command is issued in OCW2 where: $R = 1$, $SEOI = 1$; L0-L2 is the binary priority level code of the bottom priority device.

Observe that in this mode internal status is updated by software control during OCW2. However, it is independent of the End of Interrupt (EOI) command (also executed by OCW2). Priority changes can be executed during an EOI command by using the Rotate on Specific EOI Command in OCW2 ($R = 1$, $SL = 1$, $EOI = 1$ and L0-L2 = IR level to receive bottom priority).



- NOTES
1. MASTER CLEAR ACTIVE ONLY DURING ICW1
 2. FREEZE/IS ACTIVE DURING \overline{INTA} / AND POLL SEQUENCES ONLY
 3. TRUTH TABLE FOR D-LATCH

C	D	Q	OPERATION
1	D_i	D_i	FOLLOW
0	X	Q_{n-1}	HOLD

Priority Cell — Simplified Logic Diagram

LEVEL TRIGGERED MODE

This mode is programmed using bit 3 in ICW1.

If $LTIM = '1'$, an interrupt request will be recognized by a 'high' level on IR Input, and there is no need for an edge detection. The interrupt request must be removed before the EOI command is issued or the CPU interrupt is enabled to prevent a second interrupt from occurring.

The above figure shows a conceptual circuit to give the reader an understanding of the level sensitive and edge sensitive input circuitry of the 8259A. Be sure to note that the request latch is a transparent D type latch.

READING THE 8259A STATUS

The input status of several internal registers can be read to update the user information on the system. The following registers can be read via OCW3 (IRR and ISR or OCW1 (IMR).

Interrupt Request Register (IRR): 8-bit register which contains the levels requesting an interrupt to be acknowledged. The highest request level is reset from the IRR when an interrupt is acknowledged. (Not affected by IMR).

In-Service Register (ISR): 8-bit register which contains the priority levels that are being serviced. The ISR is updated when an End of Interrupt command is issued.

Interrupt Mask Register: 8-bit register which contains the interrupt request lines which are masked.

The IRR can be read when, prior to the \overline{RD} pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 0).

The ISR can be read when, prior to the \overline{RD} pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 1).

There is no need to write an OCW3 before every status read operation, as long as the status read corresponds with the previous one; i.e., the 8259A "remembers" whether the IRR or ISR has been previously selected by the OCW3. This is not true when poll is used.

After initialization the 8259A is set to IRR.

For reading the IMR, no OCW3 is needed. The output data bus will contain the IMR whenever \overline{RD} is active and $A0 = 1$ (OCW1).

Polling overrides status read when $P = 1$, $RR = 1$ in OCW3.

8259A/8259A-2/8259A-8

SUMMARY OF 8259A INSTRUCTION SET

Inst. #	Mnemonic	A0	D7	D6	D5	D4	D3	D2	D1	D0	Operation Description
1	ICW1 A	0	A7	A6	A5	1	0	1	1	0	Format = 4, single, edge triggered Format = 4, single, level triggered } Byte 1 Initialization Format = 4, not single, edge triggered Format = 4, not single, level triggered } No ICW4 Required Format = 8, single, edge triggered Format = 8, single, level triggered Format = 8, not single, edge triggered Format = 8, not single, level triggered
2	ICW1 B	0	A7	A6	A5	1	1	1	1	0	
3	ICW1 C	0	A7	A6	A5	1	0	1	0	0	
4	ICW1 D	0	A7	A6	A5	1	1	1	0	0	
5	ICW1 E	0	A7	A6	0	1	0	0	1	0	
6	ICW1 F	0	A7	A6	0	1	1	0	1	0	
7	ICW1 G	0	A7	A6	0	1	0	0	0	0	
8	ICW1 H	0	A7	A6	0	1	1	0	0	0	
9	ICW1 I	0	A7	A6	A5	1	0	1	1	1	Format = 4, single, edge triggered Format = 4, single, level triggered } Byte 1 Initialization Format = 4, not single, edge triggered Format = 4, not single, level triggered } ICW4 Required Format = 8, single, edge triggered Format = 8, single, level triggered Format = 8, not single, edge triggered Format = 8, not single, level triggered
10	ICW1 J	0	A7	A6	A5	1	1	1	1	1	
11	ICW1 K	0	A7	A6	A5	1	0	1	0	1	
12	ICW1 L	0	A7	A6	A5	1	1	1	0	1	
13	ICW1 M	0	A7	A6	0	1	0	0	1	1	
14	ICW1 N	0	A7	A6	0	1	1	0	1	1	
15	ICW1 O	0	A7	A6	0	1	0	0	0	1	
16	ICW1 P	0	A7	A6	0	1	1	0	0	1	
17	ICW2	1	A15	A14	A13	A12	A11	A10	A9	A8	Byte 2 initialization
18	ICW3 M	1	S7	S6	S5	S4	S3	S2	S1	S0	Byte 3 initialization — master
19	ICW3 S	1	0	0	0	0	0	S2	S1	S0	Byte 3 initialization — slave
20	ICW4 A	1	0	0	0	0	0	0	0	0	No action, redundant
21	ICW4 B	1	0	0	0	0	0	0	0	1	Non-buffered mode, no AEOI, 8086/8088
22	ICW4 C	1	0	0	0	0	0	0	1	0	Non-buffered mode, AEOI, MCS-80/85
23	ICW4 D	1	0	0	0	0	0	0	0	1	Non-buffered mode, AEOI, 8086/8088
24	ICW4 E	1	0	0	0	0	0	1	0	0	No action, redundant
25	ICW4 F	1	0	0	0	0	0	1	0	1	Non-buffered mode, no AEOI, 8086/8088
26	ICW4 G	1	0	0	0	0	0	1	1	0	Non-buffered mode, AEOI, MCS-80/85
27	ICW4 H	1	0	0	0	0	0	1	1	1	Non-buffered mode, AEOI, 8086/8088
28	ICW4 I	1	0	0	0	0	1	0	0	0	Buffered mode, slave, no AEOI, MCS-80/85
29	ICW4 J	1	0	0	0	0	1	0	0	1	Buffered mode, slave, no AEOI, 8086/8088
30	ICW4 K	1	0	0	0	0	1	0	1	0	Buffered mode, slave, AEOI, MCS-80/85
31	ICW4 L	1	0	0	0	0	1	0	1	1	Buffered mode, slave, AEOI, 8086/8088
32	ICW4 M	1	0	0	0	0	1	1	0	0	Buffered mode, master, no AEOI, MCS-80/85
33	ICW4 N	1	0	0	0	0	1	1	0	1	Buffered mode, master, no AEOI, 8086/8088
34	ICW4 O	1	0	0	0	0	1	1	1	0	Buffered mode, master, AEOI, MCS-80/85
35	ICW4 P	1	0	0	0	0	1	1	1	1	Buffered mode, master AEOI, 8086, 8088
36	ICW4 NA	1	0	0	0	1	0	0	0	0	Fully nested mode, MCS-80, non buffered, no AEOI } ICW4 NB through ICW4 ND are identical to ICW4 B through ICW4 D with the addition of Fully Nested Mode } Fully Nested Mode, MCS-80/85, non-buffered, no AEOI
37	ICW4 NB	1	0	0	0	1	0	0	0	1	
38	ICW4 NC	1	0	0	0	1	0	0	1	0	
39	ICW4 ND	1	0	0	0	1	0	0	1	1	
40	ICW4 NE	1	0	0	0	1	0	1	0	0	
41	ICW4 NF	1	0	0	0	1	0	1	0	1	
42	ICW4 NG	1	0	0	0	1	0	1	1	0	
43	ICW4 NH	1	0	0	0	1	0	1	1	1	
44	ICW4 NI	1	0	0	0	1	1	0	0	0	
45	ICW4 NJ	1	0	0	0	1	1	0	0	1	ICW4 NF through ICW4 NP are identical to ICW4 F through ICW4 P with the addition of Fully Nested Mode
46	ICW4 NK	1	0	0	0	1	1	1	0	1	
47	ICW4 NL	1	0	0	0	1	1	0	1	1	
48	ICW4 NM	1	0	0	0	1	1	1	0	0	
49	ICW4 NN	1	0	0	0	1	1	1	0	1	
50	ICW4 NO	1	0	0	0	1	1	1	1	0	
51	ICW4 NP	1	0	0	0	1	1	1	1	1	
52	OCW1	1	M7	M6	M5	M4	M3	M2	M1	M0	
53	OCW2 E	0	0	0	1	0	0	0	0	0	Non-specific EOI
54	OCW2 SE	0	0	1	1	0	0	L2	L1	L0	Specific EOI, L0-L2 code of IS FF to be reset
55	OCW2 RE	0	1	0	1	0	0	0	0	0	Rotate on Non-Specific EOI
56	OCW2 RSE	0	1	1	1	0	0	L2	L1	L0	Rotate on Specific EOI L0-L2 code of line
57	OCW2 R	0	1	0	0	0	0	0	0	0	Rotate in Auto EOI (set)
58	OCW2 CR	0	0	0	0	0	0	0	0	0	Rotate in Auto EOI (clear)
59	OCW2 RS	0	1	1	0	0	0	L2	L1	L0	Set Priority Command
60	OCW3 P	0	0	0	0	0	1	1	0	0	Poll mode
61	OCW3 RIS	0	0	0	0	0	1	0	1	1	Read IS register

8259A/8259A-2/8259A-8

SUMMARY OF 8259A INSTRUCTION SET (Cont.)

Inst. #	Mnemonic	A0	D7	D6	D5	D4	D3	D2	D1	D0	Operation Description	
46	OCW3 RR	0	0	0	0	0	0	1	0	1	0	Read request register
47	OCW3 SM	0	0	1	1	0	1	0	0	0	0	Set special mask mode
48	OCW3 RSM	0	0	1	0	0	1	0	0	0	0	Reset special mask mode

Note: 1. In the master mode \overline{SP} pin = 1, in slave mode \overline{SP} = 0

Cascading

The 8259A can be easily interconnected in a system of one master with up to eight slaves to handle up to 64 priority levels.

A typical MCS-80/85 system is shown in Figure 2. The master controls, through the 3 line cascade bus, which one of the slaves will release the corresponding address.

As shown in Figure 2, the slave interrupt outputs are connected to the master interrupt request inputs. When a slave request line is activated and afterwards acknowledged, the master will enable the corresponding slave

to release the device routine address during bytes 2 and 3 of \overline{INTA} . (Byte 2 only for 8086/8088).

The cascade bus lines are normally low and will contain the slave address code from the trailing edge of the first \overline{INTA} pulse to the trailing edge of the third pulse. It is obvious that each 8259A in the system must follow a separate initialization sequence and can be programmed to work in a different mode. An EOI command must be issued twice: once for the master and once for the corresponding slave. An address decoder is required to activate the Chip Select (\overline{CS}) input of each 8259A.

The cascade lines of the Master 8259A are activated for any interrupt input, even if no slave is connected to that input.

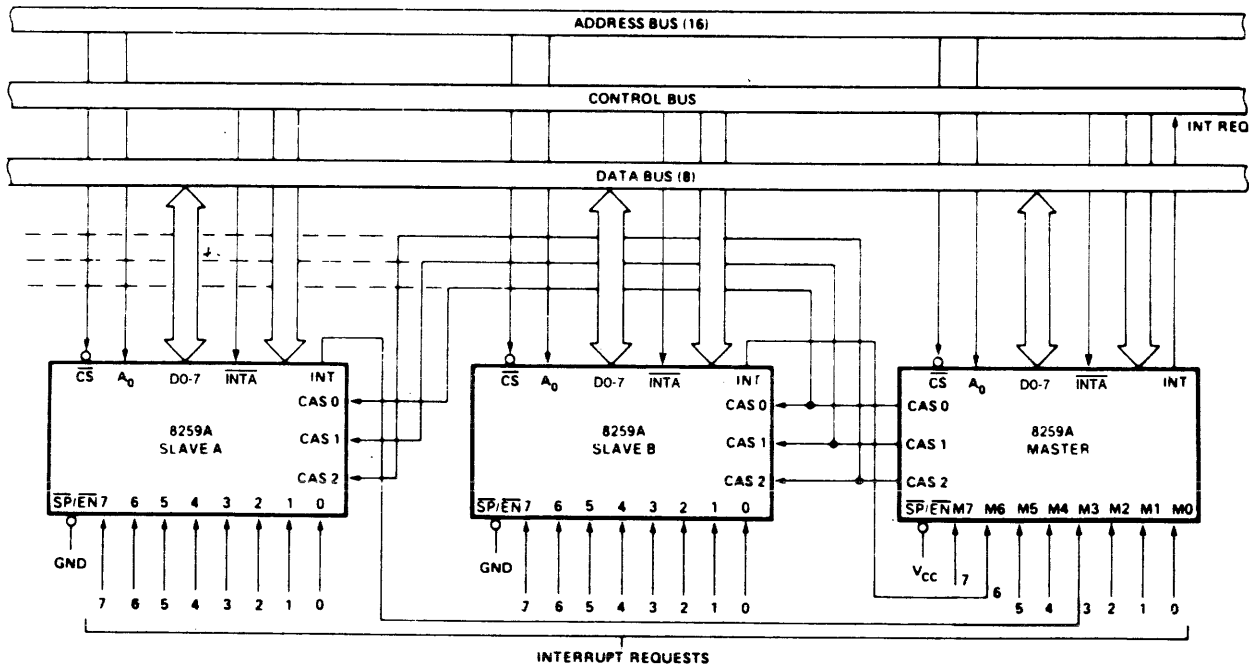


Figure 2. Cascading the 8259A

8259A/8259A-2/8259A-8

PIN FUNCTIONS

NAME	I/O	PIN#	FUNCTION				
V _{CC}	I	28	+5v supply	INT	0	17	Interrupt: This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus it is connected to the CPU's interrupt pin.
GND	I	14	Ground				
$\overline{\text{CS}}$	I	1	Chip Select A low on this pin enables RD and WR communication between the CPU and the 8259A. INTA functions are independent of CS.	IR ₀ -IR ₇	I	18-25	Interrupt Requests: Asynchronous inputs. An interrupt request can be generated by raising an IR input (low to high) and holding it high until it is acknowledged (Edge Triggered Mode), or just by a high level on an IR input (Level Triggered Mode).
$\overline{\text{WR}}$	I	2	Write: A low on this pin when CS is low, enables the 8259A to accept command words from the CPU.				
$\overline{\text{RD}}$	I	3	Read: A low on this pin when CS is low enables the 8259A to release status onto the data bus for the CPU.	$\overline{\text{INTA}}$	I	26	Interrupt Acknowledge: This pin is used to enable 8259A interrupt-vector data onto the data bus. This is done by a sequence of interrupt acknowledge pulses issued by the CPU.
D ₇ -D ₀	I/O	4-11	Bidirectional Data Bus: Control, status and interrupt-vector information is transferred via this bus.	A ₀	I	27	A0 Address Line: This pin acts in conjunction with the CS, WR, and RD pins. It is used by the 8259A to decipher between various Command Words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A0 address line (A1 for 8086/8088).
CAS ₀ -CAS ₂	I/O	12,13,15	Cascade Lines: The CAS lines form a private 8259A bus to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.				
$\overline{\text{SP}}/\overline{\text{EN}}$	I/O	16	Slave Program/Enable Buffer: This is a dual function pin. When in the Buffered Mode it can be used as an output to control buffer transceivers (EN). When not in the buffered mode it is used as an input to designate a master (SP = 1) or slave (SP = 0).				

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias - 40°C to 85°C
 Storage Temperature - 65°C to + 150°C
 Voltage On Any Pin
 With Respect to Ground - 0.5V to + 7V
 Power Dissipation 1 Watt

*COMMENT

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = 5V ± 10% (8259-A), V_{CC} = 5V ± 10% (8259A)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V _{IL}	Input Low Voltage	- .5	V		
V _{IH}	Input High Voltage	2.0	V _{CC} + .5V	V	
V _{OL}	Output Low Voltage		.45	V	I _{OL} = 2.2 mA
V _{OH}	Output High Voltage	2.4		V	I _{OH} = - 400 μA
V _{OH(INT)}	Interrupt Output High Voltage	3.5 2.4		C V	I _{OH} = - 100 μA I _{OH} = - 400 μA
I _{LI}	Input Load Current		10	μA	V _{IN} = V _{CC} to 0V
I _{LOL}	Output Leakage Current		-10	μA	V _{OUT} = 0.45V
I _{CC}	V _{CC} Supply Current		85	mA	
I _{LIR}	IR Input Load Current		-300 10	μA μA	V _{IN} = 0 V _{IN} = V _{CC}

8259A/8259A-2/8259A-8

8259A A.C. CHARACTERISTICS

$T_A = 0^\circ\text{C}$ to 70°C $V_{CC} = 5V \pm 5\%$ (8259A-8) $V_{CC} = 5V \pm 10\%$ (8259A)

TIMING REQUIREMENTS

Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TAHRL	AO/ \overline{CS} Setup to $\overline{RD}/\overline{INTA}$]	50		0		0		ns	
TRHAX	AO/ \overline{CS} Hold after $\overline{RD}/\overline{INTA}$]	5		0		0		ns	
TRLRH	\overline{RD} Pulse Width	420		235		160		ns	
TAHWL	AO/ \overline{CS} Setup to \overline{WR}]	50		0		0		ns	
TWHAX	AO/ \overline{CS} Hold after \overline{WR}]	20		0		0		ns	
TWLWH	\overline{WR} Pulse Width	400		290		190		ns	
TDVWH	Data Setup to \overline{WR}]	300		240		160		ns	
TWHDX	Data Hold after \overline{WR}]	40		0		0		ns	
TJLJH	Interrupt Request Width (Low)	100		100		100		ns	See Note 1
TCVIAL	Cascade Setup to Second or Third \overline{INTA}] (Slave Only)	55		55		40		ns	
TRHRL	End of \overline{RD} to Next Command	160		160		160		ns	
TWHRL	End of \overline{WR} to Next Command	190		190		190		ns	

Note: This is the low time required to clear the input latch in the edge triggered mode.

TIMING RESPONSES

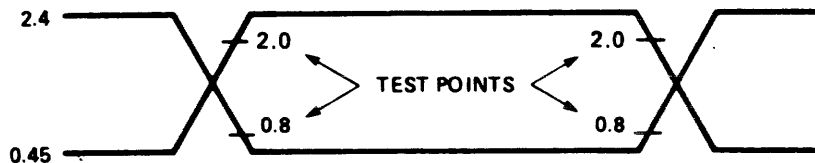
Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TRLDV	Data Valid from $\overline{RD}/\overline{INTA}$]		300		200		120	ns	C of Data Bus = 100 pF
TRHDZ	Data Float after $\overline{RD}/\overline{INTA}$]	10	200		100		85	ns	C of Data Bus Max test C = 100 pF
TJHIH	Interrupt Output Delay		400		350		300	ns	Min. test C = 15 pF
TIAHCV	Cascade Valid from First \overline{INTA}] (Master Only)		565		565		360	ns	$C_{INT} = 100$ pF
TRLEL	Enable Active from \overline{RD}] or \overline{INTA}]		160		125		100	ns	$C_{CASCADE} = 100$ pF
TRHEH	Enable Inactive from \overline{RD}] or \overline{INTA}]		325		150		d150	ns	
TAHDV	Data Valid from Stable Address		350		200		200	ns	
TCVDV	Cascade Valid to Valid Data		300		300		200	ns	

CAPACITANCE

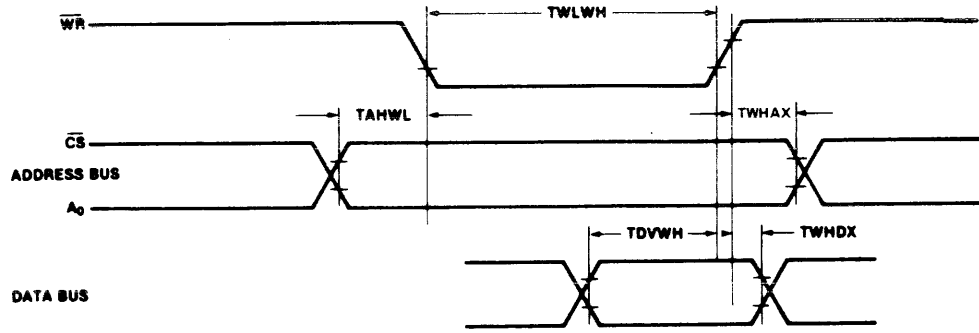
$T_A = 25^\circ\text{C}$; $V_{CC} = \text{GND} = 0V$

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	$f_c = 1$ MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

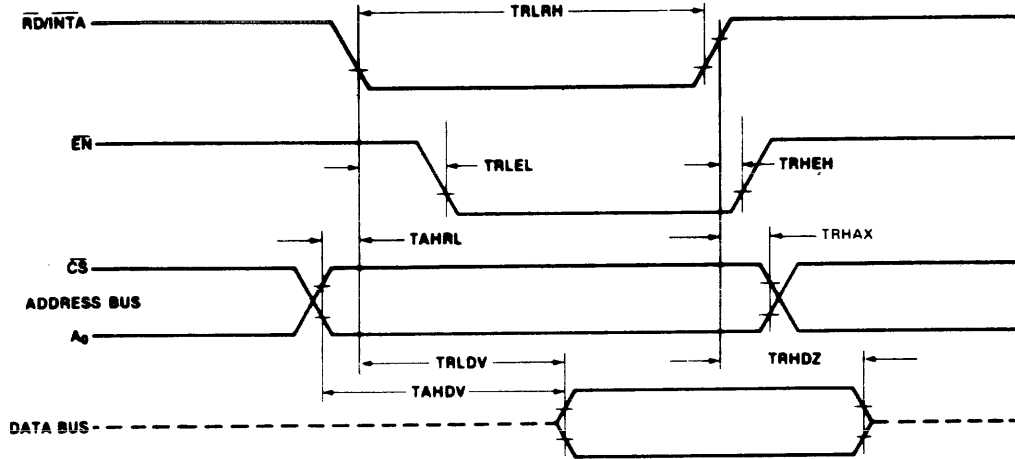
Input and Output Waveforms for A.C. Tests



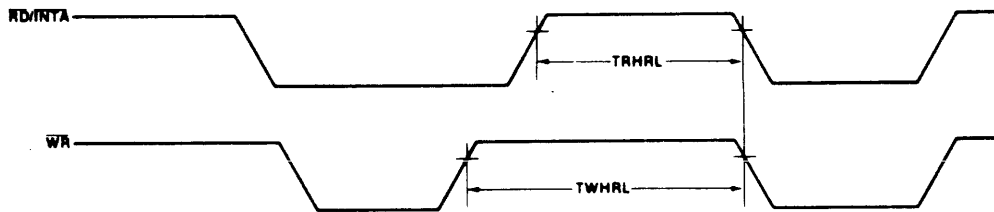
WRITE MODE



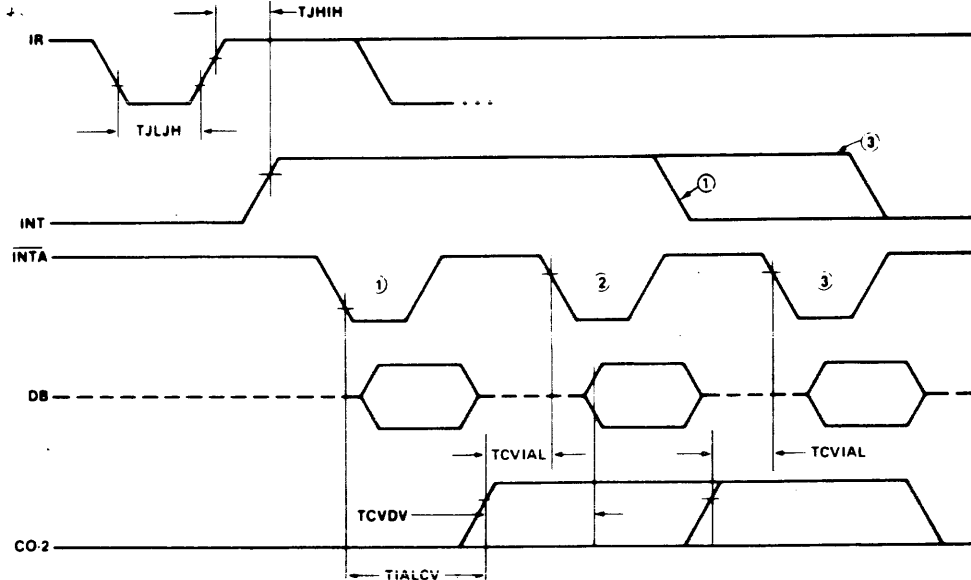
READ/INTA MODE



OTHER TIMING



INTA SEQUENCE



NOTE: Interrupt output must remain HIGH for 100ns until leading edge of first INTA.
 ① MCS 8088 Systems only.
 ② Cycle 1 in MCS 88 Systems. The Data Bus is not active.

September 1979

**Using the 8259A Programmable
Interrupt Controller**

Robin Jigour
Microcomputer Applications

Using the 8259A Programmable Interrupt Controller

Contents

INTRODUCTION

CONCEPTS

- MCS80™-8259A Overview
- MCS85™-8259A Overview
- MCS86/88™-8259A Overview

FUNCTIONAL BLOCK DIAGRAM

- Interrupt Registers and Control Logic
- Other Functional Blocks
- Pin Functions

OPERATION OF THE 8259A

- Interrupt Vectoring
 - MCS80/85 Mode
 - MCS86/88 Mode

- Interrupt Priorities
 - Fully Nested Mode
 - End of Interrupt
 - Automatic Rotation
 - Specific Rotation
 - Interrupt Masking

- Interrupt Triggering
 - Level Triggered Mode
 - Edge Triggered Mode

- Interrupt Status
 - Reading Interrupt Registers
 - Poll Command

- Interrupt Cascading
 - Cascade Mode
 - Special Fully Nested Mode
 - Buffered Mode

PROGRAMMING THE 8259A

- Initialization Command Words (ICWs)
- Operational Command Words (OCWs)

APPLICATION EXAMPLES

- Power Fail/Auto Start with Battery Back-Up RAM
- 78 Level Interrupt Structure
- Timer Controlled Interrupts

CONCLUSIONS

APPENDIX A

APPENDIX B

INTRODUCTION

The Intel 8259A is a Programmable Interrupt Controller (PIC) designed for use in real-time interrupt driven microcomputer systems. The 8259A manages eight levels of interrupts and has built-in features for expansion up to 64 levels with additional 8259A's. Its versatile design allows it to be used within MCS-80, MCS-85, MCS-86, and MCS-88 microcomputer systems. Being fully programmable, the 8259A provides a wide variety of modes and commands to tailor 8259A interrupt processing for the specific needs of the user. These modes and commands control a number of interrupt oriented functions such as interrupt priority selection and masking of interrupts. The 8259A programming may be dynamically changed by the software at any time, thus allowing complete interrupt control throughout program execution.

The 8259A is an enhanced, fully compatible revision of its predecessor, the 8259. This means the 8259A can use all hardware and software originally designed for the 8259 without any changes. Furthermore, it provides additional modes that increase its flexibility in MCS-80 and MCS-85 systems and allow it to work in MCS-86 and MCS-88 systems. These modes are:

- MCS-86/88 Mode
- Automatic End of Interrupt Mode
- Level Triggered Mode
- Special Fully Nested Mode
- Buffered Mode

Each of these are covered in depth further in this application note.

This application note was written to explain completely how to use the 8259A within MCS-80, MCS-85, MCS-86, and MCS-88 microcomputer systems. It is divided into five sections. The first section, "Concepts", explains the concepts of interrupts and presents an overview of how the 8259A works with each microcomputer system mentioned above. The second section, "Functional Block Diagram", describes the internal functions of the 8259A in block diagram form and provides a detailed functional description of each device pin. "Operation of the 8259A", the third section, explains in depth the operation and use of each of the 8259A modes and commands. For clarity of explanation, this section doesn't make reference to the actual programming of the 8259A. Instead, all programming is covered in the fourth section, "Programming the 8259A". This section explains how to program the 8259A with the modes and commands mentioned in the previous section. These two sections are referenced in Appendix A. The fifth and final section "Application Examples", shows the 8259A in three typical applications. These applications are fully explained with reference to both hardware and software.

The reader should note that some of the terminology used throughout this application note may differ slightly from existing data sheets. This is done to better clarify and explain the operation and programming of the 8259A.

1. CONCEPTS

In microcomputer systems there is usually a need for the processor to communicate with various Input/Out-

put (I/O) devices such as keyboards, displays, sensors, and other peripherals. From the system viewpoint, the processor should spend as little time as possible servicing the peripherals since the time required for these I/O chores directly affects the amount of time available for other tasks. In other words, the system should be designed so that I/O servicing has little or no effect on the total system throughput. There are two basic methods of handling the I/O chores in a system: status polling and interrupt servicing.

The status poll method of I/O servicing essentially involves having the processor "ask" each peripheral if it needs servicing by testing the peripheral's status line. If the peripheral requires service, the processor branches to the appropriate service routine; if not, the processor continues with the main program. Clearly, there are several problems in implementing such an approach. First, how often a peripheral is polled is an important constraint. Some idea of the "frequency-of-service" required by each peripheral must be known and any software written for the system must accommodate this time dependence by "scheduling" when a device is polled. Second, there will obviously be times when a device is polled that is not ready for service, wasting the processor time that it took to do the poll. And other times, a ready device would have to wait until the processor "makes its rounds" before it could be serviced, slowing down the peripheral.

Other problems arise when certain peripherals are more important than others. The only way to implement the "priority" of devices is to poll the high priority devices more frequently than lower priority ones. It may even be necessary to poll the high priority devices while in a low priority device service routine. It is easy to see that the polled approach can be inefficient both time-wise and software-wise. Overall, the polled method of I/O servicing can have a detrimental effect on system throughput, thus limiting the tasks that can be performed by the processor.

A more desirable approach in most systems would allow the processor to be executing its main program and only stop to service the I/O when told to do so by the I/O itself. This is called the interrupt service method. In effect, the device would asynchronously signal the processor when it required service. The processor would finish its current instruction and then vector to the service routine for the device requesting service. Once the service routine is complete, the processor would resume exactly where it left off. Using the interrupt service method, no processor time is spent testing devices, scheduling is not needed, and priority schemes are readily implemented. It is easy to see that, using the interrupt service approach, system throughput would increase, allowing more tasks to be handled by the processor.

However, to implement the interrupt service method between processor and peripherals, additional hardware is usually required. This is because, after interrupting the processor, the device must supply information for vectoring program execution. Depending on the processor used, this can be accomplished by the device taking control of the data bus and "jamming" an instruction(s) onto it. The instruction(s) then vectors the pro-

gram to the proper service routine. This of course requires additional control logic for each interrupt requesting device. Yet the implementation so far is only in the most basic form. What if certain peripherals are to be of higher priority than others? What if certain interrupts must be disabled while others are to be enabled? The possible variations go on, but they all add up to one theme; to provide greater flexibility using the interrupt service method, hardware requirements increase.

So, we're caught in the middle. The status poll method is a less desirable way of servicing I/O in terms of throughput, but its hardware requirements are minimal. On the other hand, the interrupt service method is most desirable in terms of flexibility and throughput, but additional hardware is required.

The perfect situation would be to have the flexibility and throughput of the interrupt method in an implementation with minimal hardware requirements. The 8259A Programmable Interrupt Controller (PIC) makes this all possible.

The 8259A Programmable Interrupt Controller (PIC) was designed to function as an overall manager of an interrupt driven system. No additional hardware is required. The 8259A alone can handle eight prioritized interrupt levels, controlling the complete interface between peripherals and processor. Additional 8259A's can be "cascaded" to increase the number of interrupt levels processed. A wide variety of modes and commands for programming the 8259A give it enough flexibility for almost any interrupt controlled structure. Thus, the 8259A is the feasible answer to handling I/O servicing in microcomputer systems.

Now, before explaining exactly how to use the 8259A, let's go over interrupt structures of the MCS-80, MCS-85, MCS-86, and MCS-88 systems, and how they interact with the 8259A. Figure 1 shows a block diagram of the 8259A interfacing with a standard system bus. This may prove useful as reference throughout the rest of the "Concepts" section.

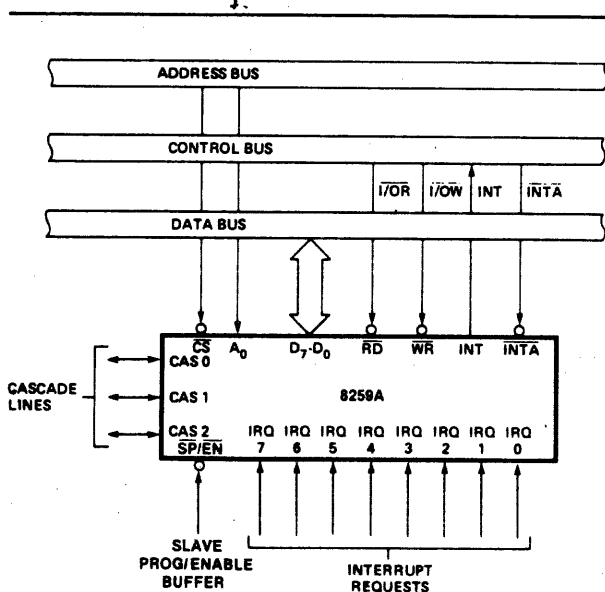


Figure 1. 8259A Interface to Standard System Bus

1.1 MCS-80™—8259A OVERVIEW

In an MCS-80—8259A interrupt configuration, as in Figure 2, a device may cause an interrupt by pulling one of the 8259A's interrupt request pins (IRQ0-IRQ7) high. If the 8259A accepts the interrupt request (this depends on its programmed condition), the 8259A's INT (interrupt) pin will go high, driving the 8080A's INT pin high.

The 8080A can receive an interrupt request any time, since its INT input is asynchronous. The 8080A, however, doesn't always have to acknowledge an interrupt request immediately. It can accept or disregard requests under software control using the EI (Enable Interrupt) or DI (Disable Interrupt) instructions. These instructions either set or reset an internal interrupt enable flip-flop. The output of this flip-flop controls the state of the INTE (Interrupt Enabled) pin. Upon reset, the 8080A interrupts are disabled, making INTE low.

At the end of each instruction cycle, the 8080A examines the state of its INT pin. If an interrupt request is present and interrupts are enabled, the 8080A enters an interrupt machine cycle. During the interrupt machine cycle the 8080A resets the internal interrupt enable flip-flop, disabling further interrupts until an EI instruction is executed. Unlike normal machine cycles, the interrupt machine cycle doesn't increment the program counter. This ensures that the 8080A can return to the pre-interrupt program location after the interrupt is completed. The 8080A then issues an \overline{INTA} (Interrupt Acknowledge) pulse via the 8228 System Controller Bus Driver. This \overline{INTA} pulse signals the 8259A that the 8080A is honoring the request and is ready to process the interrupt.

The 8259A can now vector program execution to the corresponding service routine. This is done during a sequence of the three \overline{INTA} pulses from the 8080A via the 8228. Upon receiving the first \overline{INTA} pulse the 8259A places the opcode for a CALL instruction on the data bus. This causes the contents of the program counter to be pushed onto the stack. In addition, the CALL instruction causes two more \overline{INTA} pulses to be issued, allowing the 8259A to place onto the data bus the starting address of the corresponding service routine. This address is called the interrupt-vector address. The lower 8 bits (LSB) of the interrupt-vector address are released during the second \overline{INTA} pulse and the upper 8 bits (MSB) during the third \overline{INTA} pulse. Once this sequence is completed, program execution then vectors to the service routine at the interrupt-vector address.

If the same registers are used by both the main program and the interrupt service routine, their contents should be saved when entering the service routine. This includes the Program Status Word (PSW) which consists of the accumulator and flags. The best way to do this is to "PUSH" each register used onto the stack. The service routine can then "POP" each register off the stack in the reverse order when it is completed. This prevents any ambiguous operation when returning to the main program.

Once the service routine is completed, the main program may be re-entered by using a normal RET (Return) instruction. This will "POP" the original con-

tents of the program counter back off the stack to resume program execution where it left off. Note, that because interrupts are disabled during the interrupt acknowledge sequence, the EI instruction must be executed either during the service routine or the main program before further interrupts can be processed.

For additional information on the 8080A interrupt structure and operation, refer to the MCS-80 User's Manual.

1.2 MCS-85™—8259A OVERVIEW

An MCS-85—8259A configuration processes interrupts in much the same format as an MCS-80—8259A confi-

uration. When an interrupt occurs, a sequence of three \overline{INTA} pulses causes the 8259A to release onto the data bus a CALL instruction and an interrupt-vector address for the corresponding service routine. Other events that occur during the 8080A interrupt machine cycle, such as disabling interrupts and not incrementing the program counter, also occur in the 8085A interrupt acknowledge machine cycle. Additionally, the instructions for saving registers, enabling or disabling of interrupts, and returning from service routines are literally the same.

The 8085A, however, has a different interrupt hardware scheme as shown in Figure 3. For one, the 8085A supplies its own \overline{INTA} output pin rather than using an addi-

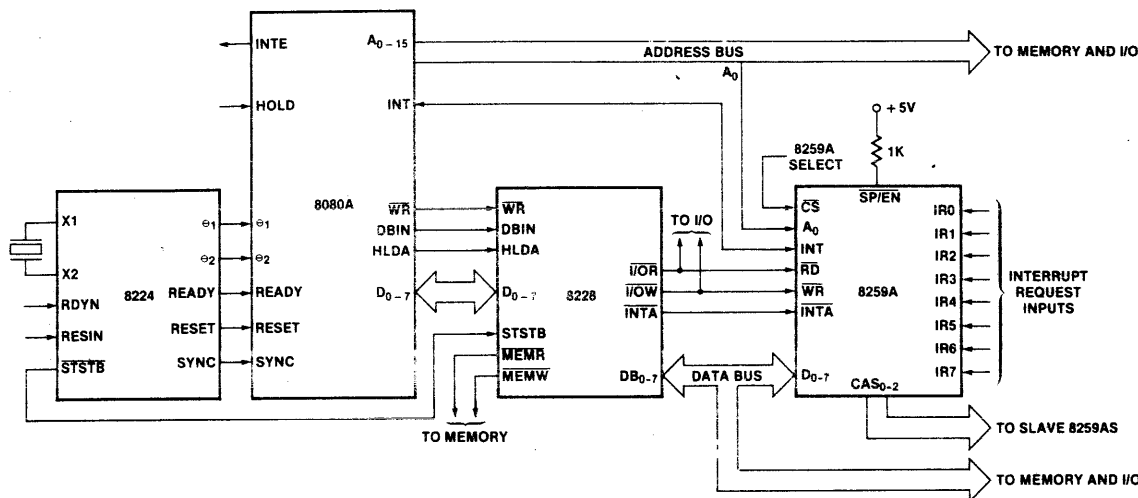


Figure 2. MCS-80 8259A Basic Configuration Example

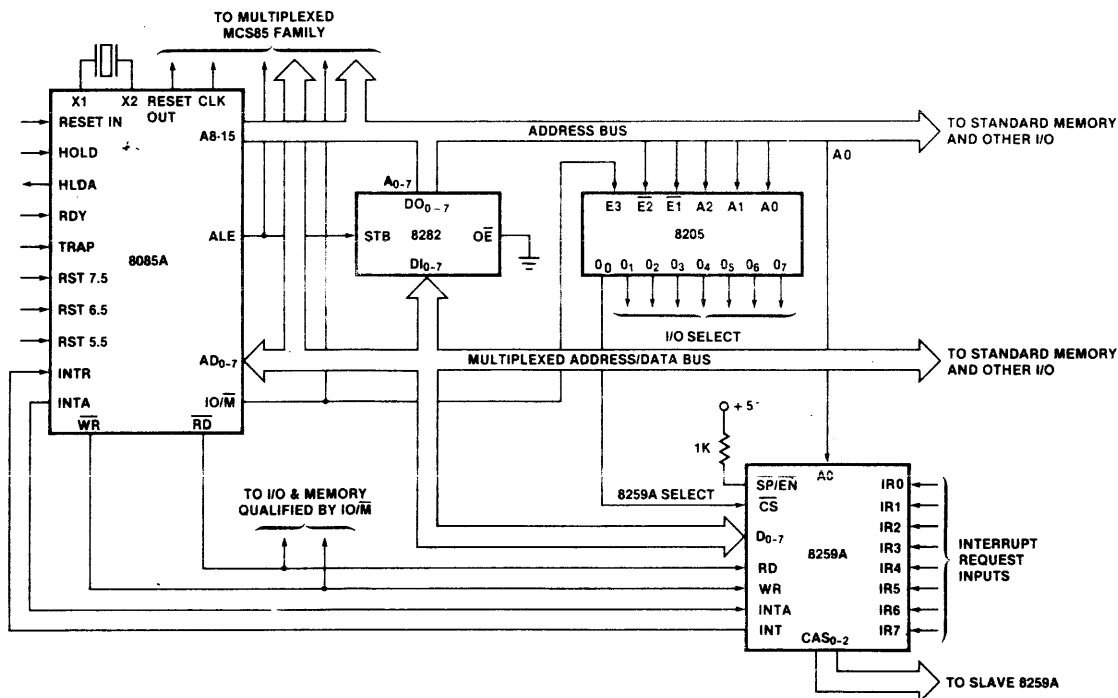


Figure 3. MCS-85™ 8259A Basic Configuration Example

For further information on 8086/8088 interrupt operation and internal interrupt structure refer to the MCS-86 User's Manual and the 8086 System Design application note.

2. 8259A FUNCTIONAL BLOCK DIAGRAM

A block diagram of the 8259A is shown in Figure 7. As can be seen from this figure, the 8259A consists of eight major blocks: the Interrupt Request Register (IRR), the In-Service Register (ISR), the Interrupt Mask Register (IMR), the Priority Resolver (PR), the cascade buffer/comparator, the data bus buffer, and logic blocks for control and read/write. We'll first go over the blocks directly related to interrupt handling, the IRR, ISR, IMR, PR, and the control logic. The remaining functional blocks are then discussed.

2.1 INTERRUPT REGISTERS AND CONTROL LOGIC

Basically, interrupt requests are handled by three "cascaded" registers: the Interrupt Request Register (IRR) is used to store all the interrupt levels requesting service; the In-Service Register (ISR) stores all the levels which are being serviced; and the Interrupt Mask Register (IMR) stores the bits of the interrupt lines to be masked. The Priority Resolver (PR) looks at the IRR, ISR and IMR, and determines whether an INT should be issued by the control logic to the processor.

Figure 8 shows conceptually how the Interrupt Request (IR) input handles an interrupt request and how the various interrupt registers interact. The figure repre-

sents one of eight "daisy-chained" priority cells, one for each IR input.

The best way to explain the operation of the priority cell is to go through the sequence of internal events that happen when an interrupt request occurs. However, first, notice that the input circuitry of the priority cell allows for both level sensitive and edge sensitive IR inputs. Deciding which method to use is dependent on the particular application and will be discussed in more detail later.

When the IR input is in an inactive state (LOW), the edge sense latch is set. If edge sensitive triggering is selected, the "Q" output of the edge sense latch will arm the input gate to the request latch. This input gate will be disarmed after the IR input goes active (HIGH) and the interrupt request has been acknowledged. This disables the input from generating any further interrupts until it has returned low to re-arm the edge sense latch. If level sensitive triggering is selected, the "Q" output of the edge sense latch is rendered useless. This means the level of the IR input is in complete control of interrupt generation; the input won't be disarmed once acknowledged.

When an interrupt occurs on the IR input, it propagates through the request latch and to the PR (assuming the input isn't masked). The PR looks at the incoming requests and the currently in-service interrupts to ascertain whether an interrupt should be issued to the processor. Let's assume that the request is the only one incoming and no requests are presently in service. The PR then causes the control logic to pull the INT line to the processor high.

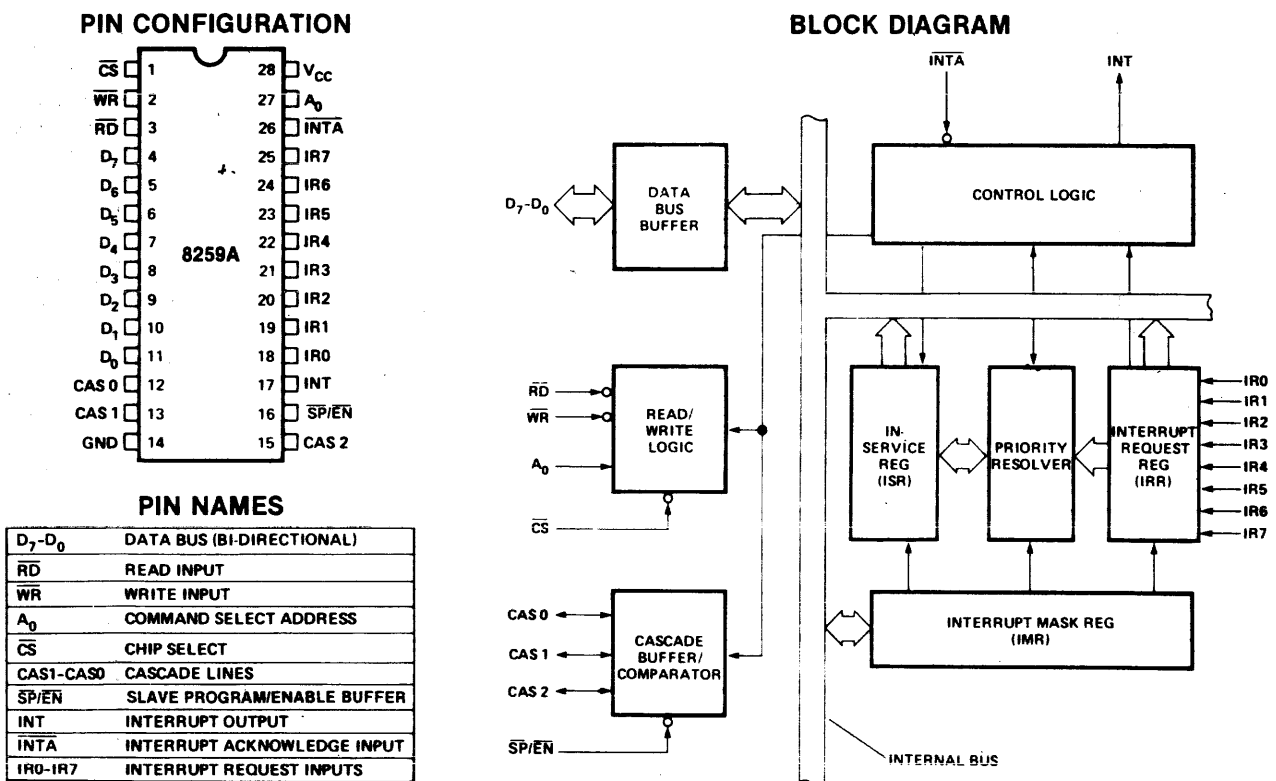


Figure 7. 8259A Block Diagram and Pin Configuration

both the code segment and the instruction pointer are then also pushed onto the stack. Thus, the stack retains the pre-interrupt flag status and pre-interrupt program location which are used to return from the service routine. The 8086/8088 then issues the first of two \overline{INTA} pulses which signal the 8259A that the 8086/8088 has honored its interrupt request. If the 8086/8088 is used in its "MIN Mode" the \overline{INTA} signal is available from the 8086/8088 on its \overline{INTA} pin. If the 8086/8088 is used in the "MAX Mode" the \overline{INTA} signal is available via the 8288 Bus Controller \overline{INTA} pin. Additionally, in the "MAX Mode" the 8086/8088 LOCK pin goes low during the interrupt acknowledge sequence. The LOCK signal can be used to indicate to other system bus masters not to gain control of the system bus during the interrupt acknowledge sequence. A "HOLD" request won't be honored while LOCK is low.

The 8259A is now ready to vector program execution to the corresponding service routine. This is done during the sequence of the two \overline{INTA} pulses issued by the 8086/8088. Unlike operation with the 8080A or 8085A, the 8259A doesn't place a CALL instruction and the starting address of the service routine on the data bus. Instead, the first \overline{INTA} pulse is used only to signal the 8259A of the honored request. The second \overline{INTA} pulse causes the 8259A to place a single interrupt-vector byte onto the data bus. Not used as a direct address, this interrupt-vector byte pertains to one of 256 interrupt "types" supported by the 8086/8088 memory. Program execution is vectored to the corresponding service routine by the contents of a specified interrupt type.

All 256 interrupt types are located in absolute memory locations 0 through 3FFH which make up the 8086/8088's interrupt-vector table. Each type in the interrupt-vector table requires 4 bytes of memory and stores a code segment address and an instruction pointer address. Figure 5 shows a block diagram of the interrupt-vector table. Locations 0 through 3FFH should be reserved for the interrupt-vector table alone. Furthermore, memory locations 00 through 7FH (types 0-31) are reserved for use by Intel Corporation for Intel hardware and software products. To maintain compatibility with present and future Intel products, these locations should not be used.

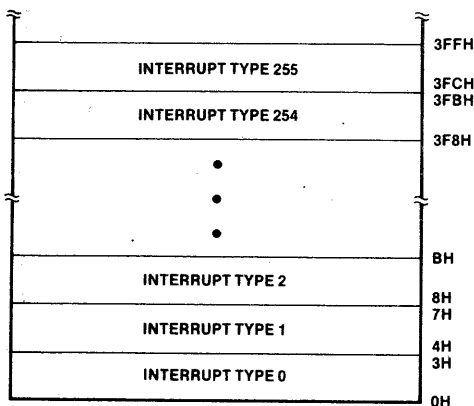


Figure 5. 8086/8088 Interrupt Vector Table

When the 8086/8088 receives an interrupt-vector byte from the 8259A, it multiplies its value by four to acquire the address of the interrupt type. For example, if the interrupt-vector byte specifies type 128 (80H), the vectored address in 8086/8088 memory is $4 \times 80H$, which equals 200H. Program execution is then vectored to the service routine whose address is specified by the code segment and instruction pointer values within type 128 located at 200H. To show how this is done, let's assume interrupt type 128 is to vector data to 8086/8088 memory location 2FF5FH. Figure 6 shows two possible ways to set values of the code segment and instruction pointer for vectoring to location 2FF5FH. Address generation by the code segment and instruction pointer is accomplished by an offset (they overlap). Of the total 20-bit address capability, the code segment can designate the upper 16 bits, the instruction pointer can designate the lower 16 bits.

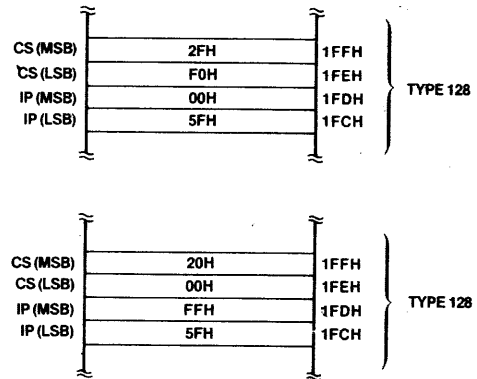


Figure 6. Two Examples of 8086/8088 Interrupt Type 128 Vectoring to Location 2FF5FH

When entering an interrupt service routine, those registers that are mutually used between the main program and service routine should be saved. The best way to do this is to "PUSH" each register used onto the stack immediately. The service routine can then "POP" each register off the stack in the same order when it is completed.

Once the service routine is completed the main program may be re-entered by using a IRET (Interrupt Return) instruction. The IRET instruction will pop the pre-interrupt instruction pointer, code segment and flags off the stack. Thus the main program will resume where it was interrupted with the same flag status regardless of changes in the service routine. Note especially that this includes the state of the IF flag, thus interrupts are re-enabled automatically when returning from the service routine.

Beside external interrupt generation from the INTR pin, the 8086/8088 is also able to invoke interrupts by software. Three interrupt instructions are provided: INT, INT (Type 3), and INTO. INT is a two byte instruction, the second byte selects the interrupt type. INT (Type 3) is a one byte instruction which selects interrupt Type 3. INTO is a conditional one byte interrupt instruction which selects interrupt Type 4 if the OF flag (trap on overflow) is set. All the software interrupts vector program execution as the hardware interrupts do.

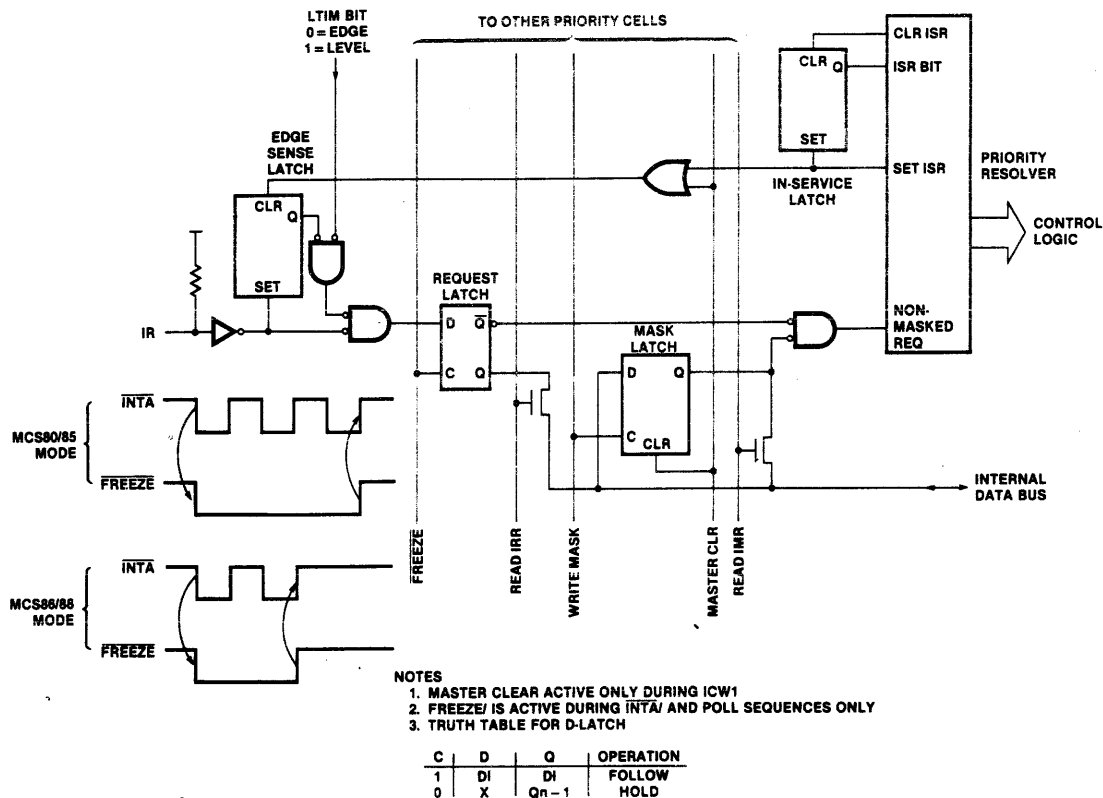


Figure 8. Priority Cell

When the processor honors the INT pulse, it sends a sequence of INTA pulses to the 8259A (three for 8080A/8085A, two for 8086/8088). During this sequence the state of the request latch is frozen (note the INTA-freeze request timing diagram). Priority is again resolved by the PR to determine the appropriate interrupt vectoring which is conveyed to the processor via the data bus.

Immediately after the interrupt acknowledge sequence, the PR sets the corresponding bit in the ISR which simultaneously clears the edge sense latch. If edge sensitive triggering is used, clearing the edge sense latch also disarms the request latch. This inhibits the possibility of a still active IR input from propagating through the priority cell. The IR input must return to an inactive state, setting the edge sense latch, before another interrupt request can be recognized. If level sensitive triggering is used, however, clearing the edge sense latch has no effect on the request latch. The state of the request latch is entirely dependent upon the IR input level. Another interrupt will be generated immediately if the IR level is left active after its ISR bit has been reset. An ISR bit gets reset with an End-of-Interrupt (EOI) command issued in the service routine. End-of-interrupts will be covered in more detail later.

2.2 OTHER FUNCTIONAL BLOCKS

Data Bus Buffer

This three-state, bidirectional 8-bit buffer is used to interface the 8259A to the processor system data bus (via

DB0-DB7). Control words, status information, and interrupt-vector data are transferred through the data bus buffer.

Read/Write Control Logic

The function of this block is to control the programming of the 8259A by accepting OUTPUT commands from the processor. It also controls the releasing of status onto the data bus by accepting INPUT commands from the processor. The initialization and operation command word registers which store the various control formats are located in this block. The \overline{RD} , \overline{WR} , $\overline{A0}$, and \overline{CS} pins are used to control access to this block by the processor.

Cascade Buffer/Comparator

As mentioned earlier, multiple 8259A's can be combined to expand the number of interrupt levels. A master-slave relationship of cascaded 8259A's is used for the expansion. The $\overline{SP/EN}$ and the CAS0-2 pins are used for operation of this block. The cascading of 8259A's is covered in depth in the "Operation of the 8259A" section of this application note.

2.3 PIN FUNCTIONS

Name Pin # I/O Function

V _{CC}	28	I	+5V supply
GND	14	I	Ground

Name	Pin #	I/O	Function
$\overline{\text{CS}}$	1	I	<i>Chip Select:</i> A low on this pin enables $\overline{\text{RD}}$ and $\overline{\text{WR}}$ communication between the CPU and the 8259A. $\overline{\text{INTA}}$ functions are independent of $\overline{\text{CS}}$.
$\overline{\text{WR}}$	2	I	<i>Write:</i> A low on this pin when $\overline{\text{CS}}$ is low enables the 8259A to accept command words from the CPU.
$\overline{\text{RD}}$	3	I	<i>Read:</i> A low on this pin when $\overline{\text{CS}}$ is low enables the 8259A to release status onto the data bus for the CPU.
D7-D0	4-11	I/O	<i>Bidirectional Data Bus:</i> Control, status and interrupt-vector information is transferred via this bus.
CAS0- CAS2	12,13, 15	I/O	<i>Cascade Lines:</i> The CAS lines form a private 8259A bus to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.
$\overline{\text{SP/EN}}$	16	I/O	<i>Slave Program/Enable Buffer:</i> This is a dual function pin. When in the buffered mode it can be used as an output to control buffer transceivers ($\overline{\text{EN}}$). When not in the buffered mode it is used as an input to designate a master ($\overline{\text{SP}} = 1$) or slave ($\overline{\text{SP}} = 0$).
INT	17	O	<i>Interrupt:</i> This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus it is connected to the CPU's interrupt pin.
IR0- IR7	18-25	I	<i>Interrupt Requests:</i> Asynchronous inputs. An interrupt request can be generated by raising an IR input (low to high) and holding it high until it is acknowledged (edge triggered mode), or just by a high level on an IR input (level triggered mode).
$\overline{\text{INTA}}$	26	I	<i>Interrupt Acknowledge:</i> This pin is used to enable 8259A interrupt-vector data onto the data bus. This is done by a sequence of interrupt acknowledge pulses issued by the CPU.
A0	27	I	<i>A0 Address Line:</i> This pin acts in conjunction with the $\overline{\text{CS}}$, $\overline{\text{WR}}$, and $\overline{\text{RD}}$ pins. It is used by the 8259A to decipher between various command words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A0 address line (A1 for 8086/8088).

3. OPERATION OF THE 8259A

Interrupt operation of the 8259A falls under five main categories: vectoring, priorities, triggering, status, and cascading. Each of these categories use various modes and commands. This section will explain the operation of these modes and commands. For clarity of explanation, however, the actual programming of the 8259A isn't

covered in this section but in "Programming the 8259A". Appendix A is provided as a cross reference between these two sections.

3.1 INTERRUPT VECTORING

Each IR input of the 8259A has an individual interrupt-vector address in memory associated with it. Designation of each address depends upon the initial programming of the 8259A. As stated earlier, the interrupt sequence and addressing of an MCS-80 and MCS-85 system differs from that of an MCS-86 and MCS-88 system. Thus, the 8259A must be initially programmed in either a MCS-80/85 or MCS-86/88 mode of operation to insure the correct interrupt vectoring.

MCS-80/85™ Mode

When programmed in the MCS-80/85 mode, the 8259A should only be used within an 8080A or an 8085A system. In this mode the 8080A/8085A will handle interrupts in the format described in the "MCS-80—8259A or MCS-85—8259A Overviews."

Upon interrupt request in the MCS-80/85 mode, the 8259A will output to the data bus the opcode for a CALL instruction and the address of the desired routine. This is in response to a sequence of three $\overline{\text{INTA}}$ pulses issued by the 8080A/8085A after the 8259A has raised INT high.

The first INTA pulse to the 8259A enables the CALL opcode " CD_H " onto the data bus. It also resolves IR priorities and effects operation in the cascade mode, which will be covered later. Contents of the first interrupt-vector byte are shown in Figure 9A.

During the second and third $\overline{\text{INTA}}$ pulses, the 8259A conveys a 16-bit interrupt-vector address to the 8080A/8085A. The interrupt-vector addresses for all eight levels are selected when initially programming the 8259A. However, only one address is needed for programming. Interrupt-vector addresses of IR0-IR7 are automatically set at equally spaced intervals based on the one programmed address. Address intervals are user definable to 4 or 8 bytes apart. If the service routine for a device is short it may be possible to fit the entire routine within an 8-byte interval. Usually, though, the service routines require more than 8 bytes. So, a 4-byte interval is used to store a Jump (JMP) instruction which directs the 8080A/8085A to the appropriate routine. The 8-byte interval maintains compatibility with current 8080A/8085A Restart (RST) instruction software, while the 4-byte interval is best for a compact jump table. If the 4-byte interval is selected, then the 8259A will automatically insert bits A0-A4. This leaves A5-A15 to be programmed by the user. If the 8-byte interval is selected, the 8259A will automatically insert bits A0-A5. This leaves only A6-A15 to be programmed by the user.

The LSB of the interrupt-vector address is placed on the data bus during the second INTA pulse. Figure 9B shows the contents of the second interrupt-vector byte for both 4 and 8-byte intervals.

The MSB of the interrupt-vector address is placed on the data bus during the third INTA pulse. Contents of the third interrupt-vector byte is shown in Figure 9C.

	D7	D6	D5	D4	D3	D2	D1	D0
CALL CODE	1	1	0	0	1	1	0	1

A. FIRST INTERRUPT VECTOR BYTE, MCS80/85 MODE

Interval = 4								
IR	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	A5	1	1	1	0	0
6	A7	A6	A5	1	1	0	0	0
5	A7	A6	A5	1	0	1	0	0
4	A7	A6	A5	1	0	0	0	0
3	A7	A6	A5	0	1	1	0	0
2	A7	A6	A5	0	1	0	0	0
1	A7	A6	A5	0	0	1	0	0
0	A7	A6	A5	0	0	0	0	0

Interval = 8								
IR	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	1	1	1	0	0	0
6	A7	A6	1	1	0	0	0	0
5	A7	A6	1	0	1	0	0	0
4	A7	A6	1	0	0	0	0	0
3	A7	A6	0	1	1	0	0	0
2	A7	A6	0	1	0	0	0	0
1	A7	A6	0	0	1	0	0	0
0	A7	A6	0	0	0	0	0	0

B. SECOND INTERRUPT VECTOR BYTE, MCS80/85 MODE

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

C. THIRD INTERRUPT VECTOR BYTE, MCS80/85 MODE

Figure 9. 9A-C. Interrupt-Vector Bytes for 8259A, MCS 80/85 Mode

MCS-86/88™ Mode

When programmed in the MCS-86/88 mode, the 8259A should only be used within an MCS-86 or MCS-88 system. In this mode, the 8086/8088 will handle interrupts in the format described earlier in the "8259A—8086/8088 Overview".

Upon interrupt in the MCS-86/88 mode, the 8259A will output a single interrupt-vector byte to the data bus. This is in response to only two INTA pulses issued by the 8086/8088 after the 8259A has raised INT high.

The first INTA pulse is used only for set-up purposes internal to the 8259A. As in the MCS-80/85 mode, this set-up includes priority resolution and cascade mode operations which will be covered later. Unlike the MCS-80/85 mode, no CALL opcode is placed on the data bus.

The second INTA pulse is used to enable the single interrupt-vector byte onto the data bus. The 8086/8088 uses this interrupt-vector byte to select one of 256 interrupt "types" in 8086/8088 memory. Interrupt type selection for all eight IR levels is made when initially programming the 8259A. However, reference to only one interrupt type is needed for programming. The upper 5 bits of the interrupt vector byte are user definable. The lower 3 bits are automatically inserted by the 8259A depending upon the IR level.

Contents of the interrupt-vector byte for 8086/8088 type selection is put on the data bus during the second INTA pulse and is shown in Figure 10.

IR	D7	D6	D5	D4	D3	D2	D1	D0
7	T7	T6	T5	T4	T3	1	1	1
6	T7	T6	T5	T4	T3	1	1	0
5	T7	T6	T5	T4	T3	1	0	1
4	T7	T6	T5	T4	T3	1	0	0
3	T7	T6	T5	T4	T3	0	1	1
2	T7	T6	T5	T4	T3	0	1	0
1	T7	T6	T5	T4	T3	0	0	1
0	T7	T6	T5	T4	T3	0	0	0

Figure 10. Interrupt Vector Byte, MCS 86/88™ Mode

3.2 INTERRUPT PRIORITIES

A variety of modes and commands are available for controlling interrupt priorities of the 8259A. All of them are programmable, that is, they may be changed dynamically under software control. With these modes and commands, many possibilities are conceivable, giving the user enough versatility for almost any interrupt controlled application.

Fully Nested Mode

The fully nested mode of operation is a general purpose priority mode. This mode supports a multilevel-interrupt structure in which priority order of all eight IR inputs are arranged from highest to lowest.

Unless otherwise programmed, the fully nested mode is entered by default upon initialization. At this time, IR0 is assigned the highest priority through IR7 the lowest. The fully nested mode, however, is not confined to this IR structure alone. Once past initialization, other IR inputs can be assigned highest priority also, keeping the multilevel-interrupt structure of the fully nested mode. Figure 11A-C shows some variations of the priority structures in the fully nested mode.

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	7	6	5	4	3	2	1	0
A								
IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	4	3	2	1	0	7	6	5
B								
IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	1	0	7	6	5	4	3	2
C								

Figure 11. A-C. Some Variations of Priority Structure in the Fully Nested Mode

Further explanation of the fully nested mode, in this section, is linked with information of general 8259A interrupt operations. This is done to ease explanation to the user in both areas.

In general, when an interrupt is acknowledged, the highest priority request is determined from the IRR (Interrupt Request Register). The interrupt vector is then placed on the data bus. In addition, the corresponding bit in the ISR (In-Service Register) is set to designate the routine in service. This ISR bit remains set until an EOI (End-Of-Interrupt) command is issued to the 8259A. EOI's will be explained in greater detail shortly.

In the fully nested mode, while an ISR bit is set, all further requests of the same or lower priority are inhibited from generating an interrupt to the microprocessor. A higher priority request, though, can generate an interrupt, thus vectoring program execution to its service routine. Interrupts are only acknowledged, however, if the microprocessor has previously executed an "Enable Interrupts" instruction. This is because the interrupt request pin on the microprocessor gets disabled automatically after acknowledgement of any interrupt. The assembly language instructions used to enable interrupts are "EI" for 8080A/8085A and "STI" for 8086/8088. Interrupts can be disabled by using the instruction "DI" for 8080A/ 8085A and "CLI" for 8086/8088. When a routine is completed a "return" instruction is executed, "RET" for 8080A/8085A and "IRET" for 8086/8088.

Figure 12 illustrates the correct usage of interrupt related instructions and the interaction of interrupt levels in the fully nested mode.

Assuming the IR priority assignment for the example in Figure 12 is IR0 the highest through IR7 the lowest, the sequence is as follows. During the main program, IR3 makes a request. Since interrupts are enabled, the microprocessor is vectored to the IR3 service routine. During the IR3 routine, IR1 asserts a request. Since IR1 has higher priority than IR3, an interrupt is generated. However, it is not acknowledged because the microprocessor disabled interrupts in response to the IR3 interrupt. The IR1 interrupt is not acknowledged until the "Enable Interrupts" instruction is executed. Thus the IR3 routine has a "protected" section of code over which no interrupts (except non-maskable) are allowed. The IR1 routine has no such "protected" section since an "Enable Interrupts" instruction is the first one in its service routine. Note that in this example the IR1 request must stay high until it is acknowledged. This is covered in more depth in the "Interrupt Triggering" section.

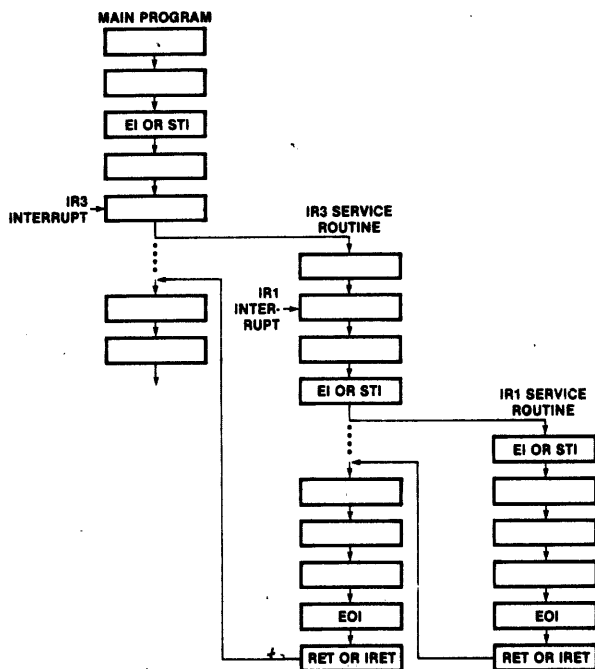


Figure 12. Fully Nested Mode Example (MCS 8085™ or MCS 8688™)

What is happening to the ISR register? While in the main program, no ISR bits are set since there aren't any interrupts in service. When the IR3 interrupt is acknowledged, the ISR3 bit is set. When the IR1 interrupt is acknowledged, both the ISR1 and the ISR3 bits are set, indicating that neither routine is complete. At this time, only IR0 could generate an interrupt since it is the only input with a higher priority than those previously in service. To terminate the IR1 routine, the routine must inform the 8259A that it is complete by resetting its ISR bit. It does this by executing an EOI command. A "return" instruction then transfers execution back to

the IR3 routine. This allows IR0-IR2 to interrupt the IR3 routine again, since ISR3 is the highest ISR bit set. No further interrupts occur in the example so the EOI command resets ISR3 and the "return" instruction causes the main program to resume at its pre-interrupt location, ending the example.

A single 8259A is essentially always in the fully nested mode unless certain programming conditions disturb it. The following programming conditions can cause the 8259A to go out of the high to low priority structure of the fully nested mode.

- The automatic EOI mode
- The special mask mode
- A slave with a master not in the special fully nested mode

These modes will be covered in more detail later, however, they are mentioned now so the user can be aware of them. As long as these program conditions aren't inacted, the fully nested mode remains undisturbed.

End of Interrupt

Upon completion of an interrupt service routine the 8259A needs to be notified so its ISR can be updated. This is done to keep track of which interrupt levels are in the process of being serviced and their relative priorities. Three different End-Of-Interrupt (EOI) formats are available for the user. These are: the non-specific EOI command, the specific EOI command, and the automatic EOI Mode. Selection of which EOI to use is dependent upon the interrupt operations the user wishes to perform.

Non-Specific EOI Command

A non-specific EOI command sent from the microprocessor lets the 8259A know when a service routine has been completed, without specification of its exact interrupt level. The 8259A automatically determines the interrupt level and resets the correct bit in the ISR.

To take advantage of the non-specific EOI the 8259A must be in a mode of operation in which it can predetermine in-service routine levels. For this reason the non-specific EOI command should only be used when the most recent level acknowledged and serviced is always the highest priority level. When the 8259A receives a non-specific EOI command, it simply resets the highest priority ISR bit, thus confirming to the 8259A that the highest priority routine of the routines in service is finished.

The main advantage of using the non-specific EOI command is that IR level specification isn't necessary as in the "Specific EOI Command", covered shortly. However, special consideration should be taken when deciding to use the non-specific EOI. Here are two program conditions in which it is best not used:

- Using the set priority command within an interrupt service routine.
- Using a special mask mode.

These conditions are covered in more detail in their own sections, but are listed here for the users reference.

Specific EOI Command

A specific EOI command sent from the microprocessor lets the 8259A know when a service routine of a particular interrupt level is completed. Unlike a non-specific EOI command, which automatically resets the highest priority ISR bit, a specific EOI command specifies an exact ISR bit to be reset. One of the eight IR levels of the 8259A can be specified in the command.

The reason the specific EOI command is needed, is to reset the ISR bit of a completed service routine whenever the 8259A isn't able to automatically determine it. An example of this type of situation might be if the priorities of the interrupt levels were changed during an interrupt routine ("Specific Rotation"). In this case, if any other routines were in service at the same time, a non-specific EOI might reset the wrong ISR bit. Thus the specific EOI command is the best bet in this case, or for that matter, any time in which confusion of interrupt priorities may exist. The specific EOI command can be used in all conditions of 8259A operation, including those that prohibit non-specific EOI command usage.

Automatic EOI Mode

When programmed in the automatic EOI mode, the microprocessor no longer needs to issue a command to notify the 8259A it has completed an interrupt routine. The 8259A accomplishes this by performing a non-specific EOI automatically at the trailing edge of the last INTA pulse (third pulse in MCS-80/85, second in MCS-86).

The obvious advantage of the automatic EOI mode over the other EOI command is no command has to be issued. In general, this simplifies programming and lowers code requirements within interrupt routines.

However, special consideration should be taken when deciding to use the automatic EOI mode because it disturbs the fully nested mode. In the automatic EOI mode the ISR bit of a routine in service is reset right after it's acknowledged, thus leaving no designation in the ISR that a service routine is being executed. If any interrupt request occurs during this time (and interrupts are enabled) it will get serviced regardless of its priority, low or high. The problem of "over nesting" may also happen in this situation. "Over nesting" is when an IR input keeps interrupting its own routine, resulting in unnecessary stack pushes which could fill the stack in a worst case condition. This is not usually a desired form of operation!

So what good is the automatic EOI mode with problems like those just covered? Well, again, like the other EOIs, selection is dependent upon the application. If interrupts are controlled at a predetermined rate, so as not to cause the problems mentioned above, the automatic EOI mode works perfect just the way it is. However, if interrupts happen sporadically at an indeterminate rate, the automatic EOI mode should only be used under the following guideline:

- When using the automatic EOI mode with an indeterminate interrupt rate, the microprocessor should keep its interrupt request input disabled during execution of service routines.

By doing this, higher priority interrupt levels will be serviced only after the completion of a routine in service. This guideline restores the fully nested structure in regards to the IRR; however, a routine in-service can't be interrupted.

Automatic Rotation — Equal Priority

Automatic rotation of priorities serves in applications where the interrupting devices are of equal priority, such as communications channels. The concept is that once a peripheral is serviced, all other equal priority peripherals should be given a chance to be serviced before the original peripheral is serviced again. This is accomplished by automatically assigning a peripheral the lowest priority after being serviced. Thus, in worst case, the device would have to wait until all other devices are serviced before being serviced again.

There are two methods of accomplishing automatic rotation. One is used in conjunction with the non-specific EOI, "rotate on non-specific EOI command". The other is used with the automatic EOI mode, "rotate in automatic EOI mode".

Rotate on Non-Specific EOI Command

When the rotate on non-specific EOI command is issued, the highest ISR bit is reset as in a normal non-specific EOI command. After it's reset though, the corresponding IR level is assigned lowest priority. Other IR priorities rotate to conform to the fully nested mode based on the newly assigned low priority.

Figures 13A and B show how the rotate on non-specific EOI command effects the interrupt priorities. Let's assume the IR priorities were assigned with IR0 the highest and IR7 the lowest, as in 13A. IR6 and IR4 are already in service but neither is completed. Being the higher priority routine, IR4 is necessarily the routine being executed. During the IR4 routine a rotate on non-specific EOI command is executed. When this happens, bit 4 in the ISR is reset. IR4 then becomes the lowest priority and IR5 becomes the highest as in 13B.

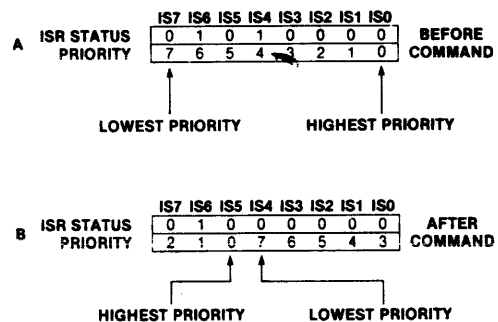


Figure 13. A-B. Rotate on Non-specific EOI Command Example

Rotate in Automatic EOI Mode

The rotate in automatic EOI mode works much like the rotate on non-specific EOI command. The main difference is that priority rotation is done automatically after

the last \overline{INTA} pulse of an interrupt request. To enter or exit this mode a rotate-in-automatic-EOI set command and rotate-in-automatic-EOI clear command is provided. After that, no commands are needed as with the normal automatic EOI mode. However, it must be remembered, when using any form of the automatic EOI mode, special consideration should be taken. Thus, the guideline for the automatic EOI mode also stands for the rotate in automatic EOI mode.

Specific Rotation — Specific Priority

Specific rotation gives the user versatile capabilities in interrupt controlled operations. It serves in those applications in which a specific device's interrupt priority must be altered. As opposed to automatic rotation which automatically sets priorities, specific rotation is completely user controlled. That is, the user selects which interrupt level is to receive lowest or highest priority. This can be done during the main program or within interrupt routines. Two specific rotation commands are available to the user, the "set priority command" and the "rotate on specific EOI command."

Set Priority Command

The set priority command allows the programmer to assign an IR level the lowest priority. All other interrupt levels will conform to the fully nested mode based on the newly assigned low priority.

An example of how the set-priority command works is shown in Figures 14A and 14B. These figures show the status of the ISR and the relative priorities of the interrupt levels before and after the set priority command. Two interrupt routines are shown to be in service in Figure 14A. Since IR2 is the highest priority, it is necessarily the routine being executed. During the IR2 routine, priorities are altered so that IR5 is the highest. This is done simply by issuing the set priority command to the 8259A. In this case, the command specifies IR4 as being the lowest priority. The result of this set priority command is shown in Figure 14B. Even though IR7 now has higher priority than IR2, it won't be acknowledged until the IR2 routine is finished (via EOI). This is because priorities are only resolved upon an interrupt request or an interrupt acknowledge sequence. If a higher priority request occurs during the IR2 routine, then priorities are resolved and the highest will be acknowledged.

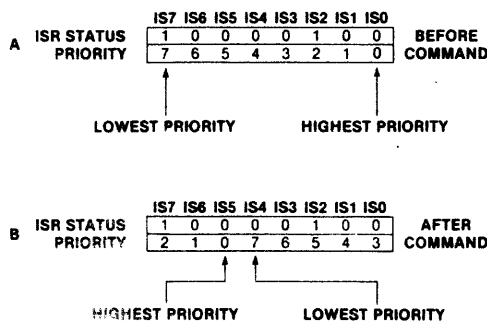


Figure 14. A-B. Set Priority Command Example

When completing a service routine in which the set priority command is used, the correct EOI must be issued. The non-specific EOI command shouldn't be used in the same routine as a set priority command. This is because the non-specific EOI command resets the highest ISR bit, which, when using the set priority command, is not always the most recent routine in service. The automatic EOI mode, on the other hand, can be used with the set priority command. This is because it automatically performs a non-specific EOI before the set priority command can be issued. The specific EOI command is the best bet in most cases when using the set priority command within a routine. By resetting the specific ISR bit of a routine being completed, confusion is eliminated.

Rotate on Specific EOI Command

The rotate on specific EOI command is literally a combination of the set priority command and the specific EOI command. Like the set priority command, a specified IR level is assigned lowest priority. Like the specific EOI command, a specified level will be reset in the ISR. Thus the rotate on specific EOI command accomplishes both tasks in only one command.

If it is not necessary to change IR priorities prior to the end of an interrupt routine, then this command is advantageous. For an EOI command must be executed anyway (unless in the automatic EOI mode), so why not do both at the same time?

Interrupt Masking

Disabling or enabling interrupts can be done by other means than just controlling the microprocessor's interrupt request pin. The 8259A has an IMR (Interrupt Mask Register) which enhances interrupt control capabilities. Rather than all interrupts being disabled or enabled at the same time, the IMR allows individual IR masking. The IMR is an 8-bit register, bits 0-7 directly correspond to IR0-IR7. Any IR input can be masked by writing to the IMR and setting the appropriate bit. Likewise, any IR input can be enabled by clearing the correct IMR bit.

There are various uses for masking off individual IR inputs. One example is when a portion of a main routine wishes only to be interrupted by specific interrupts. Another might be disabling higher priority interrupts for a portion of a lower priority service routine. The possibilities are many.

When an interrupt occurs while its IMR bit is set, it isn't necessarily forgotten. For, as stated earlier, the IMR acts only on the output of the IRR. Even with an IR input masked it is still possible to set the IRR. Thus, when resetting an IMR, if its IRR bit is set it will then generate an interrupt. This is providing, of course, that other priority factors are taken into consideration and the IR request remains active. If the IR request is removed before the IMR is reset, no interrupt will be acknowledged.

Special Mask Mode

In various cases, it may be desirable to enable interrupts of a lower priority than the routine in service. Or, in other words, allow lower priority devices to generate interrupts. However, in the fully nested mode, all IR levels of

priority below the routine in service are inhibited. So what can be done to enable them?

Well, one method could be using an EOI command before the actual completion of a routine in service. But beware, doing this may cause an "over nesting" problem, similar to in the automatic EOI mode. In addition, resetting an ISR bit is irreversible by software control, so lower priority IR levels could only be later disabled by setting the IMR.

A much better solution is the special mask mode. Working in conjunction with the IMR, the special mask mode enables interrupts from all levels except the level in service. This is done by masking the level that is in service and then issuing the special mask mode command. Once the special mask mode is set, it remains in effect until reset.

Figure 15 shows how to enable lower priority interrupts by using the Special Mask Mode (SMM). Assume that IR0 has highest priority when the main program is interrupted by IR4. In the IR4 service routine an enable interrupt instruction is executed. This only allows higher priority interrupt requests to interrupt IR4 in the normal fully nested mode. Further in the IR4 routine, bit 4 of the IMR is masked and the special mask mode is entered. Priority operation is no longer in the fully nested mode. All interrupt levels are enabled except for IR4. To leave the special mask mode, the sequence is executed in reverse.

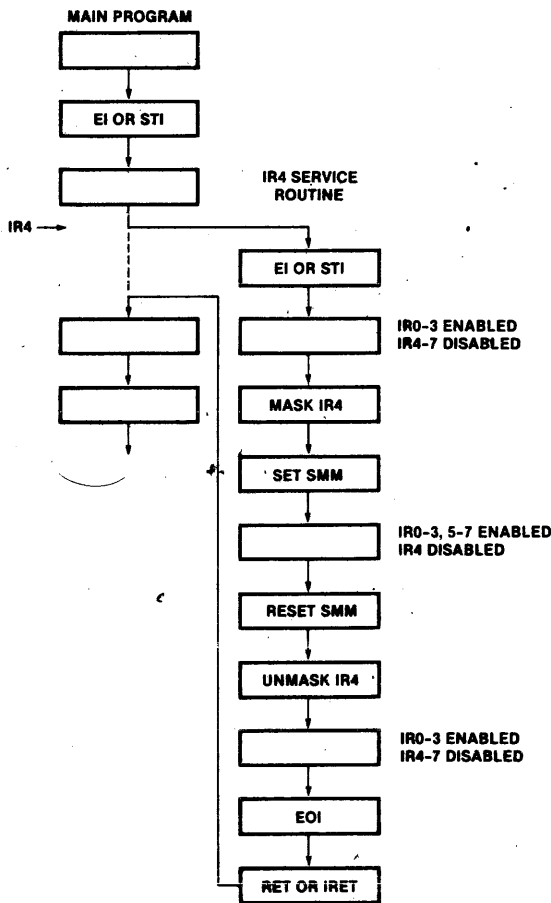


Figure 15. Special Mask Mode Example (MCS 80/85™ or MCS 86/88™)

Precautions must be taken when exiting an interrupt service routine which has used the special mask mode. A non-specific EOI command can't be used when in the special mask mode. This is because a non-specific won't clear an ISR bit of an interrupt which is masked when in the special mask mode. In fact, the bit will appear invisible. If the special mask mode is cleared before an EOI command is issued a non-specific EOI command can be used. This could be the case in the example shown in Figure 15, but, to avoid any confusion it's best to use the specific EOI whenever using the special mask mode.

It must be remembered that the special mask mode applies to all masked levels when set. Take, for instance, IR1 interrupting IR4 in the previous example. If this happened while in the special mask mode, and the IR1 routine masked itself, all interrupts would be enabled except IR1 and IR4 which are masked.

3.3 INTERRUPT TRIGGERING

There are two classical ways of sensing an active interrupt request: a level sensitive input or an edge sensitive input. The 8259A gives the user the capability for either method with the edge triggered mode and the level triggered mode. Selection of one of these interrupt triggering methods is done during the programmed initialization of the 8259A.

Level Triggered Mode

When in the level triggered mode the 8259A will recognize any active (high) level on an IR input as an interrupt request. If the IR input remains active after an EOI command has been issued (resetting its ISR bit), another interrupt will be generated. This is providing of course, the processor INT pin is enabled. Unless repetitious interrupt generation is desired, the IR input must be brought to an inactive state before an EOI command is issued in its service routine. However, it must not go inactive so soon that it disobeys the necessary timing requirements shown in Figure 16. Note that the request on the IR input must remain until after the falling edge of the first INTA pulse. If on any IR input, the request goes inactive before the first INTA pulse, the 8259A will respond as if IR7 was active. In any design in which there's a possibility of this happening, the IR7 default feature can be used as a safeguard. This can be accomplished by using the IR7 routine as a "clean-up routine" which might recheck the 8259A status or merely return program execution to its pre-interrupt location.

Depending upon the particular design and application, the level triggered mode has a number of uses. For one, it provides for repetitious interrupt generation. This is useful in cases when a service routine needs to be continually executed until the interrupt request goes inactive. Another possible advantage of the level triggered mode is it allows for "wire-OR'ed" interrupt requests. That is, a number of interrupt requests using the same IR input. This can't be done in the edge triggered mode, for if a device makes an interrupt request while the IR input is high (from another request), its transition will be "shadowed". Thus the 8259A won't recognize further interrupt requests because its IR input is already high. Note that when a "wire-OR'ed" scheme is used, the ac-

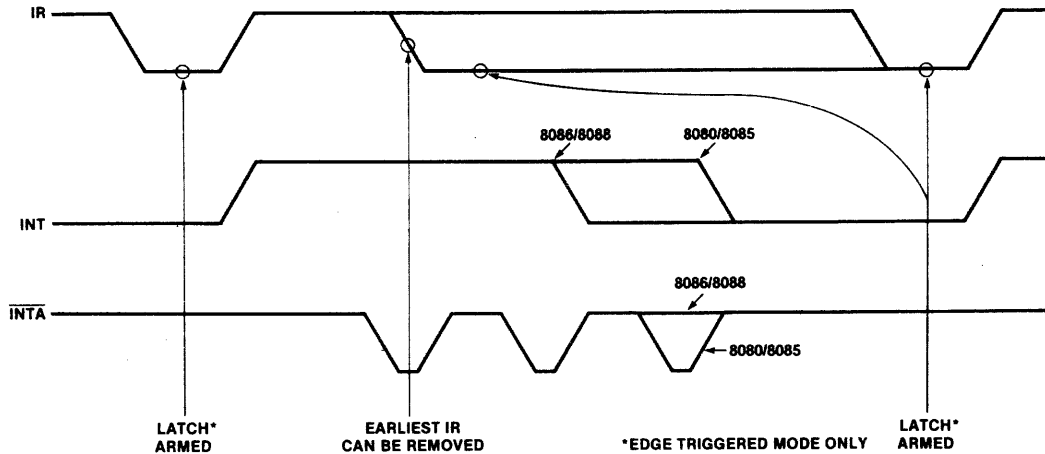


Figure 16. IR Triggering Timing Requirements

tual requesting device has to be determined by the software in the service routine.

Caution should be taken when using the automatic EOI mode and the level triggered mode together. Since in the automatic EOI mode an EOI is automatically performed at the end of the interrupt acknowledge sequence, if the processor enables interrupts while an IR input is still high, an interrupt will occur immediately. To avoid this situation interrupts should be kept disabled until the end of the service routine or until the IR input returns low.

Edge Triggered Mode

When in the edge triggered mode, the 8259A will only recognize interrupts if generated by an inactive (low) to active (high) transition on an IR input. The edge triggered mode incorporates an edge lockout method of operation. This means that after the rising edge of an interrupt request and the acknowledgement of the request, the positive level of the IR input won't generate further interrupts on this level. The user needn't worry about quickly removing the request after acknowledgement in fear of generating further interrupts as might be the case in the level triggered mode. Before another interrupt can be generated the IR input must return to the inactive state.

Referring back to Figure 16, the timing requirements for interrupt triggering is shown. Like the level triggered mode, in the edge triggered mode the request on the IR input must remain active until after the falling edge of the first INTA pulse for that particular interrupt. Unlike the level triggered mode, though, after the interrupt request is acknowledged its IRR latch is disarmed. Only after the IR input goes inactive will the IRR latch again become armed, making it ready to receive another interrupt request (in the level triggered mode, the IRR latch is always armed). Because of the way the edge triggered mode functions, it is best to use a positive level with a negative pulse to trigger the IR requests. With this type of input, the trailing edge of the pulse causes the interrupt and the maintained positive level meets the necessary timing requirements (remaining high until after the interrupt acknowledge occurs). Note that the IR7 default

feature mentioned in the "level triggered mode" section also works for the edge triggered mode.

Depending upon the particular design and application, the edge triggered mode has various uses. Because of its edge lockout operation, it is best used in those applications where repetitious interrupt generation isn't desired. It is also very useful in systems where the interrupt request is a pulse (this should be in the form of a negative pulse to the 8259A). Another possible advantage is that it can be used with the automatic EOI mode without the cautions in the level triggered mode. Overall, in most cases, the edge triggered mode simplifies operation for the user, since the duration of the interrupt request at a positive level is not usually a factor.

3.4 INTERRUPT STATUS

By means of software control, the user can interrogate the status of the 8259A. This allows the reading of the internal interrupt registers, which may prove useful for interrupt control during service routines. It also provides for a modified status poll method of device monitoring, by using the poll command. This makes the status of the internal IR inputs available to the user via software control. The poll command offers an alternative to the interrupt vector method, especially for those cases when more than 64 interrupts are needed.

Reading Interrupt Registers

The contents of each 8-bit interrupt register, IRR, ISR, and IMR, can be read to update the user's program on the present status of the 8259A. This can be a versatile tool in the decision making process of a service routine, giving the user more control over interrupt operations. Before delving into the actual process of reading the registers, let's briefly review their general descriptions:

IRR (Interrupt Request Register)	Specifies all interrupt levels requesting service.
ISR (In-Service Register)	Specifies all interrupt levels which are being serviced.
iMR (Interrupt Mask Register)	Specifies all interrupt levels that are masked.

To read the contents of the IRR or ISR, the user must first issue the appropriate read register command (read IRR or read ISR) to the 8259A. Then by applying a \overline{RD} pulse to the 8259A (an INput instruction), the contents of the desired register can be acquired. There is no need to issue a read register command every time the IRR or ISR is to be read. Once a read register command is received by the 8259A, it "remembers" which register has been selected. Thus, all that is necessary to read the contents of the same register more than once is the \overline{RD} pulse and the correct addressing ($A0 = 0$, explained in "Programming the 8259A"). Upon initialization, the selection of registers defaults to the IRR. Some caution should be taken when using the read register command in a system that supports several levels of interrupts. If the higher priority routine causes an interrupt between the read register command and the actual input of the register contents, there's no guarantee that the same register will be selected when it returns. Thus it is best in such cases to disable interrupts during the operation.

Reading the contents of the IMR is different than reading the IRR or ISR. A read register command is not necessary when reading the IMR. This is because the IMR can be addressed directly for both reading and writing. Thus all that the 8259A requires for reading the IMR is a \overline{RD} pulse and the correct addressing ($A0 = 1$, explained in "Programming the 8259A").

Poll Command

As mentioned towards the beginning of this application note, there are two methods of servicing peripherals: status polling and interrupt servicing. For most applications the interrupt service method is best. This is because it requires the least amount of CPU time, thus increasing system throughput. However, for certain applications, the status poll method may be desirable.

For this reason, the 8259A supports polling operations with the poll command. As opposed to the conventional method of polling, the poll command offers improved device servicing and increased throughput. Rather than having the processor poll each peripheral in order to find the actual device requiring service, the processor polls the 8259A. This allows the use of all the previously mentioned priority modes and commands. Additionally, both polled and interrupt methods can be used within the same program.

To use the poll command the processor must first have its interrupt request pin disabled. Once the poll command is issued, the 8259A will treat the next (\overline{CS} qualified) \overline{RD} pulse issued to it (an INput instruction) as an interrupt acknowledgment. It will then set the appropriate bit in the ISR, if there was an interrupt request, and enable a special word onto the data bus. This word shows whether an interrupt request has occurred and the highest priority level requesting service. Figure 17 shows the contents of the "poll word" which is read by the processor. Bits $W0-W2$ convey the binary code of the highest priority level requesting service. Bit I designates whether or not an interrupt request is present. If an interrupt request is present, bit I will equal 1. If there isn't an interrupt request at all, bit I will equal 0 and bits $W0-W2$ will be set to ones. Service to the requesting device is achieved by software decoding the poll word and branching to the appropriate service routine. Each

time the 8259A is to be polled, the poll command must be written before reading the poll word.

The poll command is useful in various situations. For instance, it's a good alternative when memory is very limited, because an interrupt-vector table isn't needed. Another use for the poll command is when more than 64 interrupt levels are needed (64 is the limit when cascading 8259's). The only limit of interrupts using the poll command is the number of 8259's that can be addressed in a particular system. Still another application of the poll command might be when the INT or \overline{INTA} signals are not available. This might be the case in a large system where a processor on one card needs to use an 8259A on a different card. In this instance, the poll command is the only way to monitor the interrupt devices and still take advantage of the 8259A's prioritizing features. For those cases when the 8259A is using the poll command only and not the interrupt method, each 8259A must receive an initialization sequence (interrupt vector). This must be done even though the interrupt vector features of the 8259A are not used. In this case, the interrupt vector specified in the initialization sequence could be a "fake".

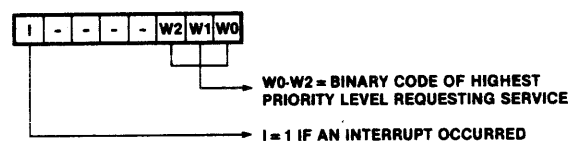


Figure 17. Poll Word

3.5 INTERRUPT CASCADING

As mentioned earlier, more than one 8259A can be used to expand the priority interrupt scheme to up to 64 levels without additional hardware. This method for expanded interrupt capability is called "cascading". The 8259A supports cascading operations with the cascade mode. Additionally, the special fully nested mode and the buffered mode are available for increased flexibility when cascading 8259A's in certain applications.

Cascade Mode

When programmed in the cascade mode, basic operation consists of one 8259A acting as a master to the others which are serving as slaves. Figure 18 shows a system containing a master and two slaves, providing a total of 22 interrupt levels.

A specific hardware set-up is required to establish operation in the cascade mode. With Figure 18 as a reference, note that the master is designated by a high on the SP/\overline{EN} pin, while the SP/\overline{EN} pins of the slaves are grounded (this can also be done by software, see buffered mode). Additionally, the INT output pin of each slave is connected to an IR input pin of the master. The $CAS0-2$ pins for all 8259A's are paralleled. These pins act as outputs when the 8259A is a master and as inputs for the slaves. Serving as a private 8259A bus, they control which slave has control of the system bus for interrupt vectoring operation with the processor. All other pins are connected as in normal operation (each 8259A receives an \overline{INTA} pulse).

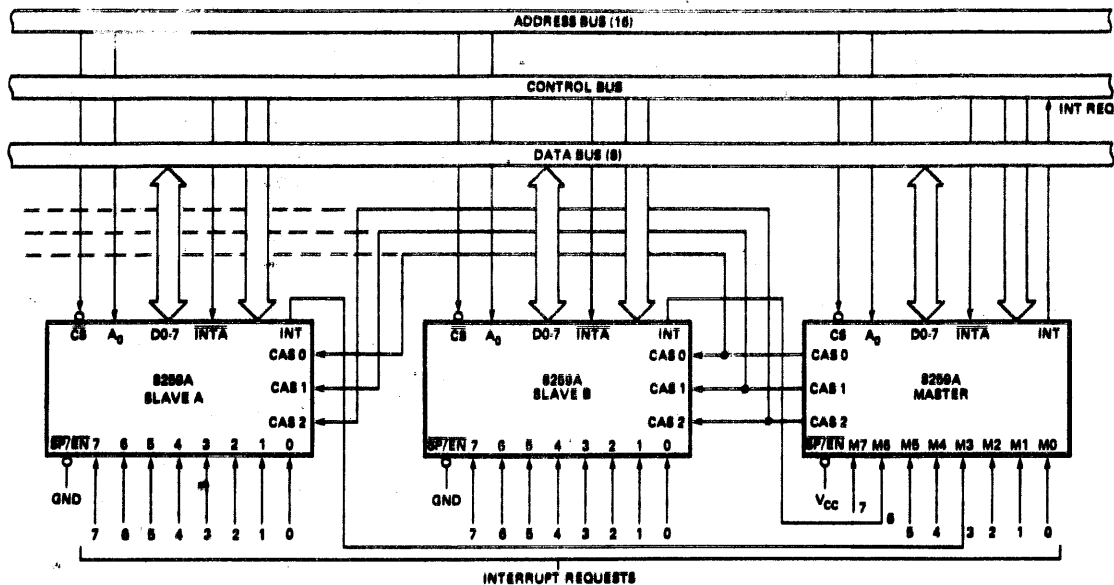


Figure 18. Cascaded 8259A's 22 Interrupt Levels

Besides hardware set-up requirements, all 8259A's must be software programmed to work in the cascade mode. Programming the cascade mode is done during the initialization of each 8259A. The 8259A that is selected as master must receive specification during its initialization as to which of its IR inputs are connected to a slave's INT pin. Each slave 8259A, on the other hand, must be designated during its initialization with an ID (0 through 7) corresponding to which of the master's IR inputs its INT pin is connected to. This is all necessary so the CAS0-2 pins of the masters will be able to address each individual slave. Note that as in normal operation, each 8259A must also be initialized to give its IR input a unique interrupt vector. More detail on the necessary programming of the cascade mode is explained in "Programming the 8259A".

Now, with background information on both hardware and software for the cascade mode, let's go over the sequence of events that occur during a valid interrupt request from a slave. Suppose a slave IR input has received an interrupt request. Assuming this request is higher priority than other requests and in-service levels on the slave, the slave's INT pin is driven high. This signals the master of the request by causing an interrupt request on a designated IR pin of the master. Again, assuming that this request to the master is higher priority than other master requests and in-service levels (possibly from other slaves), the master's INT pin is pulled high, interrupting the processor.

The interrupt acknowledge sequence appears to the processor the same as the non-cascading interrupt acknowledge sequence; however, it's different among the 8259A's. The first INTA pulse is used by all the 8259A's for internal set-up purposes and, if in the 8080/8085 mode, the master will place the CALL opcode on the data bus. The first INTA pulse also signals the master to place the requesting slave's ID code on the CAS lines. This turns control over to the slave for the rest of the interrupt acknowledge sequence, placing the

appropriate pre-programmed interrupt vector on the data bus, completing the interrupt request.

During the interrupt acknowledge sequence, the corresponding ISR bit of both the master and the slave get set. This means two EOI commands must be issued (if not in the automatic EOI mode), one for the master and one for the slave.

Special consideration should be taken when mixed interrupt requests are assigned to a master 8259A; that is, when some of the master's IR inputs are used for slave interrupt requests and some are used for individual interrupt requests. In this type of structure, the master's IRO must not be used for a slave. This is because when an IR input that isn't initialized as a slave receives an interrupt request, the CAS0-2 lines won't be activated, thus staying in the default condition addressing for IRO (slave IRO). If a slave is connected to the master's IRO when a non-slave interrupt occurs on another master IR input, erroneous conditions may result. Thus IRO should be the last choice when assigning slaves to IR inputs.

Special Fully Nested Mode

Depending on the application, changes in the nested structure of the cascade mode may be desired. This is because the nested structure of a slave 8259A differs from that of the normal fully nested mode. In the cascade mode, if a slave receives a higher priority interrupt request than one which is in service (through the same slave), it won't be recognized by the master. This is because the master's ISR bit is set, ignoring all requests of equal or lower priority. Thus, in this case, the higher priority slave interrupt won't be serviced until after the master's ISR bit is reset by an EOI command. This is most likely after the completion of the lower priority routine.

If the user wishes to have a truly fully nested structure within a slave 8259A, the special fully nested mode should be used. The special fully nested mode is pro-

grammed in the master only. This is done during the master's initialization. In this mode the master will ignore only those interrupt requests of lower priority than the set ISR bit and will respond to all requests of equal or higher priority. Thus if a slave receives a higher priority request than one in service, it will be recognized. To insure proper interrupt operation when using the special fully nested mode, the software must determine if any other slave interrupts are still in service before issuing an EOI command to the master. This is done by resetting the appropriate slave ISR bit with an EOI and then reading its ISR. If the ISR contains all zeros, there aren't any other interrupts from the slave in service and an EOI command can be sent to the master. If the ISR isn't all zeros, an EOI command shouldn't be sent to the master. Clearing the master's ISR bit with an EOI command while there are still slave interrupts in service would allow lower priority interrupts to be recognized at the master. An example of this process is shown in the second application in the "Applications Examples" section.

Buffered Mode

The buffered mode is useful in large systems where buffering is required on the data bus. Although not limited to only 8259A cascading, it's most pertinent in this use. In the buffered mode, whenever the 8259A's data bus output is enabled, its $\overline{SP/EN}$ pin will go low. This signal can be used to enable data transfer through a buffer transceiver in the required direction.

Figure 19 shows a conceptual diagram of three 8259A's in cascade, each slave is controlling an individual 8286 8-bit bidirectional bus driver by means of the buffered mode. Note the pull-up on the $\overline{SP/EN}$. It is used to enable data transfer to the 8259A for its initial programming. When data transfer is to go from the 8259A to the processor, $\overline{SP/EN}$ will go low; otherwise, it will be high.

A question should arise, however, from the fact that the $\overline{SP/EN}$ pin is used to designate a master from a slave;

how can it be used for both master-slave selection and buffer control? The answer to this is the provision for software programmable master-slave selection when in the buffer mode. The buffered mode is selected during each 8259A's initialization. At the same time, the user can assign each individual 8259A as a master or slave (see "Programming the 8259A").

4. PROGRAMMING THE 8259A

Programming the 8259A is accomplished by using two types of command words: Initialization Command Words (ICWs) and Operational Command Words (OCWs). All the modes and commands explained in the previous section, "Operation of the 8259A", are programmable using the ICWs and OCWs (see Appendix A for cross reference). The ICWs are issued from the processor in a sequential format and are used to set-up the 8259A in an initial state of operation. The OCWs are issued as needed to vary and control 8259A operation.

Both ICWs and OCWs are sent by the processor to the 8259A via the data bus (8259A $\overline{CS}=0$, $\overline{WR}=0$). The 8259A distinguishes between the different ICWs and OCWs by the state of its A0 pin (controlled by processor addressing), the sequence they're issued in (ICWs only), and some dedicated bits among the ICWs and OCWs. Those bits which are dedicated are indicated so by fixed values (0 or 1) in the corresponding ICW or OCW programming formats which are covered shortly. Note, when issuing either ICWs or OCWs, the interrupt request pin of the processor should be disabled.

4.1 INITIALIZATION COMMAND WORDS (ICWs)

Before normal operation can begin, each 8259A in a system must be initialized by a sequence of two to four programming bytes called ICWs (Initialization Command Words). The ICWs are used to set-up the necessary conditions and modes for proper 8259A operation.

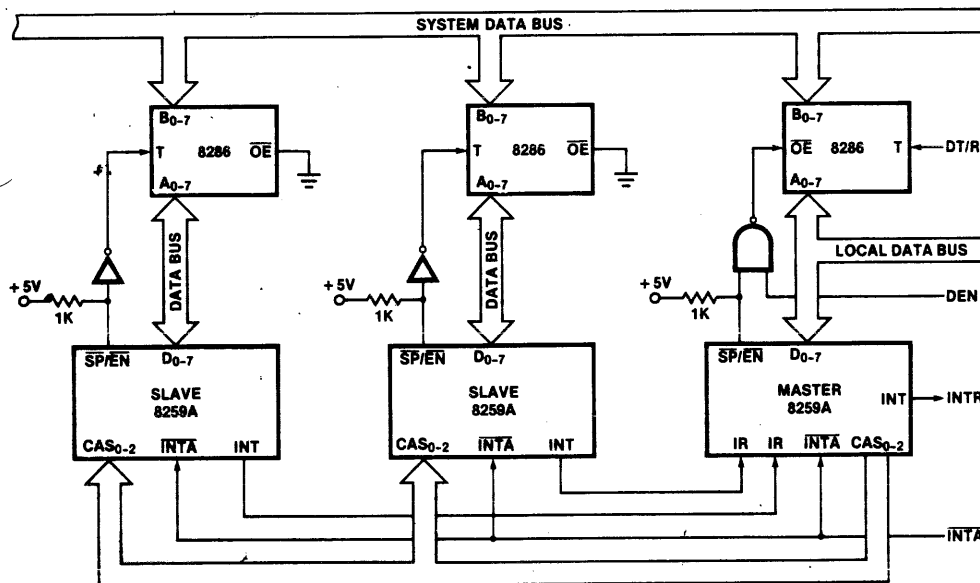


Figure 19. Cascade-Buffered Mode Example

Figure 20 shows the initialization flow of the 8259A. Both ICW1 and ICW2 must be issued for any form of 8259A operation. However, ICW3 and ICW4 are used only if designated so in ICW1. Determining the necessity and use of each ICW is covered shortly in individual groupings. Note that, once initialized, if any programming changes within the ICWs are to be made, the entire ICW sequence must be reprogrammed, not just an individual ICW.

Certain internal set-up conditions occur automatically within the 8259A after the first ICW has been issued. These are:

- A. Sequencer logic is set to accept the remaining ICWs as designated in ICW1.
- B. The ISR (In-Service Register) and IMR (Interrupt Mask Register) are both cleared.
- C. The special mask mode is reset.
- D. The rotate in automatic EOI mode flip-flop is cleared.
- E. The IRR (Interrupt Request Register) is selected for the read register command.
- F. If the IC4 bit equals 0 in ICW1, all functions in ICW4 are cleared; 8080/8085 mode is selected by default.
- G. The fully nested mode is entered with an initial priority assignment of IR0 highest through IR7 lowest.
- H. The edge sense latch of each IR priority cell is cleared, thus requiring a low to high transition to generate an interrupt (edge triggered mode effected only).

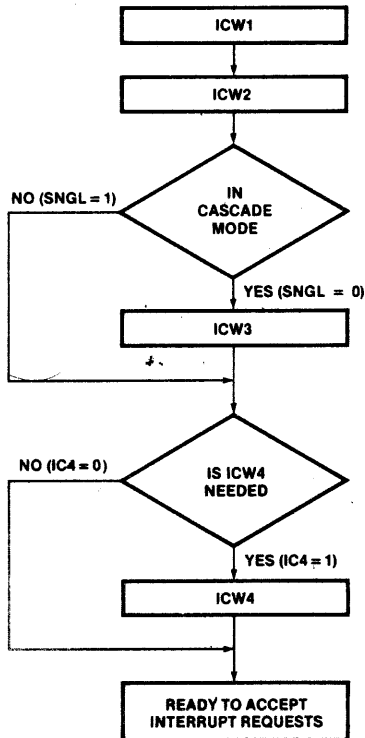
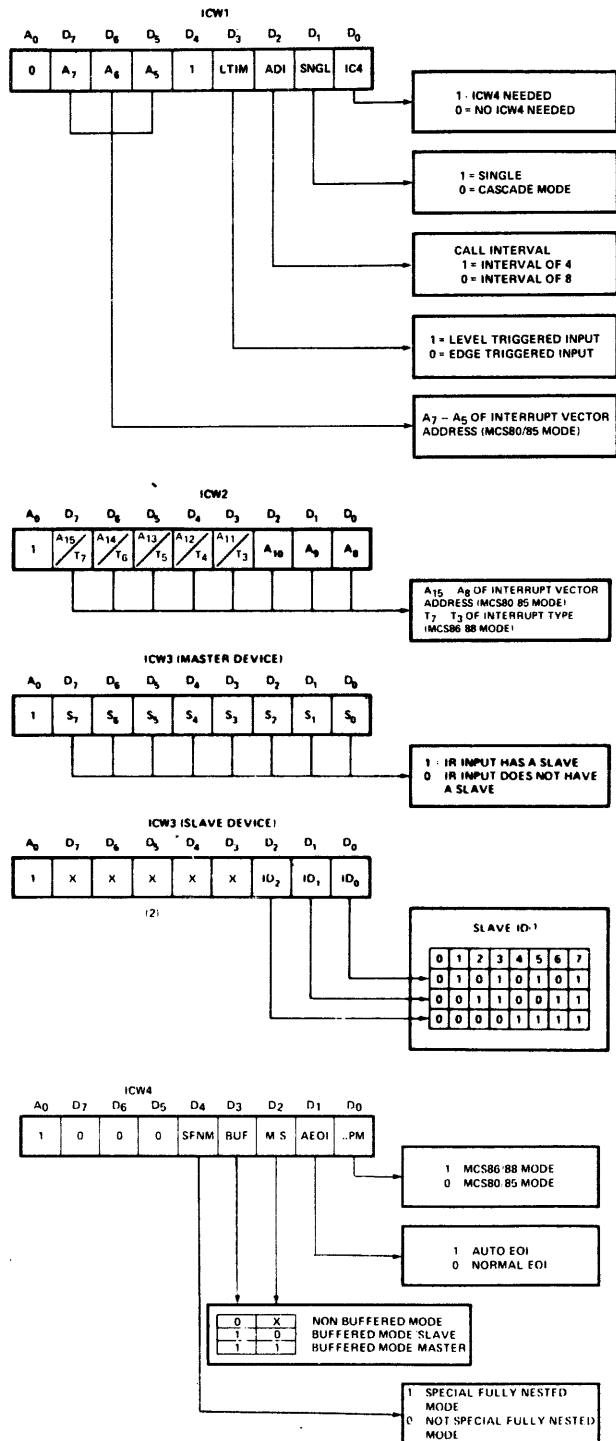


Figure 20. Initialization Flow

The ICW programming format, Figure 21, shows bit designation and a short definition of each ICW. With the ICW format as reference, the functions of each ICW will now be explained individually.



NOTE 1 SLAVE ID IS EQUAL TO THE CORRESPONDING MASTER IR INPUT
NOTE 2 X INDICATES 'DON'T CARE'

SOME OF THE TERMINOLOGY USED MAY DIFFER SLIGHTLY FROM EXISTING 8259A DATA SHEETS. THIS IS DONE TO BETTER CLARIFY AND EXPLAIN THE PROGRAMMING OF THE 8259A, THE OPERATIONAL RESULTS REMAIN THE SAME.

Figure 21. Initialization Command Words (ICWS) Programming Format

ICW1 and ICW2 -

Issuing ICW1 and ICW2 is the minimum amount of programming needed for any type of 8259A operation. The majority of bits within these two ICWs are used to designate the interrupt vector starting address. The remaining bits serve various purposes. Description of the ICW1 and ICW2 bits is as follows:

IC4: The IC4 bit is used to designate to the 8259A whether or not ICW4 will be issued. If any of the ICW4 operations are to be used, ICW4 must equal 1. If they aren't used, then ICW4 needn't be issued and IC4 can equal 0. Note that if IC4 = 0, the 8259A will assume operation in the MCS-80/85 mode.

SNGL: The SNGL bit is used to designate whether or not the 8259A is to be used alone or in the cascade mode. If the cascade mode is desired, SNGL must equal 0. In doing this, the 8259A will accept ICW3 for further cascade mode programming. If the 8259A is to be used as the single 8259A within a system, the SNGL bit must equal 1; ICW3 won't be accepted.

ADI: The ADI bit is used to specify the address interval for the MCS-80/85 mode. If a 4-byte address interval is to be used, ADI must equal 1. For an 8-byte address interval, ADI must equal 0. The state of ADI is ignored when the 8259A is in the MCS-86/88 mode.

LTIM: The LTIM bit is used to select between the two IR input triggering modes. If LTIM = 1, the level triggered mode is selected. If LTIM = 0, the edge triggered mode is selected.

A5-A15: The A5-A15 bits are used to select the interrupt vector address when in the MCS-80/85 mode. There are two programming formats that can be used to do this. Which one is implemented depends upon the selected address interval (ADI). If ADI is set for the 4-byte interval, then the 8259A will automatically insert A0-A4 (A0, A1=0 and A2, A3, A4=IR0-7). Thus A5-A15 must be user selected by programming the A5-A15 bits with the desired address. If ADI is set for the 8-byte interval, then A0-A5 are automatically inserted (A0, A1, A2=0 and A3, A4, A5=IR0-7). This leaves A6-A15 to be selected by programming the A6-A15 bits with the desired address. The state of bit 5 is ignored in the latter format.

T3-T7: The T3-T7 bits are used to select the interrupt type when the MCS-86/88 mode is used. The programming of T3-T7 selects the upper 5 bits. The lower 3 bits are automatically inserted, corresponding to the IR level causing the interrupt. The state of bits A5-A10 will be ignored when in the MCS-86/88 mode. Establishing the actual memory address of the interrupt is shown in Figure 22.

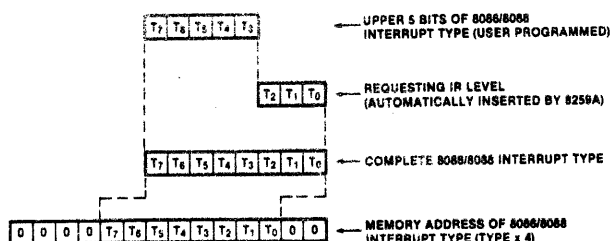


Figure 22. Establishing Memory Address of 8086/8088 Interrupt Type

ICW3

The 8259A will only accept ICW3 if programmed in the cascade mode (ICW1, SNGL=0). ICW3 is used for specific programming within the cascade mode. Bit definition of ICW3 differs depending on whether the 8259A is a master or a slave. Definition of the ICW3 bits is as follows:

S0-7 (Master): If the 8259A is a master (either when the $\overline{SP/EN}$ pin is tied high or in the buffered mode when M/S = 1 in ICW4), ICW3 bit definition is S0-7, corresponding to "slave 0-7". These bits are used to establish which IR inputs have slaves connected to them. A 1 designates a slave, a 0 no slave. For example, if a slave was connected to IR3, the S3 bit should be set to a 1. (S0) should be last choice for slave designation.

ID0-ID2 (Slave): If the 8259A is a slave (either when the $\overline{SP/EN}$ pin is low or in the buffered mode when M/S = 0 in ICW4), ICW3 bit definition is used to establish its individual identity. The ID code of a particular slave must correspond to the number of the masters IR input it is connected to. For example, if a slave was connected to IR6 of the master, the slaves ID0-2 bits should be set to ID0=0, ID1=1, and ID2=1.

ICW4

The 8259A will only accept ICW4 if it was selected in ICW1 (bit IC4 = 1). Various modes are offered by using ICW4. Bit definition of ICW4 is as follows:

μ PM: The μ PM bit allows for selection of either the MCS-80/85 or MCS-86/88 mode. If set as a 1 the MCS-86/88 mode is selected, if a 0, the MCS-80/85 mode is selected.

AEOI: The AEOI bit is used to select the automatic end of interrupt mode. If AEOI=1, the automatic end of interrupt mode is selected. If AEOI=0, it isn't selected; thus an EOI command must be used during a service routine.

M/S: The M/S bit is used in conjunction with the buffered mode. If in the buffered mode, M/S defines whether the 8259A is a master or a slave. When M/S is set to a 1, the 8259A operates as the master; when M/S is 0, it operates as a slave. If not programmed in the buffered mode, the state of the M/S bit is ignored.

BUF: The BUF bit is used to designate operation in the buffered mode, thus controlling the use of the $\overline{SP/EN}$ pin. If BUF is set to a 1, the buffered mode is programmed and $\overline{SP/EN}$ is used as a transceiver enable output. If BUF is 0, the buffered mode isn't programmed and $\overline{SP/EN}$ is used for master/slave selection. Note if ICW4 isn't programmed, $\overline{SP/EN}$ is used for master/slave selection.

SFNM: The SFNM bit designates selection of the special fully nested mode which is used in conjunction with the cascade mode. Only the master should be programmed in the special fully nested mode to assure a truly fully nested structure among the slave IR inputs. If SFNM is set to a 1, the special fully nested mode is selected; if SFNM is 0, it is not selected.

4.2 OPERATIONAL COMMAND WORD (OCWs)

Once initialized by the ICWs, the 8259A will most likely be operating in the fully nested mode. At this point, operation can be further controlled or modified by the use of OCWs (Operation Command Words). Three OCWs are available for programming various modes and commands. Unlike the ICWs, the OCWs needn't be in any type of sequential order. Rather, they are issued by the processor as needed within a program.

Figure 23, the OCW programming format, shows the bit designation and short definition of each OCW. With the OCW format as reference, the functions of each OCW will be explained individually.

OCW1

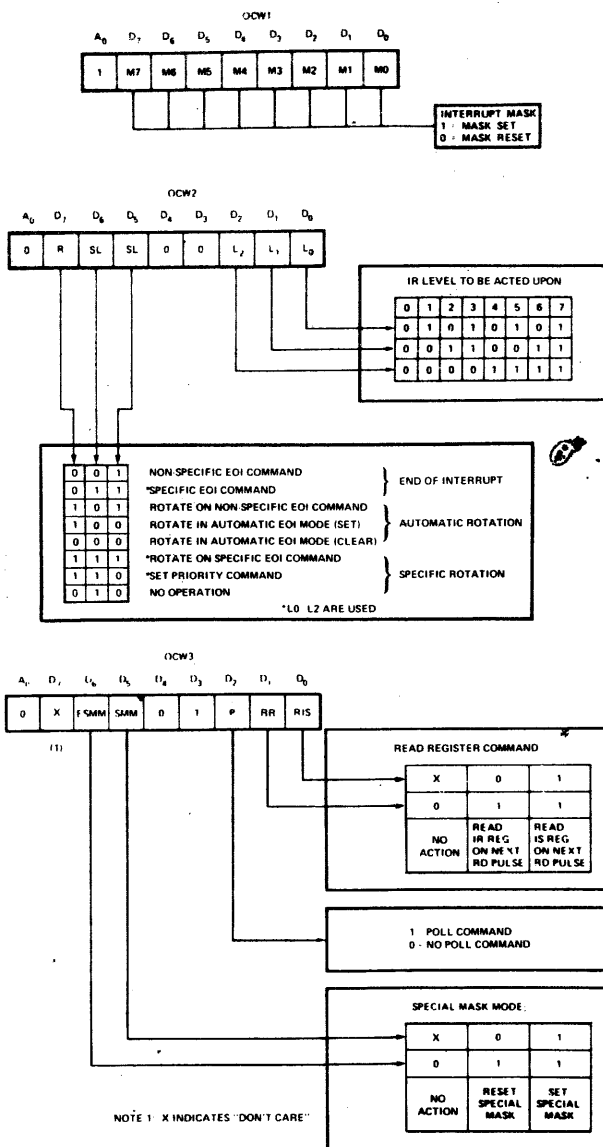
OCW1 is used solely for 8259A masking operations. It provides a direct link to the IMR (Interrupt Mask Register). The processor can write to or read from the IMR via OCW1. The OCW1 bit definition is as follows:

M0-M7: The M0-M7 bits are used to control the masking of IR inputs. If an M bit is set to a 1, it will mask the corresponding IR input. A 0 clears the mask, thus enabling the IR input. These bits convey the same meaning when being read by the processor for status update.

OCW2

OCW2 is used for end of interrupt, automatic rotation, and specific rotation operations. Associated commands and modes of these operations (with the exception of AEOI initialization), are selected using the bits of OCW2 in a combined fashion. Selection of a command or mode should be made with the corresponding table for OCW2 in the OCW programming format (Figure 20), rather than on a bit by bit basis. However, for completeness of explanation, bit definition of OCW2 is as follows:

L0-L2: The L0-L2 bits are used to designate an interrupt level (0-7) to be acted upon for the operation selected by the EOI, SL, and R bits of OCW2. The level designated will either be used to reset a specific ISR bit or to set a specific priority. The L0-L2 bits are enabled or disabled by the SL bit.



SOME OF THE TERMINOLOGY USED MAY DIFFER SLIGHTLY FROM EXISTING 8259A DATA SHEETS. THIS IS DONE TO BETTER CLARIFY AND EXPLAIN THE PROGRAMMING OF THE 8259A, THE OPERATIONAL RESULTS REMAIN THE SAME.

Figure 23. Operational Command Words (OCWs) Programming Format

EOI: The EOI bit is used for all end of interrupt commands (not automatic end of interrupt mode). If set to a 1, a form of an end of interrupt command will be executed depending on the state of the SL and R bits. If EOI is 0, an end of interrupt command won't be executed.

SL: The SL bit is used to select a specific level for a given operation. If SL is set to a 1, the L0-L2 bits are enabled. The operation selected by the EOI and R bits will be executed on the specified interrupt level. If SL is 0, the L0-L2 bits are disabled.

R: The R bit is used to control all 8259A rotation operations. If the R bit is set to a 1, a form of priority rotation will be executed depending on the state of SL and EOI bits. If R is 0, rotation won't be executed.

OCW3

OCW3 is used to issue various modes and commands to the 8259A. There are two main categories of operation associated with OCW3: interrupt status and interrupt masking. Bit definition of OCW3 is as follows:

- RIS:** The RIS bit is used to select the ISR or IRR for the read register command. If RIS is set to 1, ISR is selected. If RIS is 0, IRR is selected. The state of the RIS is only honored if the RR bit is a 1.
- RR:** The RR bit is used to execute the read register command. If RR is set to a 1, the read register command is issued and the state of RIS determines the register to be read. If RR is 0, the read register command isn't issued.
- P:** The P bit is used to issue the poll command. If P is set to a 1, the poll command is issued. If it is 0, the poll command isn't issued. The poll command will override a read register command if set simultaneously.
- SMM:** The SMM bit is used to set the special mask mode. If SMM is set to a 1, the special mask mode is selected. If it is 0, it is not selected. The state of the SMM bit is only honored if it is enabled by the ESMM bit.
- ESMM:** The ESMM bit is used to enable or disable the effect of the SMM bit. If ESMM is set to a 1, SMM is enabled. If ESMM is 0, SMM is disabled. This bit is useful to prevent interference of mode and command selections in OCW3.

5. APPLICATION EXAMPLES

In this section, the 8259A is shown in three different application examples. The first is an actual design implementation supporting an 8080A microprocessor system, "Power Fail/Auto Start with Battery Back-Up RAM". The second is a conceptual example of incorporating more than 64 interrupt levels in an 8080A or 8085A system, "78 Level Interrupt System". The third application is a conceptual design using an 8086 system, "Timer Controlled Interrupts". Although specific microprocessor systems are used in each example, these applications can be applied to either MCS-80, MCS-85, MCS-86, or MCS-88 systems, providing the necessary hardware and software changes are made. Overall, these applications should serve as a useful guide, illustrating the various procedures in using the 8259A.

5.1 POWER FAIL/AUTO-START WITH BATTERY BACK-UP RAM

The first application illustrates the 8259A used in an 8080A system, supporting a battery back-up scheme for the RAM (Random Access Memory) in a microcomputer system. Such a scheme is important in numerical and process control applications. The entire microcomputer system could be supported by a battery back-up scheme, however, due to the large amount of current usually required and the fact that most machinery is not supported by an auxiliary power source, only the state of calculations and variables usually need to be saved. In the event of a loss of power, if these items are not already stored in RAM, they can be transferred there and saved using a simple battery back-up system.

The vehicle used in this application is the Intel® SBC-80/20 Single Board Computer. An 8259A is used in the SBC-80/20 along with control lines helpful in implementing the power-down and automatic restart sequence used in a battery back-up system. The SBC-80/20 also contains user-selectable jumpers which allow the on-board RAM to be powered by a supply separate from the supply used for the non-RAM components. Also, the output of an undedicated latch is available to be connected to the IR inputs of the 8259A (the latch is cleared via an output port). In addition, an undedicated, buffered input line is provided, along with an input to the RAM decoder that will protect memory when asserted.

The additional circuitry to be described was constructed on an SBC-905 prototyping board. An SBC-635 power supply was used to power the non-RAM section of the SBC-80/20 while an external DC supply was used to simulate the back-up battery supplying power to the RAM. The SBC-635 was used since it provides an open collector ACLO output which indicates that the AC input line voltage is below 103/206 VAC (RMS).

The following is an example of a power-down and restart sequence that introduces the various power fail signals.

1. An AC power failure occurs and the ACLO goes high (ACLO is pulled up by the battery supply). This indicates that DC power will be reliable for at most 7.5 ms. The power fail circuitry generates a Power Fail Interrupt ($\overline{\text{PFI}}$) signal. This signal sets the $\overline{\text{PFI}}$ latch, which is connected to the IR0 input of the 8259A, and sets the Power Fail Sense (PFS) latch. The state of this latch will indicate to the processor, upon reset, whether it is coming up from a power failure (warm start) or if it is coming up initially (cold start).
2. The processor is interrupted by the 8259A when the PFI latch is set. This pushes the pre-power-down program counter onto the stack and calls the service routine for the IR0 input. The IR0 service routine saves the processor status and any other needed variables. The routine should end with a HALT instruction to minimize bus transitions.
3. After a predetermined length of time (5 ms in this example) the power fail circuitry generates a Memory Protect ($\overline{\text{MPRO}}$) signal. All processing for the power failure (including the interrupt response delays) must be completed within this 5 ms window. The $\overline{\text{MPRO}}$ signal ensures that spurious transitions on the system control bus caused by power going down do not alter the contents of the RAM.
4. DC power goes down.
5. AC power returns. The power-on reset circuitry on the SBC-80/20 generates a system RESET.
6. The processor reads the state of the $\overline{\text{PFS}}$ line to determine the appropriate start-up sequence. The PFS latch is cleared, the $\overline{\text{MPRO}}$ signal is removed, and the PFI latch driving IR0 is cleared by the Power Fail Sense Reset ($\overline{\text{PFSR}}$) signal. The system then continues from the pre-power-down location for a warm start by restoring the processor status and popping the pre-power-down program counter off the stack.

Figure 24 illustrates this timing.

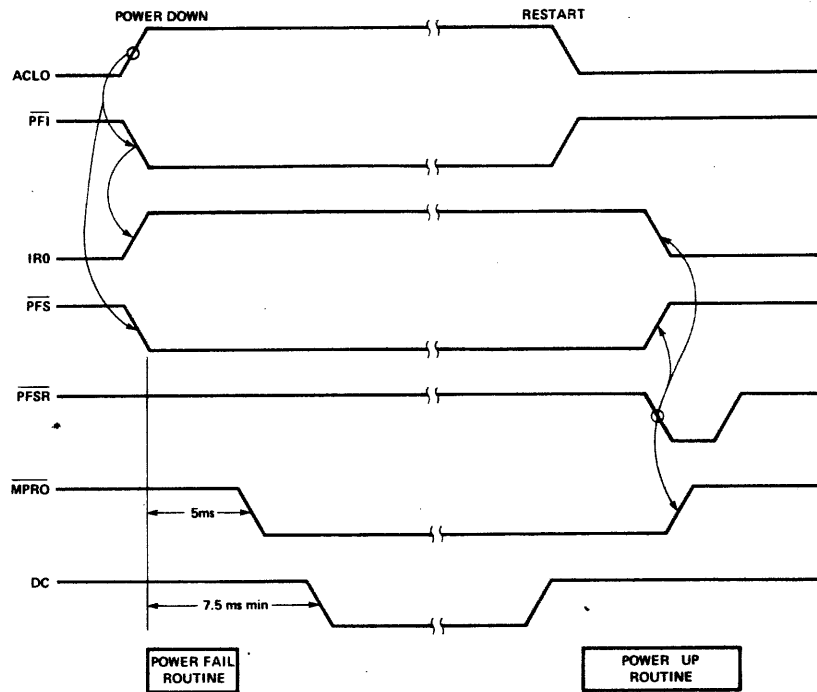


Figure 24. Power Down Restart Timing

Figure 25 shows the block diagram for the system. Notice that the RAM, the RAM decoder, and the power-down circuitry are powered by the battery supply.

The schematic of the power-down circuitry and the SBC-80/20 interface is shown in Figure 26. The design is very straightforward and uses CMOS logic to minimize the battery current requirements. The cold start switch is necessary to ensure that during a cold start, the PFS line is indicating "cold start" sense (PFS high). Thus, for

a cold start, the cold start switch is depressed during power on. After that, no further action is needed. Notice that the PFI signal sets the on-board PFI latch. The output of this latch drives the 8259A IRO input. This latch is cleared during the restart routine by executing an OUTput D4H instruction. The state of the PFS line may be read on the least significant data bus line (DB0) by executing an INput D4H instruction. An 8255 port (8255 #1, port C, bit 0) is used to control the PFSR line.

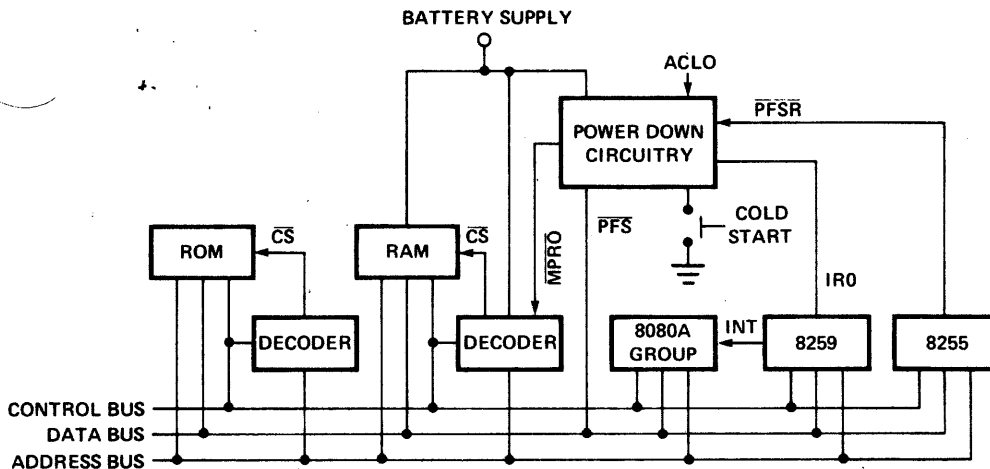


Figure 25. Block Diagram of SBC 80/20 with Power Down Circuit

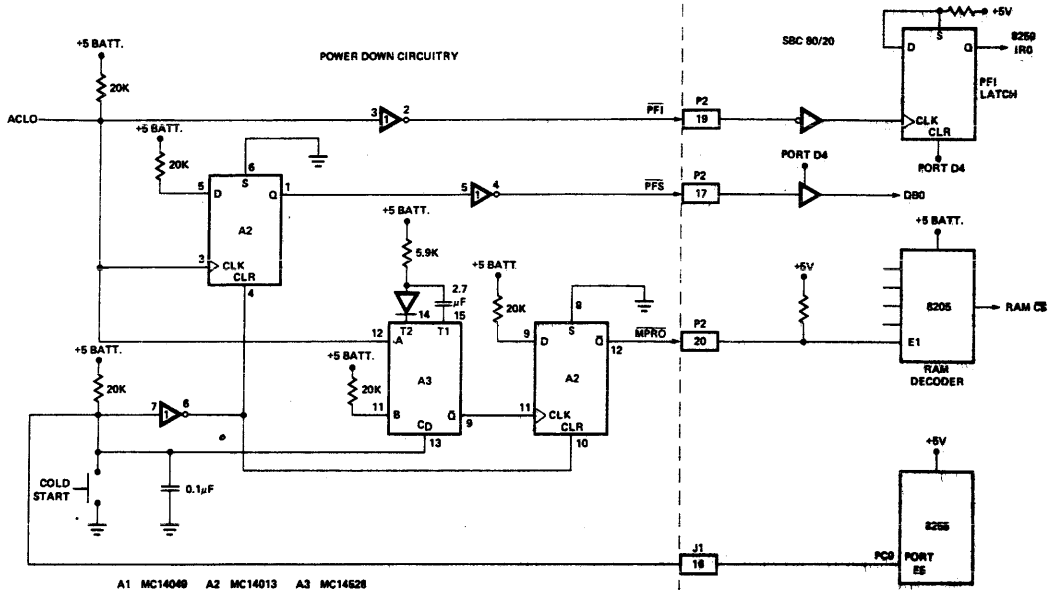


Figure 26. Power Down Circuit - SBC 80/20 Interface

The fully nested mode for the 8259A is used in its initial state to ensure the IR0 always has the highest priority. The remaining IR inputs can be used for any other purpose in the system. The only constraint is that the service routines must enable interrupts as early as possible. Obviously, this is to ensure that the power-down interrupt does not have to wait for service. If a rotating priority scheme is desired, another 8259A could be added as a slave and be programmed to operate in a rotating mode. The master would remain in the initial state of the fully nested mode so that the IR0 still remains the highest priority input.

The software to support the power-down circuitry is shown in Figure 27. The flow for each label will be discussed.

After any system reset, the processor starts execution at location 0000H (START). The PFS status is read and execution is transferred to CSTART if PFS indicates a cold start (i.e., someone is depressing the cold start switch) or WSTART if a warm start is indicated (PFS LOW). CSTART is the start of the user's program. The Stack Pointers (SP) and device initialization were included just to remind the reader that these must occur. The first EI instruction must appear after the 8259A has received its initialization sequence. The 8259A (and other devices) are initialized in the INIT subroutine.

When a power failure occurs, execution is vectored by the 8259A to REGSAV by way of the jump table at JSTART. The pre-power-down program counter is placed on the stack. REGSAV saves the processor registers and flags in the usual manner by pushing them onto the stack. Other items, such as output port status, program-

able peripheral states, etc., are pushed onto the stack at this time. The Stack Pointer (SP) could be pushed onto the stack by way of the register pair HL but the top of the stack can exist anywhere in memory and there is no way then of knowing where that is when in the power-up routine. Thus, the SP is saved at a dedicated location in RAM. It isn't really necessary to send an EOI command to the 8259A in REGSAV since power will be removed from the 8259A, but one is included for completeness. The final instruction before actually losing power is a HALT. This minimizes somewhat spurious transitions on the various busses and lets the processor die gracefully.

On reset, when a warm start is detected, execution is transferred to WSTART. WSTART activates PFSR by way of the 8255 (all outputs go low then the 8255 is initialized). In the power-down circuitry, PFSR clears the PFS latch and removes the MPRO signal which then allows access to the RAM. WSTART also clears the PFI latch which arms the 8259A IR0 input. Then the 8259A is re-initialized along with any other devices. The SP is retrieved from RAM and the processor registers and flags are restored by popping them off the stack. Interrupts are then enabled. Now the power-down program counter is on top of the stack, so executing a RETURN instruction transfers the processor to exactly where it left off before the power failure.

Aside from illustrating the usefulness of the 8259A (and the SBC-80/20) in implementing a power failure protected microcomputer system, this application should also point out a way of preserving the processor status when using interrupts.

LOC	OBJ	SLD	SOURCE	COMMENT	LOC	OBJ	SLD	SOURCE	COMMENT				
					55				ADD ANY OTHER INITIALIZATIONS HERE				
					56								
					57				RET				
					58				RETURN				
					59								
					60				POWER DOWN ROUTINE TO SAVE REGISTERS AND STATUS				
					61								
					62								
0004				8253 PORT WITH AH=0	0026	FD			63	PSW	PUSH	PSW	SAVE R PLUS FLAG
0005				8253 PORT WITH AH=1	0027	CD			64	PSW	PUSH	R	SAVE HL
0007				8253 BI CONTROL PORT	0028	DC			65	PSW	PUSH	D	SAVE DE
0006				8253 BI PORT C	0029	CC			66	PSW	PUSH	C	SAVE BC
3000				SP STORAGE IN RAM	002A	Z10000			67	LDI	H,0000H		SET TO 0 OCT SP
0001				MSB OF 8253 JUMP TABLE	002D	ZD			68	LDI	SP		SP NOW IN HL
					002E	Z0020			69	SHL	SP,SP		SAVE SP IN RAM
									70				
									71				
									72				
									73				
0000					0031	Z020			74	MVI	R,00H		NON-SPECIFIC EOI
0000	D004			KEYD PFS/ STATUS	0032	D004			75	OUT	PT50A		8253 PORT WITH AH=0
0002	1F			PFS/ ON D004 - PUT IN CARRY	0035	76			76	HL			ENABLE - GO DOWN SUCCESSFULLY
0002	D02001			PFS=HL - THEN COLD START					77				
									78				
									79				
									80				
									81				
									82				
0006	3E00			SET 8255 HL TO OUTPUT MODE	0100				83	ORG	1000H		
0006	D3E7			8255 CONTROL PORT - PFS/ GOES LOW	0100	C32600			84	JMP	1E0200H		INT
					0103	00			85	NOP			
					0104	C31030			86	JMP	20100H		INT
					0107	00			87	NOP			
					0108	C32030			88	JMP	20200H		INT
					0109	00			89	NOP			
					0110	C34030			90	JMP	20400H		INT
					0113	00			91	NOP			
					0114	C35030			92	JMP	20500H		INT
					0117	00			93	NOP			
					0118	C36030			94	JMP	20600H		INT
					0119	00			95	NOP			
					011C	C37030			96	JMP	20700H		INT
					011F	00			97	NOP			
									98				
									99				
									100				
									101				
									102				
									103				
									104				
									105				
									106				
									107				
									108				
									109				
									110				

Figure 27. Power Down and Restart Software

5.2 78 LEVEL INTERRUPT SYSTEM

The second application illustrates an interrupt structure with greater than 64 levels for an 8080A or 8085A system. In the cascade mode, the 8259A supports up to 64 levels with direct vectoring to the service routine. Extending the structure to greater than 64 levels requires polling, using the poll command. A 78 level interrupt structure is used as an illustration; however, the principles apply to systems with up to 512 levels.

To implement the 78 level structure, 3 tiers of 8259A's are used. Nine 8259A's are cascaded in the master-slave scheme, giving 64 levels at tier 2. Two additional 8259A's are connected, by way of the INT outputs, to two of the 64 inputs. The 16 inputs at tier 3, combined with the 62 remaining tier 2 inputs, give 78 total levels. The fully nested structure is preserved over all levels, although direct vectoring is supplied for only the tier 2 inputs. Software is required to vector any tier 3 requests. Figure 28 shows the tiered structure used in this example. Notice that the tier 3 8259A's are connected to the bottom level slave (SA7). The master-slaves are interconnected as shown in "Interrupt Cascading", while the tier 3 8259A's are connected as "masters"; that is, the SP/EN pins are pulled high and the CAS pins are left unconnected. Since these 8259A's are only going to be used with the poll command, no INTA is required, therefore the INTA pins are pulled high.

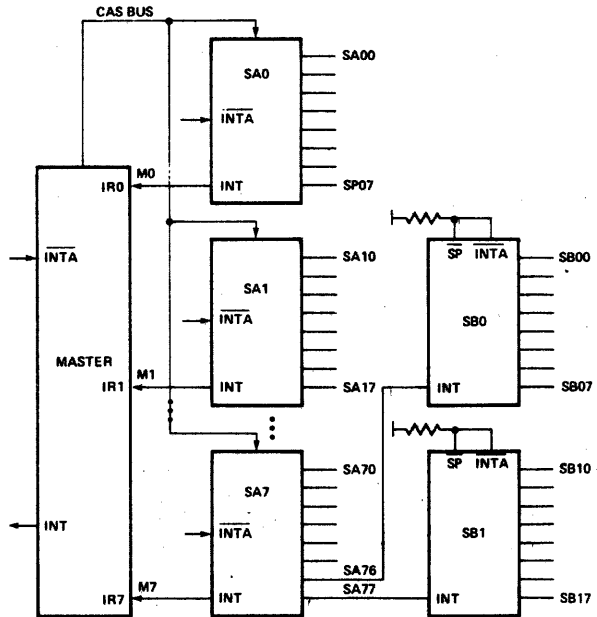


Figure 28. 78 Level Interrupt Structure

The concept used to implement the 78 levels is to directly vector to all tier 2 input service routines. If a tier 2 input contains a tier 3 8259A, the service routine for that input will poll the tier 3 8259A and branch to the tier 3 input service routine based on the poll word read after the poll command. Figure 29 shows how the jump table is organized assuming a starting location of 1000H and contiguous tables for all the tier 2 8259A's. Note that "SA35" denotes the IR5 input of the slave connected to the master IR3 input. Also note that for the normal tier 2 inputs, the jump table vectors the processor directly to the service routine for that input, while for the tier 2 inputs with 8259A's connected to their IR inputs, the processor is vectored to a service routine (i.e., SB0) which will poll to determine the actual tier 3 input requesting service. The polling routine utilizes the jump table starting at 1200H to vector the processor to the correct tier 3 service routine.

Each 8259A must receive an initialization sequence regardless of the mode. Since the tier 1 and 2 8259A's are in cascade and the special fully nested mode is used (covered shortly), all ICW's are required. The tier 3 8259A's don't require ICW3 or ICW4 since only polling will be used on them and they are connected as masters not in the cascade mode. The initialization sequence for each tier is shown in Figure 30. Notice that the master is initialized with a "dummy" jump table starting at 00H since all vectoring is done by the slaves. The tier 3 devices also receive "dummy" tables since only polling is used on tier 3.

As explained in "Interrupt Cascading", to preserve a truly fully nested mode within a slave, the master 8259A should be programmed in the special fully nested mode. This allows the master to acknowledge all interrupts at and above the level in service disregarding only those of lower priority. The special fully nested mode is programmed in the master only, so it only affects the immediate slaves (tier 2 not tier 3). To implement a fully nested structure among tier 3 slaves some special housekeeping software is required in all the tier-2-with-tier-3-slave routines. The software should simply save the state of the tier 2 IMR, mask all the lower tier 2 interrupts, then issue a specific EOI, resetting the ISR of the tier 2 interrupt level. On completion of the routine the IMR is restored.

Figure 31 shows an example flow and program for any tier 2 service routine without a tier 3 8259A. Figure 32 shows an example flow and program for any tier 2 service routine with a tier 3 8259A. Notice the reading of the ISR in both examples; this is done to determine whether or not to issue an EOI command to the master (refer to the section on "Special Fully Nested Mode" for further details).

LOCATION	8259	CODE	COMMENTS
1000 H	SA0	JMP SA00	SA00 SERVICE ROUTINE
101C H		JMP SA07	SA07 SERVICE ROUTINE
1020 H	SA1	JMP SA10	SA10 SERVICE ROUTINE
103C H		JMP SA17	SA17 SERVICE ROUTINE
			SA20-SA67 SERVICE ROUTINES
10E0 H	SA7	JMP SA70	SA70 SERVICE ROUTINE
10F8 H		JMP SB0	SB0 POLL ROUTINE
10FC H		JMP SB1	SB1 POLL ROUTINE
1200 H	SB0	JMP SB00	SB00 SERVICE ROUTINE
121C H		JMP SB07	SB07 SERVICE ROUTINE
1220 H	SB1	JMP SB10	SB10 SERVICE ROUTINE
123C H		JMP SB17	SB17 SERVICE ROUTINE

Figure 29. Jump Table Organization

```

INITIALIZATION SEQUENCE FOR 78 LEVEL INTERRUPT STRUCTURE
INITIALIZE MASTER
MINT: MVI A,15H ; ICW1, LTM=0, ADI=1, S=0, IC4=1
      OUT MPTA ; MASTER PORT A0=0
      MVI A,00H ; ICW2, DUMMY ADDRESS
      OUT MPTB ; MASTER PORT A0=1
      MVI A,0FFH ; ICW3, S7-S0=1
      OUT MPTB ; MASTER PORT A0=1
      MVI A,10H ; ICW4, SFNM=1
      OUT MPTB ; MASTER PORT A0=1

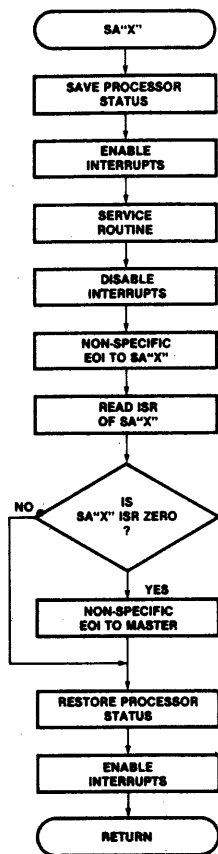
INITIALIZE SA SLAVES - X DENOTES SLAVE ID (SEE KEY)
SAXINT: MVI A,x ; SEE KEY FOR ICW1, LTM=0, ADI=1, S=0, IC4=1
        OUT SAXPTA ; SA"X" PORT A0=0
        MVI A,10H ; ICW2, ADDRESS MSB
        OUT SAXPTB ; SA"X" PORT A0=1
        MVI A,0XH ; ICW3, SA ID
        OUT SAXPTB ; SA"X" PORT A0=1
        MVI A,10H ; ICW4, SFNM=1
        OUT SAXPTB ; SA"X" PORT A0=1

REPEAT ABOVE FOR EACH SA SLAVE

INITIALIZE SB SLAVES - X DENOTES 0 or 1 (DO SB0, REPEAT FOR SB1)
SBXINT MVI A,16H ; ICW1, LTM=0, ADI=1, S=1, IC4=0
        OUT SBXPTA ; SB"X" PORT A0=0
        MVI A,00H ; ICW2, DUMMY ADDRESS
        OUT SBXPTB ; SB"X" PORT A0=1
    
```

SA"X"	a (ICW1)	JUMP TABLE START (H)
0	15	1000
1	35	1020
2	55	1040
3	75	1060
4	95	1080
5	B5	10A0
5	D5	10C0
7	F5	10E0

Figure 30. Initialization Sequence for 78 Level Interrupt Structure



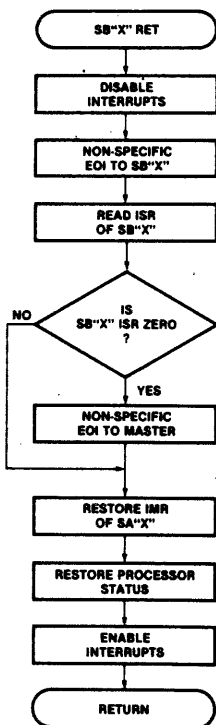
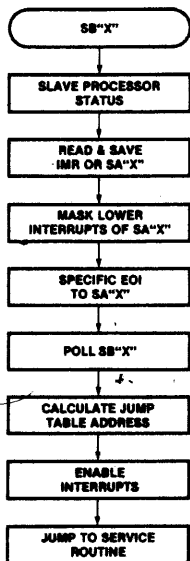
SA'X' ROUTINE - GENERAL INTERRUPT SERVICE ROUTINE FOR TIER 2 INTERRUPTS WITHOUT TIER 3 8259A

SAX: PUSH D : SAVE DE
 PUSH B : SAVE BC
 PUSH H : SAVE HL
 PUSH PSW : SAVE A, FLAGS
 EI : ENABLE INTERRUPTS

SERVICE ROUTINE GOES HERE

DI : DISABLE INTERRUPTS
 MVI 20H : OCW2, NON-SPECIFIC EOI
 OUT SAXPTA : SA'X' PORT A0 = 0
 MUI A,0BH : OCW3, READ REGISTER, ISR
 OUT SAXPTA : SA'X' PORT A0 = 0
 IN SAXPTA : SA'X' PORT A0 = 0, SA'X' ISR
 ANI 0FH : TEST FOR ZERO
 JZ SAXRSR : IF NOT ZERO, RESTORE STATUS
 MVI A,0BH : OCW2, NON-SPECIFIC EOI
 OUT MASPTA : MASTER PORT A0 = 0
 SAXRSR: POP PSW : RESTORE A, FLAGS
 POP H : RESTORE HL
 POP B : RESTORE BC
 POP D : RESTORE DE
 EI : ENABLE INTERRUPTS
 RET : RETURN

Figure 31. Example Service Routine for Tier 2 Interrupt (SA'X') without Tier 3 8259A (SB'X')



SB'X' ROUTINE - SERVICE ROUTINE FOR TIER 2 INTERRUPTS WITH TIER 3 8259A

SBX: PUSH D : SAVE DE
 PUSH B : SAVE BC
 PUSH H : SAVE HL
 PUSH PSW : SAVE A, FLAGS
 IN SAXPTB : READ SA'X' IMR
 MOV D,A : SAVE
 MVI A,XXH : MASK SA'X' LOWER IR
 OUT SAXPTB : SA'X' PORT A0 = 1
 MVI A,XXH : OCW2 SPECIFIC EOI SA'X'
 OUT SAXPTA : SA'X' PORT A0 = 1
 LXI H,1200H : JUMP TABLE START
 MVI B,00H : CLEAR B
 MVI A,0CH : OCW3, POLL COMMAND
 OUT SBXPTA : SB'X' PORT A0 = 0
 IN SBXPTA : GET POLL WORD
 ANI 07H : LIMIT TO 3 BITS
 ADD A : GET TABLE OFFSET
 ADD A : GET TABLE OFFSET
 MOV C,A : OFFSET TO C
 DAD B : HL HAS TABLE ADDRESS
 EI : ENABLE INTERRUPTS

SB'X' RET ROUTINE - FOR EOI AND MASK RESTORE AFTER SB'X' ROUTINE

SBXRET DI : DISABLE INTERRUPTS
 MVI A,20H : OCW2, NON SPECIFIC EOI
 OUT SBXPTA : SA'X' PORT A0 = 0
 MVI A,0BH : OCW3, READ REGISTER ISR
 OUT SAXPTA : SA'X' PORT A0 = 0
 IN SBXPTA : SA'X' PORT A0 = 0, ISR
 ANI 0FH : TEST FOR ZERO
 JNZ SBXRSR : IF = 0 RESTORE IMR
 MVI A,20H : OCW2, NON-SPECIFIC EOI
 OUT MASPTA : MASTER PORT A0 = 0
 SBXRSR: MOV A,D : RESTORE SA'X' IMR
 OUT SAXPTB : SA'X' PORT A0 = 1
 POP PSW : RESTORE A, FLAGS
 POP H : RESTORE HL
 POP B : RESTORE BC
 POP D : RESTORE DE
 EI : RESTORE DE
 RET : RETURN

Figure 32. Example Service Routine for Tier 2 Interrupt (SA'X') with Tier 3 8259A (SB'X')

5.3 TIMER CONTROLLED INTERRUPTS

In a large number of controller type microprocessor designs, certain timing requirements must be implemented throughout program execution. Such time dependent applications include control of keyboards, displays, CRTs, printers, and various facets of industrial control. These examples, however, are just a few of many designs which require device servicing at specific rates or generation of time delays. Trying to maintain these timing requirements by processor control alone can be costly in throughput and software complexity. So, what can be done to alleviate this problem? The answer, use the 8259A Programmable Interrupt Controller and external timing to interrupt the processor for time dependent device servicing.

This application example uses the 8259A for timer controlled interrupts in an 8086 system. External timing is done by two 8253 Programmable Interval Timers. Figure 33 shows a block diagram of the timer controlled interrupt circuitry which was built on the breadboard area of an SDK-86 (system design kit). Besides the 8259A and the 8253's, the necessary I/O decoding is also shown. The timer controlled interrupt circuitry interfaces with the SDK-86 which serves as the vehicle of operation for this design.

A short overview of how this application operates is as follows. The 8253's are programmed to generate interrupt requests at specific rates to a number of the 8259A IR inputs. The 8259A processes these requests by interrupting the 8086 and vectoring program execution to the appropriate service routine. In this example, the routines use the SDK-86 display panel to display the number of the interrupt level being serviced. These routines are merely for demonstration purposes to show the necessary procedures to establish the user's own routines in a timer controlled interrupt scheme.

Let's go over the operation starting with the actual interrupt timing generation which is done by two 8253 Programmable Interval Timers (8253 #1 and 8253 #2). Each 8253 provides three individual 16-bit counters (counters

0-2) which are software programmable by the processor. Each counter has a clock input (CLK), gate input (GATE), and an output (OUT). The output signal is based on divisions of the clock input signal. Just how or when the output occurs is determined by one of the 8253's six programmable modes, a programmable 16-bit count, and the state of the gate input.

Figure 34 shows the 8253 timing configuration used for generating interrupts to the 8259A. The SDK-86's PCLK (peripheral clock) signal provides a 400 ns period clock to CLK0 of 8253 #1. Counter 0 is used in mode 3 (square wave rate generator), and acts as a prescaler to provide the clock inputs of the other counters with a 10 ms period square wave. This 10 ms clock period made it easy to calculate exact timings for the other counters. Counter 2 of the 8253 #1 is used in mode 2 (rate generator), it is programmed to output a 10 ms pulse for every 200 pulses it receives (every 2 sec). The output of counter 2 causes an interrupt on IR1 of the 8259A. All the 8253 #2 counters are used in mode 5 (hardware triggered strobe) in which the gate input initiates counter operations. In this case the output of 8253 #1 counter 2 controls the gate of each 8253 #2 counter. When one of the 8253 #2 counters receive the 8253 #1 counter 2 output pulse on its gate, it will output a pulse (10 ms in duration) after a certain preprogrammed number of clock pulses have occurred. The programmed number of clock pulses for the 8253 #2 counters is as follows: 50 pulses (0.5 sec) for counter 0, 100 pulses (1 sec) for counter 1, and 150 pulses (1.5 sec) for counter 2. The outputs of these counters cause interrupt requests on IR2 through IR4 of the 8259A. Counter 1 of 8253 #1 is used in mode 0 (interrupt on terminal count). Unlike the other modes used which initialize operation automatically or by gate triggering, mode 0 allows software controlled counter initialization. When counter 1 of 8253 #1 is set during program execution, it will count 25 clocks (250 ms) and then pull its output high, causing an interrupt request on IR0 of the 8259A. Figure 35 shows the timing generated by the 8253's which cause interrupt request on the 8259A IR inputs.

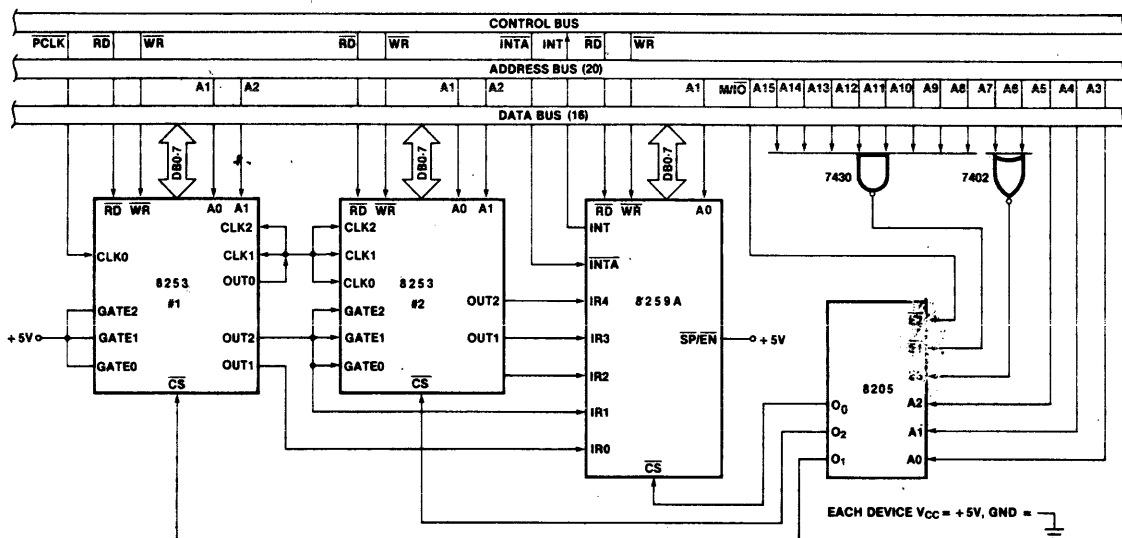


Figure 33. Timer Controlled Interrupt Circuit on SDK 86 Breadboard Area

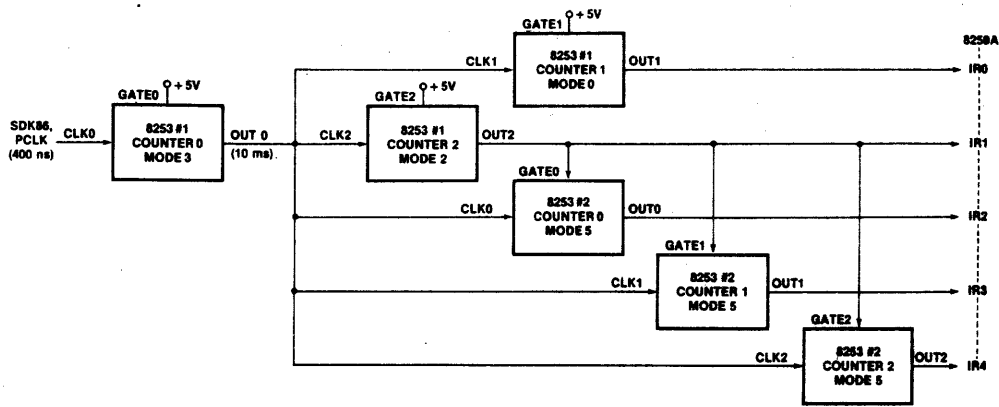


Figure 34. 8253 Timing Configuration for Timer Controlled Interrupts

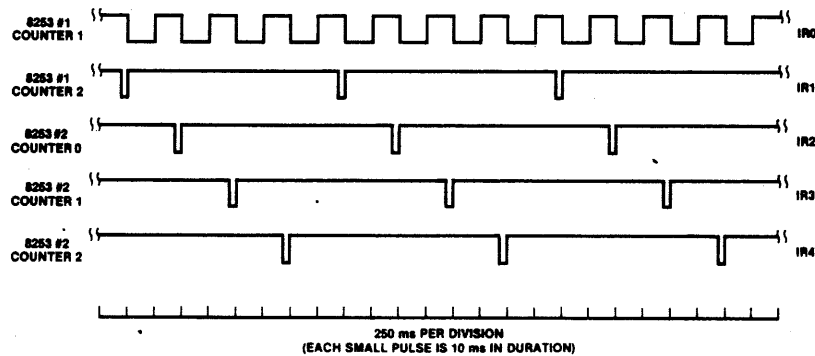


Figure 35. 8259A IR Input Signal From 8253S

There are basically two methods of timing generation that can be used in a timer controlled interrupt structure: dependent timing and independent timing. Dependent timing uses a single timing occurrence as a reference to base other timing occurrences on. On the other hand, independent timing has no mutual reference between occurrences. Industrial controller type applications are more apt to use dependent timing, whereas independent timing is prone to individual device control.

Although this application uses primarily dependent timing, independent timing is also incorporated as an example. The use of dependent timing can be seen back in Figure 34, where timing for IR2 through IR4 uses the IR1 pulse as reference. Each one of the 8253 #2 counters will generate an interrupt request a specific amount of times after the IR1 interrupt request occurs. When using the dependent method, as in this case, the IR2 through IR4 requests must occur before the next IR1 request. Independent timing is used to control the IR0 interrupt request. Note that its timing isn't controlled by any of the other IR requests. In this timer controlled interrupt configuration the dependent timing is initially set to be self running and the independent timing is software initialized. However, both methods can work either way by using the various 8253 modes to generate the same interrupt timing.

The 8259A processes the interrupts generated by the 8253's according to how it is programmed. In this application it is programmed to operate in the edge triggered mode, MCS-86/88 mode, and automatic EOI mode. In the edge triggered mode an interrupt request on an 8259A

IR input becomes active on the rising edge. With this in mind, Figure 35 shows that IR0 will generate an interrupt every half second and IR1 through IR4 will each generate an interrupt every 2 seconds spaced apart at half second intervals. Interrupt vectoring in the MCS-86/88 mode is programmed so IR0, when activated, will select interrupt type 72. This means IR1 will select interrupt type 73, IR2 interrupt type 74, and so on through IR4. Since IR5 through IR7 aren't used, they are masked off. This prevents the possibility of any accidental interrupts and rids the necessity to tie the unused IR inputs to a steady level. Figure 36 shows the 8259A IR levels (IR0-IR4) with their corresponding interrupt type in the 8086 interrupt-vector table. Type 77 in the table is selected by a software "INT" instruction during program execution. Each type is programmed with the necessary code segment and instruction pointer values for vectoring to the appropriate service routine. Since the 8259A is programmed in the automatic EOI Mode, it doesn't require an EOI command to designate the completion of the service routine.

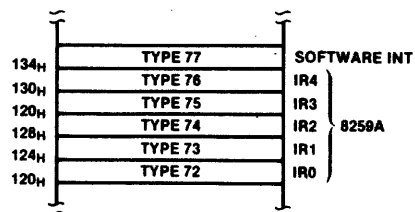


Figure 36. Interrupt "Type" Designation

As mentioned earlier, the interrupt service routines in this application are used merely to demonstrate the timer controlled interrupt scheme, not to implement a particular design. Thus a service routine simply displays the number of its interrupting level on the SDK-86 display panel. The display panel is controlled by the 8279 Keyboard and Display Controller. It is initialized to display "1r" in its two left-most digits during the entire display sequence. When an interrupt from IR1 through IR4 occurs the corresponding routine will display its IR number via the 8279. During each IR1 through IR4 service routine a software "INT77" instruction is executed. This instruction vectors program execution to the service routine designated by type 77, which sets the 8253 counter controlling IR0 so it will cause an interrupt in 250 ms. When the IR0 interrupt occurs its routine will turn off the digit displayed by the IR1 through IR4 routines. Thus each IR level (IR1-IR4) will be displayed for 250 ms followed by a 250 ms off time caused by IR0. Figure 37 shows the entire display sequence of the timer controlled interrupt application.

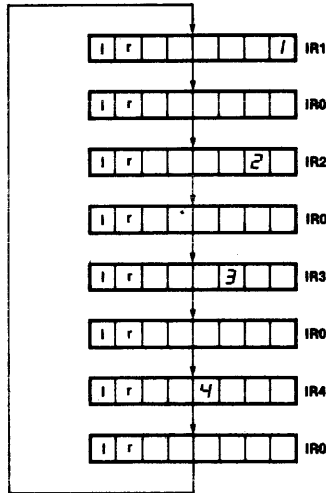


Figure 37. SDK Display Sequence for Timer Controlled Interrupts Program (Each Display Block Shown is 250 msec in Duration)

Now that we've covered the operation, let's move on to the program flow and structure of the timer controlled interrupt program. The program flow is made up of an Initialization section and six interrupt service routines. The initialization program flow is shown in Figure 38. It starts by initializing some of the 8086's registers for program operation; this includes the extra segment, data segment, stack segment, and stack pointer. Next, by using the extra segment as reference, interrupt types 72 through 77 are set to vector interrupts to the appropriate routines. This is done by moving the code segment and instruction pointer values of each service routine into the corresponding type location. The 8253 counters are then programmed with the proper mode and count to provide the interrupt timing mentioned earlier. All counters with the exception of the 8253 #1, counter 1 are fully initialized at this point and will start counting. Counter 1 of 8253 #1 starts counting when its counter is loaded during the "INTR77" service routine, which will be covered shortly. Next, the 8259A is issued ICW1, ICW2, ICW4, and OCW1. The ICWs program the

8259A for the edge triggered mode, automatic EOI mode, and the proper interrupt vectoring (IR0, type 72). OCW1 is used to mask off the unused IR inputs (IR5-IR7). The 8279 is then set to display "1r" on its two left-most digits. After that the 8086 enables interrupts and a "dummy" main program is executed to wait for interrupt requests.

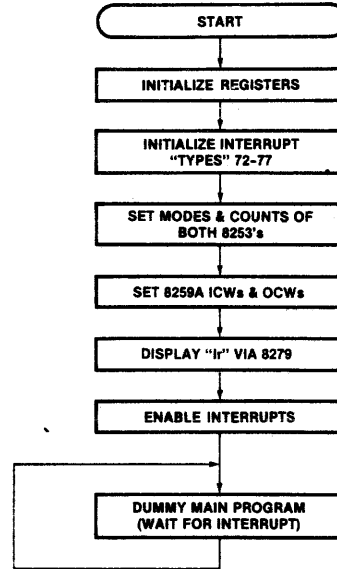


Figure 38. Initialization Program Flow for Timer Controlled Interrupts

There are six different interrupt service routines used in the program. Five of these routines, "INTR72" through "INTR76", are vectored to via the 8259A. Figure 39A-C shows the program flow for all six service routines. Note that "INTR73" through "INTR76" (IR1-IR4) basically use the same flow. These four similar routines display the number of its interrupting IR level on the SDK-86 display panel. The "INTR77" routine is vectored to by software during each of the previously mentioned routines and sets up interrupt timing to cause the "INTR72" (IR0) routine to be executed. The "INTR72" routine turns off the number on the SDK-86 display panel.

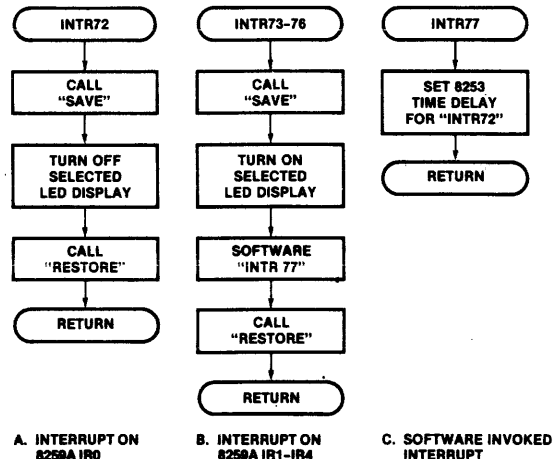


Figure 39. A-C. Interrupts Service Routine Flow for Timer Controlled Interrupts.

To best explain how these service routines work, let's assume an interrupt occurred on IR1 of the 8259A. The associated service routine for IR1 is "INTR73". Entering "INTR73", the first thing done is saving the pre-interrupt program status. This isn't really necessary in this program since a "dummy" main program is being executed; however, it is done as an example to show the operation. Rather than having code for saving the registers in each separate routine, a mutual call routine, "SAVE", is used. This routine will save the register status by pushing it on the stack. The next portion of "INTR73" will display the number of its IR level, "1", in the first digit of the SDK-86 display panel. After that, a software INT instruction is executed to vector program execution to the "INTR77" service routine. The "INTR77" service routine simply sets the 8253 #1 counter 1 to cause an interrupt on IR0 in 250 ms and then returns to "INTR73". Once back in "INTR73", the pre-interrupt status is restored by a call routine, "RESTORE". It does the opposite of "SAVE", returning the register status by popping it off the stack. The "INTR73" routine then returns to the "dummy" main program. The flow for the "INTR74" through "INTR76" routines are the same except for the digit location and the IR level displayed.

After 250 ms have elapsed, counter 1 of 8253 #1 makes an interrupt request on IR0 of the 8259A. This causes the "INTR72" service routine to be executed. Since this routine interrupts the main program, it also uses the "SAVE" routine to save pre-interrupt program status. It then turns off the digit displaying the IR level. In the case of the "INTR73" routine, the "1" is blanked out. The pre-interrupt status is then restored using the "RESTORE" routine and program execution returns to the "dummy" main program.

The complete program for the timer controlled interrupts application is shown in Appendix B. The program was executed in SDK-86 RAM starting at location 0500H (code segment = 0050, instruction pointer = 0).

CONCLUSION

This application note has explained the 8259A in detail and gives three applications illustrating the use of some of the numerous programmable features available. It should be evident from these discussions that the 8259A is an extremely flexible and easily programmable member of the Intel® MCS-80, MCS-85, MCS-86, and MCS-88 families.

This table is provided merely for reference information between the "Operation of the 8259A" and "Programming the 8259A" sections of this application note. It shouldn't be used as a programming reference guide (see "Programming the 8259A").

Operational Description	Command Words	Bits
MCS-80/85™ Mode	ICW1, ICW4*	IC4, μ PM*
Address Interval for MCS-80/85 Mode	ICW1	ADI
Interrupt Vector Address for MCS-80/85 Mode	ICW1, ICW2	A5-A15
MCS-86/88 Mode	ICW1, ICW4	IC4, μ PM
Interrupt Vector Byte for MCS-86/88 Mode	ICW2	T3-T7
Fully Nested Mode	OCW-Default	—
Non-Specific EOI Command	OCW2	EOI
Specific EOI Command	OCW2	SEOI, EOI, LO-L2
Automatic EOI Mode	ICW1, ICW4	IC4, AEOI
Rotate On Non-Specific EOI Command	OCW2	EOI
Rotate In Automatic EOI Mode	OCW2	R, SEOI, EOI
Set Priority Command	OCW2	L0-L2
Rotate on Specific EOI Command	OCW2	R, SEOI, EOI
Interrupt Mask Register	OCW1	M0-M7
Special Mask Mode	OCW3	ESMM-SMM
Level Triggered Mode	ICW1	LTIM
Edge Triggered Mode	ICW1	LTIM
Read Register Command, IRR	OCW3	ERIS, RIS
Read Register Command, ISR	OCW3	ERIS, RIS
Read IMR	OCW1	M0-M7
Poll Command	OCW3	P
Cascade Mode	ICW1, ICW3	SNGL, S0-7, ID0-2
Special Fully Nested Mode	ICW1, ICW4	IC4, SFNM
Buffered Mode	ICW1, ICW4	IC4, BUF, M/S

*Only needed if ICW4 is used for purposes other than μ P mode set.

MCS-86 ASSEMBLER TC159A

ISIS-IT MCS-86 ASSEMBLER V1.0 ASSEMBLY OF MODULE TC159A
 OBJECT MODULE PLACED IN: F1:TC159A.OBJ
 ASSEMBLER INVOKED BY: F1:ASM86 F1:TC159A.SRC

```

LOC OBJ                LINE  SOURCE
-----
                                1  ;***** TIMER CONTROLLED INTERRUPTS *****
                                2  ;
                                3  ;
                                4  ;
                                5  ;
                                6  ;
                                7  EXTRA SEGMENT
                                8  ;
0120                      9      ORG      120H
0120 0401                 10  TP72IP  DW      INTR72      ;TYPE 72 INSTRUCTION POINTER
0122 ????                 11  TP72CS  DW      ?          ;TYPE 72 CODE SEGMENT
0124 1801                 12  TP73IP  DW      INTR73      ;TYPE 73 INSTRUCTION POINTER
0126 ????                 13  TP73CS  DW      ?          ;TYPE 73 CODE SEGMENT
0128 3001                 14  TP74IP  DW      INTR74      ;TYPE 74 INSTRUCTION POINTER
012A ????                 15  TP74CS  DW      ?          ;TYPE 74 CODE SEGMENT
012C 4801                 16  TP75IP  DW      INTR75      ;TYPE 75 INSTRUCTION POINTER
012E ????                 17  TP75CS  DW      ?          ;TYPE 75 CODE SEGMENT
0130 6001                 18  TP76IP  DW      INTR76      ;TYPE 76 INSTRUCTION POINTER
0132 ????                 19  TP76CS  DW      ?          ;TYPE 76 CODE SEGMENT
0134 7801                 20  TP77IP  DW      INTR77      ;TYPE 77 INSTRUCTION POINTER
0136 ????                 21  TP77CS  DW      ?          ;TYPE 77 CODE SEGMENT
                                22  ;
-----                   23  EXTRA ENDS
                                24  ;
                                25  ;
                                26  ;
                                27  DATA  SEGMENT
                                28  ;
0000 ????                 29  STACK1 DW      ?          ;VARIABLE TO SAVE CALL ADDRESS
0002 ????                 30  AXTEMP DW      ?          ;VARIABLE TO SAVE AX REGISTER
0004 ??                   31  DIGIT  DB      ?          ;VARIABLE TO SAVE SELECTED DIGIT
                                32  ;
-----                   33  DATA  ENDS
                                34  ;
                                35  ;
                                36  ;
                                37  CODE   SEGMENT
                                38  ;
                                39  ASSUME ES:EXTRA,DS:DATA,CS:CODE
                                40  ;
                                41  ;
                                42  ;
                                43  ;
                                44  ;
                                45  ;
                                46  ;
                                47  ;
                                48  ;
                                49  ;
0000 B80000              43  START: MOV     AX,0H          ;EXTRA SEGMENT AT 0H
0003 8ED8                44          MOV     ES,AX
0005 B87000              45          MOV     AX,70H          ;DATA SEGMENT AT 70H
0008 8ED8                46          MOV     DS,AX
000A B87800              47          MOV     AX,78H          ;STACK SEGMENT AT 78H
000D 8ED8                48          MOV     SS,AX
000F BC8000              49          MOV     SP,80H          ;STACK POINTER AT 80H (STACK=800H)

```

AP-59

MCS-86 ASSEMBLER TC159A

LOC	OBJ	LINE	SOURCE	
		50		
		51		LOAD INTERRUPT VECTOR TABLE
		52		
0012	B80401	53	TYPES: MOV AX, OFFSET (INTR72)	:LOAD TYPE 72
0015	26A32001	54	MOV IP72IP, AX	
0019	268C0E2201	55	MOV IP72CS, CS	
001E	B81801	56	MOV AX, OFFSET (INTR73)	:LOAD TYPE 73
0021	26A32401	57	MOV IP73IP, AX	
0025	268C0E2601	58	MOV IP73CS, CS	
002A	B83001	59	MOV AX, OFFSET (INTR74)	:LOAD TYPE 74
002D	26A32801	60	MOV IP74IP, AX	
0031	268C0E2A01	61	MOV IP74CS, CS	
0036	B84001	62	MOV AX, OFFSET (INTR75)	:LOAD TYPE 75
0039	26A32C01	63	MOV IP75IP, AX	
003D	268C0E2E01	64	MOV IP75CS, CS	
0042	B86001	65	MOV AX, OFFSET (INTR76)	:LOAD TYPE 76
0045	26A33001	66	MOV IP76IP, AX	
0049	268C0E3201	67	MOV IP76CS, CS	
004E	B87801	68	MOV AX, OFFSET (INTR77)	:LOAD TYPE 77
0051	26A33401	69	MOV IP77IP, AX	
0055	268C0E3601	70	MOV IP77CS, CS	
		71		
		72		8253 INITIALIZATION
		73		
005A	BA0EFF	74	SETS31: MOV DX, 0FF0EH	:8253 #1 CONTROL WORD
005D	B036	75	MOV AL, 36H	:COUNTER 0, MODE 3, BINARY
005F	EE	76	OUT DX, AL	
0060	B071	77	MOV AL, 71H	:COUNTER 1, MODE 0, BCD
0062	EE	78	OUT DX, AL	
0063	B065	79	MOV AL, 065H	:COUNTER 2, MODE 2, BCD
0065	EE	80	OUT DX, AL	
0066	BA08FF	81	MOV DX, 0FF08H	:LOAD COUNTER 0 (10MS)
0069	B09B	82	MOV AL, 09BH	:LSB
006B	E1	83	OUT DX, AL	
006C	B061	84	MOV AL, 61H	:MSB
006E	EE	85	OUT DX, AL	
006F	BA0CFE	86	MOV DX, 0FF0CH	:LOAD COUNTER 2 (2SEC)
0072	B000	87	MOV AL, 00H	:LSB
0074	EE	88	OUT DX, AL	
0075	B002	89	MOV AL, 02H	:MSB
0077	EE	90	OUT DX, AL	
0078	BA16FF	91	SETS32: MOV DX, 0FF16H	:8253 #2 CONTROL WORD
007B	B03B	92	MOV AL, 3BH	:COUNTER 0, MODE 5, BCD
007D	EE	93	OUT DX, AL	
007E	B07B	94	MOV AL, 7BH	:COUNTER 1, MODE 5, BCD
0080	EE	95	OUT DX, AL	
0081	B0BB	96	MOV AL, 0BBH	:COUNTER 2, MODE 5, BCD
0083	EE	97	OUT DX, AL	
0084	BA10FF	98	MOV DX, 0FF10H	:LOAD COUNTER 0 (.5SEC)
0087	B050	99	MOV AL, 50H	:LSB
0089	EE	100	OUT DX, AL	
008A	B000	101	MOV AL, 00H	:MSB
008C	EE	102	OUT DX, AL	
008D	BA12FF	103	MOV DX, 0FF12H	:LOAD COUNTER 1 (1SEC)
0090	B000	104	MOV AL, 00H	:LSB

MCS-86 ASSEMBLER TC159A

LOC	OBJ	LINE	SOURCE		
0092	EE	105	OUT	DX, AL	
0093	B001	106	MOV	AL, 01H	; MSB
0095	EE	107	OUT	DX, AL	
0096	BA14FF	108	MOV	DX, 0FF14H	; LOAD COUNTER 2 (1.5SEC)
0099	B050	109	MOV	AL, 50H	; LSB
009B	EE	110	OUT	DX, AL	
009C	B001	111	MOV	AL, 01H	; MSB
009E	EE	112	OUT	DX, AL	
		113			
		114		8259A INITIALIZATION	
		115			
009F	BA00FF	116	SETS9A: MOV	DX, 0FF00H	; 8259A A0=0
00A2	B013	117	MOV	AL, 13H	; ICM1-LTIM=0, S=1, IC4=1
00A4	EE	118	OUT	DX, AL	
00A5	BA02FF	119	MOV	DX, 0FF02H	; 8259A A0=1
00A8	B048	120	MOV	AL, 48H	; ICM2-INTERRUPT TYPE 72 (120H)
00AA	EE	121	OUT	DX, AL	
00AB	B003	122	MOV	AL, 03H	; ICM4-SFNM=0, BUF=0, AED1=1, MPM=1
00AD	EE	123	OUT	DX, AL	
00AE	B0E0	124	MOV	AL, 0E0H	; OCM1-MASK IRS, 6, 7 (NOT USED)
00B0	EE	125	OUT	DX, AL	
		126			
		127		8279 INITIALIZATION	
		128			
00B1	BAE8FF	129	SET79: MOV	DX, 0FE8H	; 8279 COMMAND WORDS AND STATUS
00B4	B0D0	130	MOV	AL, 0D0H	; CLEAR DISPLAY
00B6	EE	131	OUT	DX, AL	
00B7	EC	132	WAIT79: IN	AL, DX	; READ STATUS
00B8	D0C0	133	ROL	AL, 1	; "DU" BIT TO CARRY
00BA	72FB	134	JB	WAIT79	; JUMP IF DISPLAY IS UNAVAILABLE
00BC	B087	135	MOV	AL, 87H	; DIGIT 8
00BE	EE	136	OUT	DX, AL	
00BF	BAE8FF	137	MOV	DX, 0FE8H	; 8279 DATA WORD
00C2	B006	138	MOV	AL, 06H	; CHARACTER "1"
00C4	EE	139	OUT	DX, AL	
00C5	BAE8FF	140	MOV	DX, 0FE8H	; 8279 COMMAND WORD
00C8	B086	141	MOV	AL, 86H	; DIGIT 7
00CA	EE	142	OUT	DX, AL	
00CB	BAE8FF	143	MOV	DX, 0FE8H	; 8279 DATA WORD
00CE	B050	144	MOV	AL, 50H	; CHARACTER "R"
00D0	EE	145	OUT	DX, AL	
00D1	FB	146	STI		; ENABLE INTERRUPTS
		147			
		148			
		149		DUMMY PROGRAM	
		150			
00D2	EBFE	151	DUMMY: JMP	DUMMY	; WAIT FOR INTERRUPT
		152			
		153			
00D4	A30200	154	SAVE: MOV	AXTEMP, AX	; SAVE AX
00D7	58	155	POP	AX	; POP CALL RETURN ADDRESS
00D8	A30000	156	MOV	STACK1, AX	; SAVE CALL RETURN ADDRESS
00DB	A10200	157	MOV	AX, AXTEMP	; RESTORE AX
00DE	58	158	PUSH	AX	; SAVE PROCESSOR STATUS
00DF	53	159	PUSH	BX	

AP-59

MCS-86 ASSEMBLER TC159A

LOC	OBJ	LINE	SOURCE		
00E0	51	160	PUSH	CX	
00E1	52	161	PUSH	DX	
00E2	55	162	PUSH	BP	
00E3	56	163	PUSH	SI	
00E4	57	164	PUSH	DI	
00E5	1E	165	PUSH	DS	
00E6	06	166	PUSH	ES	
00E7	R10000	167	MOV	AX, STACK1	; RESTORE CALL RETURN ADDRESS
00EA	50	168	PUSH	AX	; PUSH CALL RETURN ADDRESS
00EB	C3	169	RET		
		170			
00EC	50	171	SECTOR: POP	AX	; POP CALL RETURN ADDRESS
00ED	R30000	172	MOV	STACK1, AX	; SAVE CALL RETURN ADDRESS
00F0	07	173	POP	ES	; RESTORE PROCESSOR STATUS
00F1	1F	174	POP	DS	
00F2	5F	175	POP	DI	
00F3	5E	176	POP	SI	
00F4	5D	177	POP	BP	
00F5	5A	178	POP	DX	
00F6	59	179	POP	CX	
00F7	58	180	POP	BX	
00F8	58	181	POP	AX	
00F9	R30200	182	MOV	AXTEMP, AX	; SAVE AX
00FC	R10000	183	MOV	AX, STACK1	; RESTORE CALL RETURN ADDRESS
00FF	50	184	PUSH	AX	; PUSH CALL RETURN ADDRESS
0100	R10200	185	MOV	AX, AXTEMP	; RESTORE AX
0103	C3	186	RET		
		187			
		188			
		189			INTERRUPT 72, CLEAR DISPLAY, IRQ 8259A
		190			
0104	E8CDF	191	INTR72: CALL	SAVE	; ROUTINE TO SAVE PROCESSOR STATUS
0107	BAE8FF	192	MOV	DX, 0FFEAH	; 8279 COMMAND WORD
010A	A00400	193	MOV	AL, DIGIT	; SELECTED LED DIGIT
010D	EE	194	OUT	DX, AL	
010E	BAE8FF	195	MOV	DX, 0FFEAH	; 8279 DATA
0111	B000	196	MOV	AL, 00H	; BLANK OUT DIGIT
0113	EE	197	OUT	DX, AL	
0114	E8D5FF	198	CALL	RESTOR	; ROUTINE TO RESTORE PROCESSOR STATUS
0117	CF	199	IRET		; RETURN FROM INTERRUPT
		200			
		201			
		202			INTERRUPT 73, IRQ 8259A
		203			
0118	E8B9FF	204	INTR73: CALL	SAVE	; ROUTINE TO SAVE PROCESSOR STATUS
011B	BAE8FF	205	MOV	DX, 0FFEAH	; 8279 COMMAND WORD
011E	B080	206	MOV	AL, 80H	; LED DISPLAY DIGIT 1
0120	A20400	207	MOV	DIGIT, AL	
0123	EE	208	OUT	DX, AL	
0124	BAE8FF	209	MOV	DX, 0FFEAH	; 8279 DATA
0127	B006	210	MOV	AL, 06H	; CHARACTER "1"
0129	EE	211	OUT	DX, AL	
012A	CD40	212	INT	77	; TIMER DELAY FOR LED ON TIME
012C	E8B0FF	213	CALL	RESTOR	; ROUTINE TO RESTORE PROCESSOR STATUS
012F	CF	214	IRET		; RETURN FROM INTERRUPT

AP-59

MCS-86 ASSEMBLER TC159A

LOC	OBJ	LINE	SOURCE	
		215	:	
		216	:	
		217	:	INTERRUPT 74, IR2 8259A
		218	:	
0130	E8A1FF	219	INTR74: CALL	SAVE ; ROUTINE TO SAVE PROCESSOR STATUS
0133	B8E8FF	220	MOV	DX, 0FFEAH ; 8279 COMMAND WORD
0136	B081	221	MOV	AL, 81H ; LED DISPLAY DIGIT 2
0138	A20400	222	MOV	DIGIT, AL
013B	EE	223	OUT	DX, AL
013C	B8E8FF	224	MOV	DX, 0FFEAH ; 8279 DATA
013F	B05B	225	MOV	AL, 5BH ; CHARACTER "2"
0141	EE	226	OUT	DX, AL
0142	CD4D	227	INT	77 ; TIMER DELAY FOR LED ON TIME
0144	E8A5FF	228	CALL	RESTOR ; ROUTINE TO RESTORE PROCESSOR STATUS
0147	CF	229	IRET	; RETURN FROM INTERRUPT
		230	:	
		231	:	
		232	:	INTERRUPT 75, IR3 8259A
		233	:	
0148	E889FF	234	INTR75: CALL	SAVE ; ROUTINE TO SAVE PROCESSOR STATUS
014B	B8E8FF	235	MOV	DX, 0FFEAH ; 8279 COMMAND WORD
014E	B082	236	MOV	AL, 82H ; LED DISPLAY DIGIT 3
0150	A20400	237	MOV	DIGIT, AL
0153	EE	238	OUT	DX, AL
0154	B8E8FF	239	MOV	DX, 0FFEAH ; 8279 DATA
0157	B04F	240	MOV	AL, 4FH ; CHARACTER "3"
0159	EE	241	OUT	DX, AL
015A	CD4D	242	INT	77 ; TIMER DELAY FOR LED ON TIME
015C	E880FF	243	CALL	RESTOR ; ROUTINE TO RESTORE PROCESSOR STATUS
015F	CF	244	IRET	; RETURN FROM INTERRUPT
		245	:	
		246	:	
		247	:	INTERRUPT 76, IR4 8259A
		248	:	
0160	E871FF	249	INTR76: CALL	SAVE ; ROUTINE TO SAVE PROCESSOR STATUS
0163	B8E8FF	250	MOV	DX, 0FFEAH ; 8279 COMMAND WORD
0166	B083	251	MOV	AL, 83H ; LED DISPLAY DIGIT 4
0168	A20400	252	MOV	DIGIT, AL
016B	EE	253	OUT	DX, AL
016C	B8E8FF	254	MOV	DX, 0FFEAH ; 8279 DATA
016F	B066	255	MOV	AL, 66H ; CHARACTER "4"
0171	EE	256	OUT	DX, AL
0172	CD4D	257	INT	77 ; TIMER DELAY FOR LED ON TIME
0174	E875FF	258	CALL	RESTOR ; ROUTINE TO RESTORE PROCESSOR STATUS
0177	CF	259	IRET	; RETURN FROM INTERRUPT
		260	:	
		261	:	
		262	:	INTERRUPT 77, TIMER DELAY, SOFTWARE CONTROLLED
		263	:	
0178	B808FF	264	INTR77: MOV	DX, 0FF0AH ; LOAD COUNTER 1 8253 #1 (250 MSEC)
017B	B025	265	MOV	AL, 25H ; LSB
017D	EE	266	OUT	DX, AL
017E	B000	267	MOV	AL, 00H ; MSB
0180	EE	268	OUT	DX, AL
0181	CF	269	IRET	; RETURN FROM INTERRUPT

MCS-86 ASSEMBLER TC159A

LOC	OBJ	LINE	SOURCE
		270	;
		271	;
----		272	CODE ENDS;
		273	;
		274	;
0000		275	END START

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
??SEG	SEGMENT		SIZE=0000H PARA PUBLIC
AXTEMP	V WORD	0002H	DATA
CODE	SEGMENT		SIZE=0182H PARA
DATA	SEGMENT		SIZE=0005H PARA
DIGIT	V BYTE	0004H	DATA
DUMMY	L NEAR	0002H	CODE
EXTRA	SEGMENT		SIZE=0138H PARA
INTR72	L NEAR	0104H	CODE
INTR73	L NEAR	0118H	CODE
INTR74	L NEAR	0130H	CODE
INTR75	L NEAR	0148H	CODE
INTR76	L NEAR	0160H	CODE
INTR77	L NEAR	0176H	CODE
RESTOR	L NEAR	00E6H	CODE
SAVE	L NEAR	0004H	CODE
SET531	L NEAR	005AH	CODE
SET532	L NEAR	0076H	CODE
SET59A	L NEAR	009FH	CODE
SET79	L NEAR	00B1H	CODE
STACK1	V WORD	0000H	DATA
START	L NEAR	0000H	CODE
TP7205	V WORD	0122H	EXTRA
TP721P	V WORD	0120H	EXTRA
TP7305	V WORD	0126H	EXTRA
TP731P	V WORD	0124H	EXTRA
TP7405	V WORD	012AH	EXTRA
TP741P	V WORD	0128H	EXTRA
TP7505	V WORD	012EH	EXTRA
TP751P	V WORD	012CH	EXTRA
TP7605	V WORD	0132H	EXTRA
TP761P	V WORD	0130H	EXTRA
TP7705	V WORD	0136H	EXTRA
TP771P	V WORD	0134H	EXTRA
TYPES	L NEAR	0012H	CODE
WAIT79	L NEAR	00B7H	CODE

ASSEMBLY COMPLETE, NO ERRORS FOUND

Votrax[®]

A Division of Federal Screw Works
500 Stephenson Highway
Troy, Michigan 48084

SC-01 SPEECH SYNTHESIZER

DATA SHEET

Votrax[®] CMOS Phoneme Speech Synthesizer

GENERAL DESCRIPTION

The SC-01 Speech Synthesizer is a completely self-contained solid state device. This single chip phonetically synthesizes continuous speech, of unlimited vocabulary, from low data rate inputs. Figure 1.

Speech is synthesized by combining phonemes (the building blocks of speech) in the appropriate sequence. The SC-01 Speech Synthesizer contains 64 different phonemes which are accessed by a 6-bit code. It is the proper sequential combination of these phoneme codes that creates continuous speech.

The SC-01 Speech Synthesizer is cost-effective, consumes minimal power and enables in-house product development without vendor dependency. Signals from the SC-01 are applied to an audio output device to amplify and distribute the synthesized speech. See Figure 2.

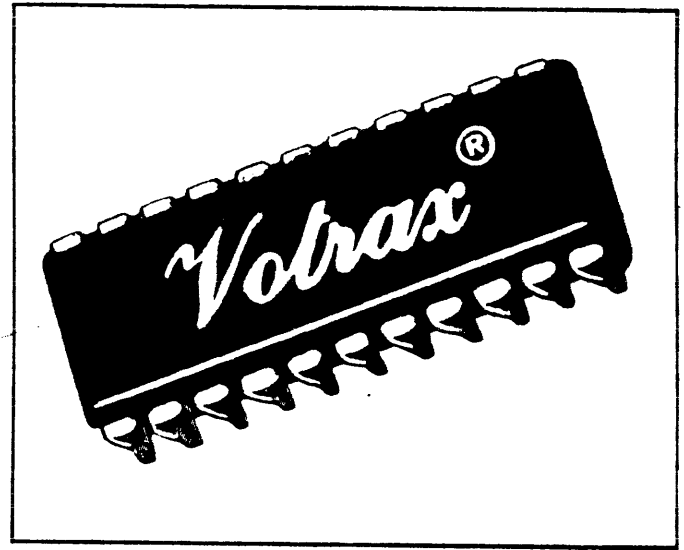


Figure 1. Votrax[®] SC-01 Speech Synthesizer

FEATURES

- Single CMOS chip
- 70 bits per second
- 22 pin package
- 9 ma. current drain
- Wide voltage supply range
- Latched 5v. compatible inputs
- Digital pitch level inputs
- Automatic inflection
- On-chip master clock circuit
- Optional external master clock
- Variety of voice effects
- Sound effects
- Customer product security

The design of the equipment specified herein is proprietary. Rights for the reproduction and distribution of the data contained herein are granted except for the manufacture and reproduction of the subject equipment.

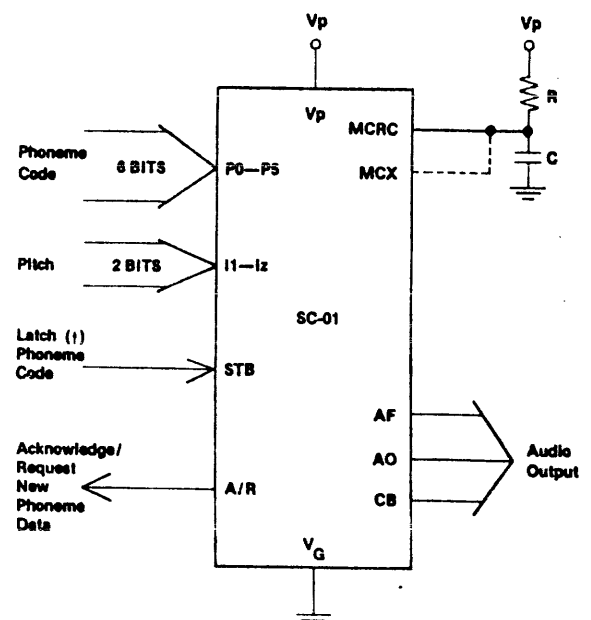


Figure 2. SC-01 Flow Diagram

PHYSICAL DESCRIPTION

The SC-01 Speech Synthesizer comprises a 22 pin monolithic (single level or substrate) integrated circuit of CMOS (Complementary Metal On Silicon) design. See Figure 3. The P and N areas of the substrate, made by ion implantation, create a push-pull (Class B) transistor amplification and digital signal processing network. High impedance, rapid signal processing, and minimal current drain result from this technology.

ELECTRICAL DESCRIPTION

The SC-01 Speech Synthesizer is a program-compatible with existing Votrax[®] phoneme synthesizers. It requires 70 bits of data per second for continuous speech production. The 6-bit phoneme codes are 5 volt compatible and are latched for data bus line applications. A phoneme-construction algorithm and filters, within the chip, create the synthesized audio output.

PHONEME DESCRIPTION

Table 1 lists the 64 phonemes produced by the SC-01. Each phoneme code is accompanied by its symbol, average duration time, and an example. The underlined segments of the example word demonstrate the phoneme use, i.e., sound to be pronounced.

Table 2 subdivides the 64 phoneme symbols into seven categories. Each category represents a different production feature. The first six categories are characterized by voiced, fricative (expired voice), and nasal sounds. The seventh category is characterized by phonemes with no sound output.

PHONEME PROGRAMMING

Manual Operations: Votrax[®] maintains a library of phonetically programmed words. Reference to this library and programming manuals will aid in word synthesis.

Automatic Operations: Votrax[®] can supply a micro-computer system for automatic conversion of English text into phoneme sequences. This system is particularly useful for in-house vocabulary development and product security. Contact Votrax[®] for further information.

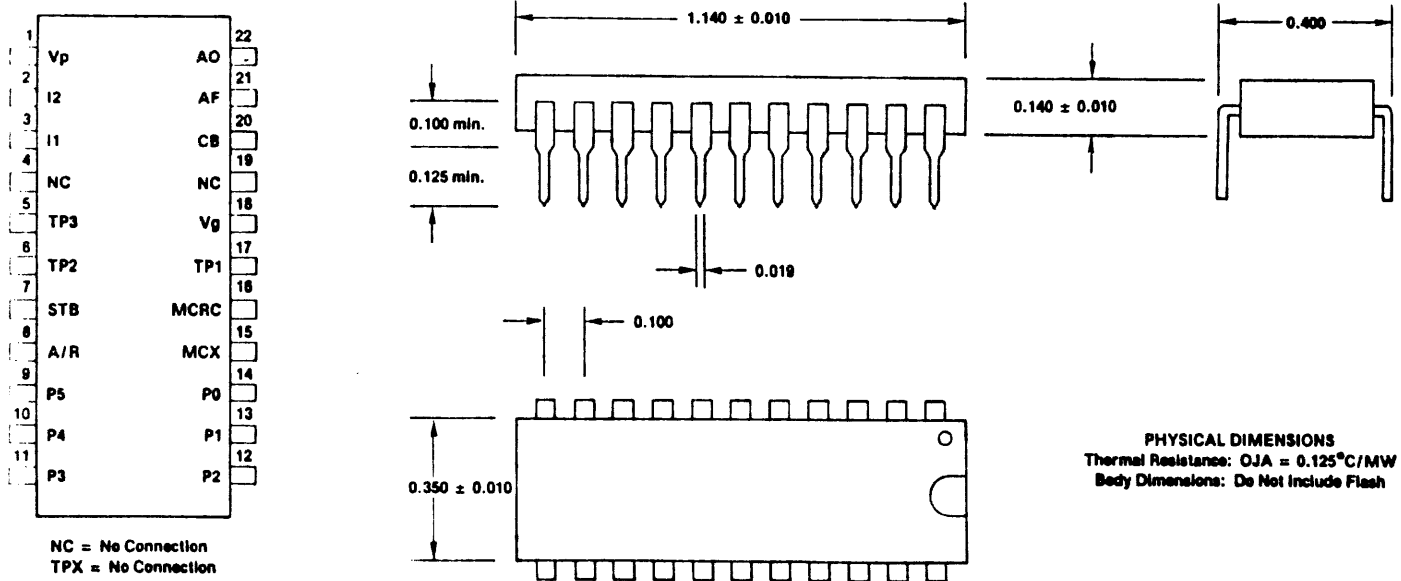


Figure 3. SC-01 Footprint and Outline Dimensions

Votrax[®] reserves the right to alter its product line at any time, or change specifications or design without notice and without obligation.

Table 1. Phoneme Chart

Phoneme Code	Phoneme Symbol	Duration (ms)	Example Word
00	EH3	59	jacket
01	EH2	71	enlist
02	EH1	121	heavy
03	PA0	47	no sound
04	DT	47	butter
05	A2	71	made
06	A1	103	made
07	ZH	90	azure
08	AH2	71	honest
09	I3	55	inhibit
0A	I2	80	inhibit
0B	I1	121	inhibit
0C	M	103	mat
0D	N	80	sun
0E	B	71	bag
0F	V	71	van
10	CH*	71	chip
11	SH	121	shop
12	Z	71	zoo
13	AW1	146	lawful
14	NG	121	thing
15	AH1	146	father
16	OO1	103	looking
17	OO	185	book
18	L	103	land
19	K	80	trick
1A	J*	47	judge
1B	H	71	hello
1C	G	71	get
1D	F	103	fast
1E	D	55	paid
1F	S	90	pass

Phoneme Code	Phoneme Symbol	Duration (ms)	Example Word
20	A	185	day
21	AY	65	day
22	Y1	80	yard
23	UH3	47	mission
24	AH	250	mop
25	P	103	past
26	O	185	cold
27	I	185	pin
28	U	185	move
29	Y	103	any
2A	T	71	tap
2B	R	90	red
2C	E	185	meet
2D	W	80	win
2E	AE	185	dad
2F	AE1	103	after
30	AW2	90	salty
31	UH2	71	about
32	UH1	103	uncle
33	UH	185	cup
34	O2	80	for
35	O1	121	aboard
36	IU	59	you
37	U1	90	you
38	THV	80	the
39	TH	71	thin
3A	ER	146	bird
3B	EH	185	get
3C	E1	121	be
3D	AW	250	call
3E	PA1	185	no sound
3F	STOP	47	no sound

* T must precede CH to produce J sound.
 D must precede J to produce CH sound.

Table 2. Phoneme Categories According to Production Features

Voiced		'Voiced' Fricat.	'Voiced' Stop	Fricat. Stop	Fricative	Nasal	No Sound			
E	EH	AE	UH	OO1	Z	B	T	S	M	PA0
E1	EH1	AE1	UH1	R	ZH	D	DT	SH	N	PA1
Y	EH2	AH	UH2	ER	J	G	K	CH	NG	STOP
Y1	EH3	AH1	UH3	L	V		P	TH		
I	A	AH2	O	IU	THV			F		
I1	A1	AW	O1	U				H		
I2	A2	AW1	O2	U1						
I3	AY	AW2	OO	W						

Votrax® reserves the right to alter its product line at any time, or change specifications or design without notice and without obligation.

SIGNAL DESCRIPTION (See Figures 4 and 5)

Phoneme 6-Bit Selection Code (P0-P5): Data input is to six pins. Latching is controlled by the strobe (STB) signal.

Strobe (STB): Latching occurs on rising edge of strobe signal.

Inflection Level Setting (I1, I2): Instantaneously set pitch of voiced phonemes.

Acknowledge/Request (A/R): Acknowledges receipt of phoneme data (signal goes from high to low one master clock cycle following active edge of STB signal). Also indicates timing out of old phoneme concurrent with request for new phoneme data (signal goes from low to high).

NOTE

If external phoneme timing is desired, phoneme requests can be ignored. However, best speech is realized with internal timing.

Master Clock Resistor-Capacitor (MCRC): This input determines the internal master clock frequency. Select R-C values for 720 KHz to achieve standard phoneme timing. Connect this input to MCX when using internal clock; ground when using external clock.

NOTE

Varying clock frequency varies voice and sound effects. As clock frequency decreases, audio frequency decreases and phoneme timing lengthens. Figures 6 and 7 illustrate manual and DAC (Digital to Analog Converter) voice variation schematics, respectively.

Master Clock External (MCX): Allows control by an external clock signal.

NOTE

Ground MCRC during MCX operation.

Audio Output (AO): Supplies analog signal to audio output device.

Audio Feedback (AF): Used with Class A or Class B transistor audio amplifiers for added stability.

Class B (CB): Current source for Class B transistor audio amplifier.

Table 3. Timing Specifications

CHARACTERISTIC	SYMBOL	MIN	TYP	MAX	UNIT
Input Setup Time (P _i to STB)	T _S			450	NS
Input Hold Time (P _i to STB)	T _H			0	NS
Rise Time of STB Edge (.8v to 4v)	T _{RS}			100	NS
A/R Width (A/R Connected to STB) †	T _{ARW}	1	1.3	2	μs
STB Width	T _{SW}	200			NS
STB Low*	T _{SL}	*			
Propagation Delay (STB to A/R after 2μs)	T _{DAR}			500	NS
A/R Rise Time (Capacitive load = 30pf)	T _{RAR}			100	NS
A/R Fall Time (Capacitive load = 30pf)	T _{FAR}			100	NS
Time from A/R Request to STB Service)	T _{ARS}	0	500		μs
Time of Phoneme Duration †	T _{PH}	47	107	250	MS

† Dependent on Master Clock frequency: 720KHz = 1.3μs

* Strobe must remain low (64 x master clock period) before rising edge.

Votrax® reserves the right to alter its product line at any time, or change specifications or design without notice and without obligation.

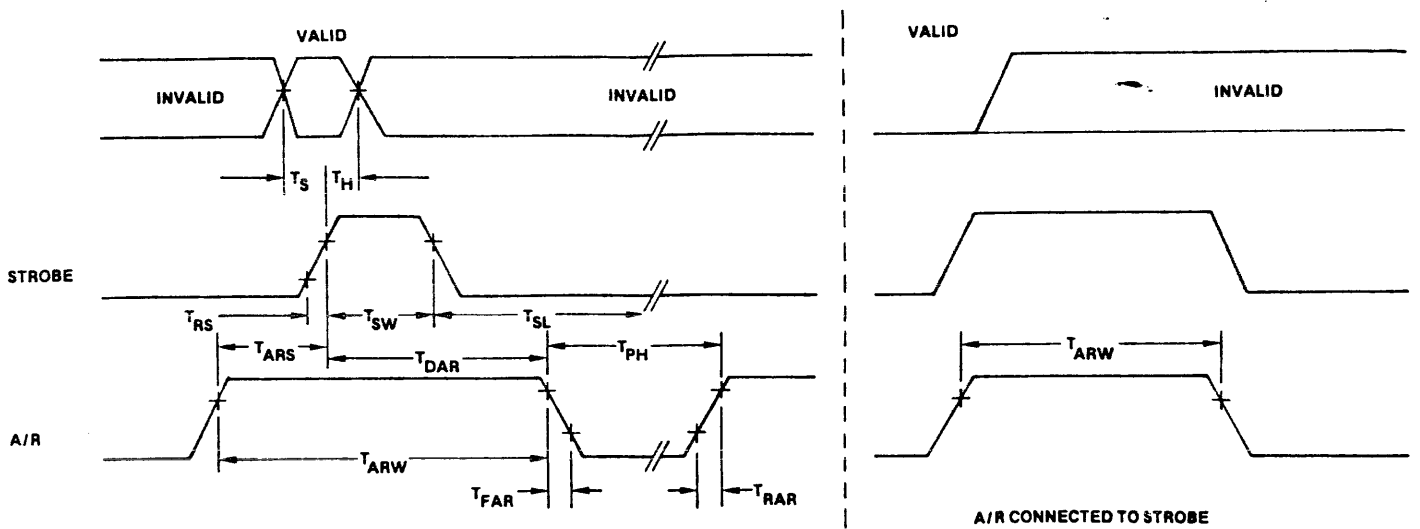


Figure 4. Timing Diagram

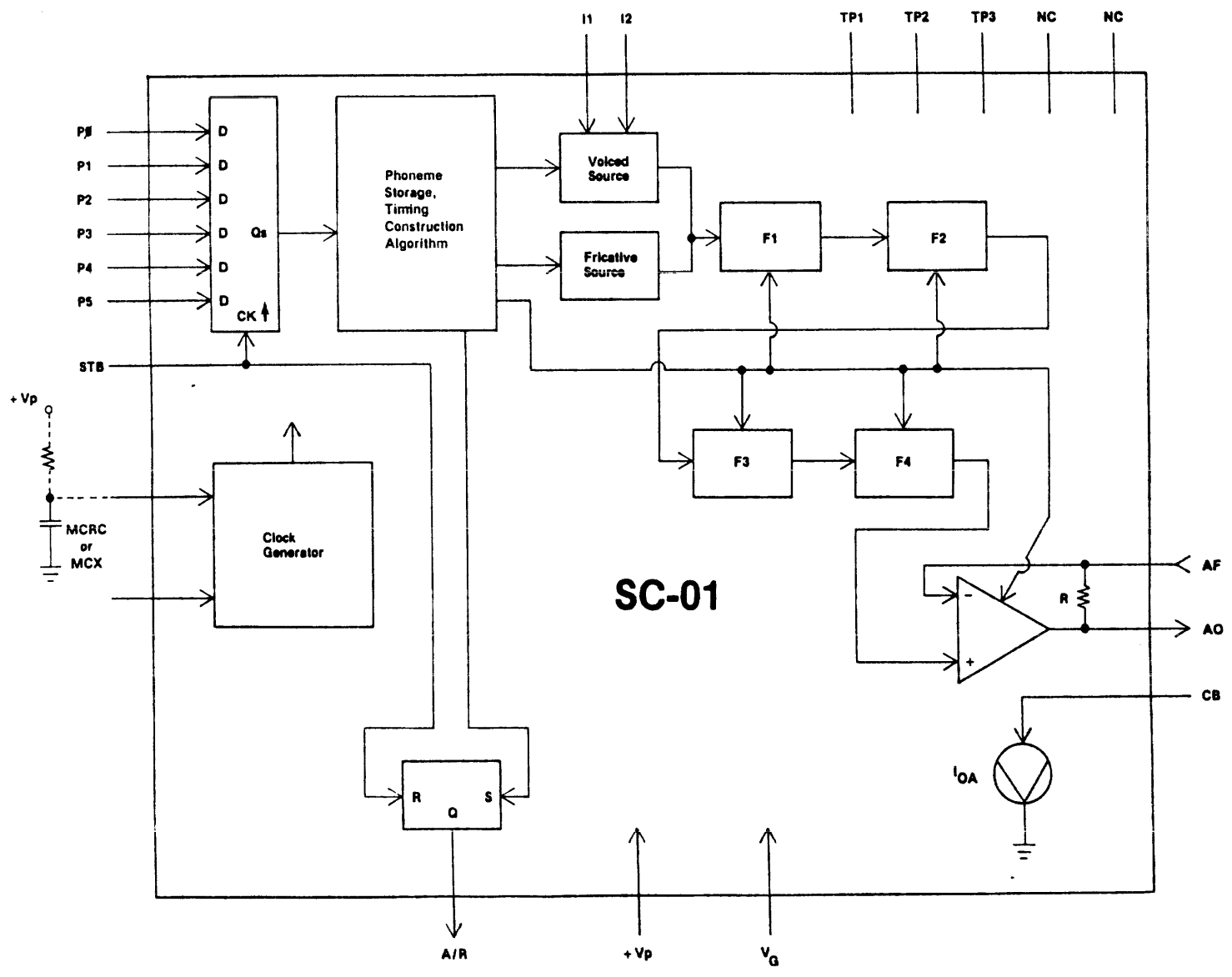


Figure 5. SC-01 Block Diagram

Votrax® reserves the right to alter its product line at any time, or change specifications or design without notice and without obligation.

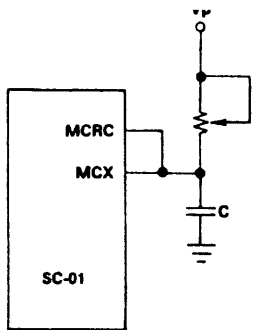


Figure 6. Variable Voice by Potentiometer Control

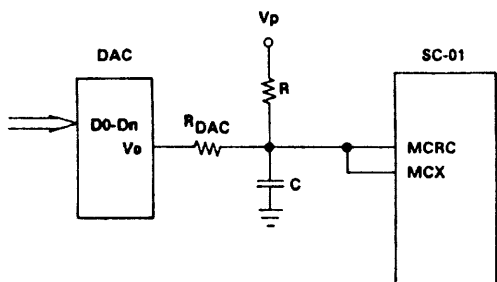


Figure 7. Variable Voice by DAC Current Injection

TYPICAL APPLICATIONS

General: The SC-01 Speech Synthesizer is easily designed into systems ranging in complexity from ROM/counters to microprocessor controllers.

Single Message System: See Figure 8. When the counter is released (START is TRUE), the message is clocked out of the ROM by the A/R signal. The system must be stopped when DONE is TRUE.

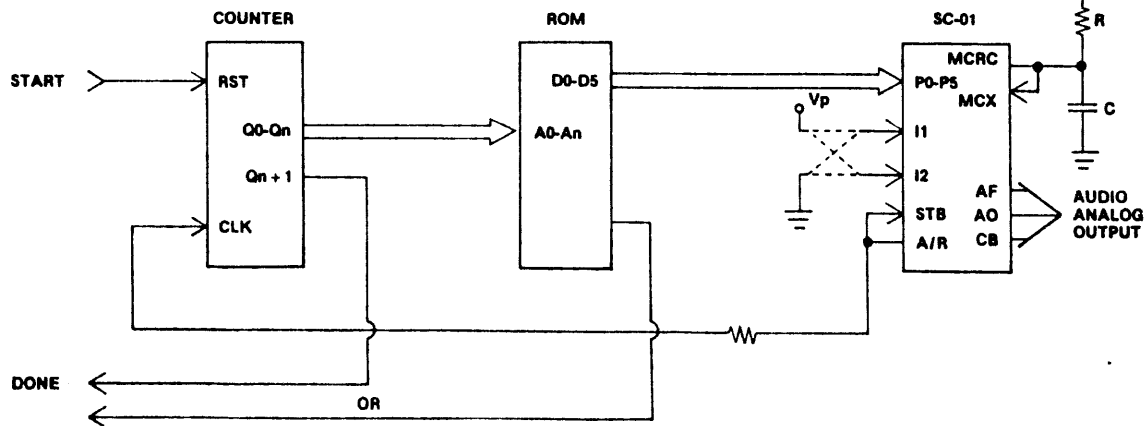


Figure 8. Single Message System

NOTE

Data at address 0 must be a pause phoneme code.

Multiple Message, Fixed Block Size: See Figure 9. Message address block is loaded into the counter. The message is then clocked out of the ROM by the A/R signal.

NOTE

Message Block = 2^n maximum.

Multiple Message, Variable Block Size: See Figure 10. The microprocessor loads phonemes into a data bus. The A/R signal generates an interrupt request for each new phoneme.

CONNECTING THE AUDIO OUTPUT DEVICE

Audio Output: The AO signal has a maximum voltage swing of 1.5 volts below V_p or above V_G , depending upon the phoneme selected. Furthermore, the AO signal is D.C. biased by a factor of $V_p/2$.

NOTE

If additional audio amplifier stability is not needed, connect the AO pin to the AF pin.

Class A Amplifier: See Figure 11. For a single transistor amplifier, the selection of R, C, or R_s values depends upon the value of V_p and the desired audio level.

NOTE

The CB pin is not used for Class A amplifiers.

Votrax® reserves the right to alter its product line at any time, or change specifications or design without notice and without obligation.

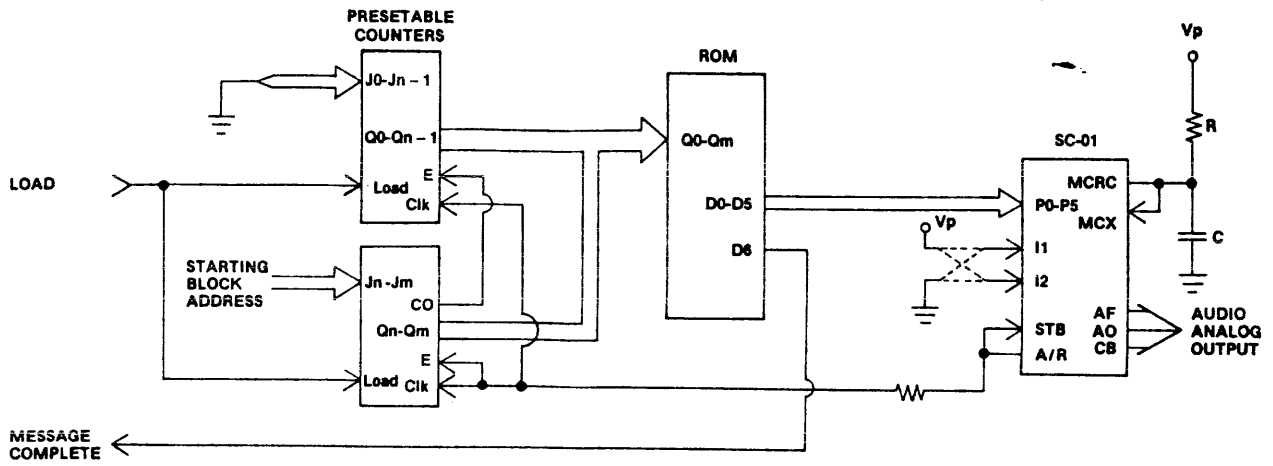


Figure 9. Multiple Message, Fixed Block Size

Class B Amplifier: See Figure 12. A current source (CB) is required for this push-pull amplifier.

NOTE

Minimum power is consumed when speech is inactive. When $V_p = +12.0$ volts and $R_s = 40$ ohms, the bias current drain is approximately 3.5 milliamps.

Controlling Audio Output Power: See Figure 13. A resistor or potentiometer from the speaker to ground can be used to control the audio output power.

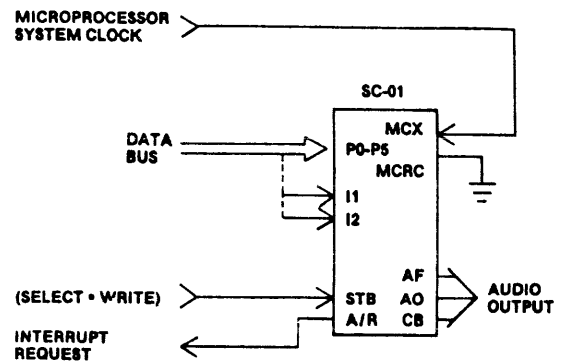


Figure 10. Multiple Message, Variable Block Size

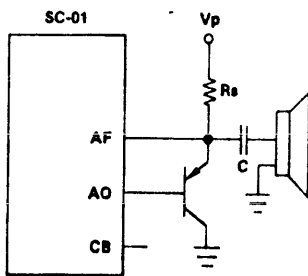


Figure 11. Class A Amplifier

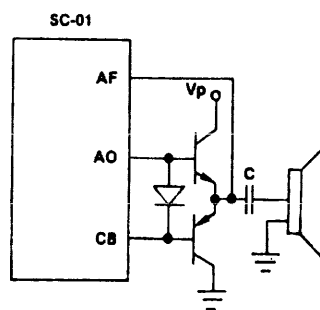


Figure 12. Class B Amplifier

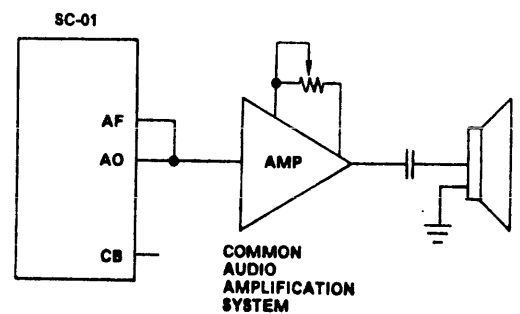


Figure 13. Controlling Audio Output Power

Table 4. Analog Output Specifications

CHARACTERISTIC	MIN	MAX	UNIT
Output Voltage (AW Phoneme)	.39·V _p	.58·V _p	V _{p-p}
Output Bias Current ** (.4v CB V _p)	3.5	7.3	MA

ELECTRICAL CHARACTERISTICS: T_O = 0 to 70°C, V_p = 7 to 14 V_{DC}

CHARACTERISTIC	MIN	TYP	MAX	UNIT
Digital Input Impedance	1 meg.			OHM
Input Capacitance (P ₁ , STB)			3	pf
Input Capacitance (I1, I2, MCX)			8	pf
Digital Input Logic "0" (except I1, I2, MCX)	V _G +0.8		V _G -0.5	V _{DC}
Digital Input Logic "0" (I1, I2, MCX)	V _G +1.0			V _{DC}
Digital Input Logic "1" (except I1, I2, MCX)	V _p +0.5		V _G +4.0	V _{DC}
Digital Input Logic "1" (I1, I2)			V _p -1.0	V _{DC}
Digital Input Logic "1" (MCX)			4.6	V _{DC}
Digital Output Logic "0" (I sink = 0.8MA)			V _G +0.5	V _{DC}
Digital Output Logic "1" (I source = 0.5MA)	V _p -0.5			V _{DC}
Power Supply Current V _p = 9v		9.1		MA
V _p = 9v **		11	18	MA
V _p = 14v **		18	27	MA
Master Clock Frequency		720K		Hz
Master Clock Resistor Value (MCRC)	6.5K			OHM
Master Clock Capacitor Value (MCRC)			300	pf

* With CB, AF, AO connector for Class B audio amplifier (see APPLICATION NOTES)

Note: TP1, TP2, TP3 must be left open for normal operation.

Votrax® reserves the right to alter its product line at any time, or change specifications or design without notice and without obligation.

Table 5. Absolute Maximum Ratings

ABSOLUTE MAXIMUM RATINGS *

RATING	SYMBOL	VALUE	UNIT
Power Supply Voltage	V_p	20	V_{DC}
Power Dissipation at 25°C	P_{DM}	650	MW
Derating Above 25°C		5	MW
Operating Ambient Temperature	T_o	0 to 70	°C
Storage Temperature	T_{STG}	-55 to 125	°C
Input Voltage	V_{INM}	-0.5 to $V_p+0.5$	V_{DC}
DC Current Max. Above $V_p+0.5v$	I_{INM}	1.0	MA
Lead Temperature (soldering 10 sec.)	T_L	300	°C

* Operation above these limits could damage the device.

NORMAL OPERATING CONDITIONS: $V_v = V_p = 14v$, $0^\circ C = T_o = 70^\circ C$

COLOR GRAPHICS SOUND EFFECTS

VOICE SYNTHESIS

ALL ON ONE BOARD

FOR THE HEATH-ZENITH 89 COMPUTER

COLOR GRAPHICS

- USES TMS 9918A
- 256 x 192 DOTS
- 16 COLORS
- 16 K ON-BOARD VIDEO RAM

A/D CONVERTER

- 8 CHANNELS
- 8 BIT RESOLUTION

PARALLEL I/O

- 2, 8 BIT PORTS
- EACH PORT EITHER INPUT OR OUTPUT

COUNTER-TIMERS

- USES 8253
- 3, 16 BIT COUNTERS
- PROGRAMMABLE

OPTION NO. 1 **\$130***
OPTION NO. 2 **\$210***
OPTION NO. 4 **\$110***

*PRICES SHOWN FOR OPTIONS
PURCHASED WITH BASIC BOARD.
ADD \$15 IF PURCHASED AT A
LATER DATE

SOUND EFFECTS

- USES AY-3-8910
- 3 TONE AND 1 NOISE CHANNEL
- PROGRAMMABLE
- ENVELOPE CONTROL

PRIORITY INTERRUPTS

- USES 8259
- 8 MASKABLE INTERRUPTS

VOICE SYNTHESIS (OPT 1)

- USES VOTRAX SC-01A
- PHONEME ORIENTED
SYNTHESIS

ARITHMETIC PROCESSOR (OPT 2)

- USES 9511A/9512/8231A/8232
- INTEGER AND FLOATING POINT
- TRANSCENDENTAL FUNCTIONS

BASIC BOARD **\$399** P/N HA-89-3
ASSEMBLED AND TESTED
SPECIFY H-17/37/47, CP/M™ OR HDOS

PASCAL/MT+ AND CP/M ARE A TRADE
MARK OF DIGITAL RESEARCH

D/A CONVERTERS (OPT 4)

- 2 CHANNELS
- 12 BIT RESOLUTION
- PRECISION REFERENCE

SOFTWARE SUPPORT

- ROUTINES FOR ALL FEATURES
- COMPATABLE WITH HA-8-3
SUPPORT ROUTINES

PASCAL/MT+™

- ALSO AVAILABLE FROM **NOGDS**
- PROGRAM DEVICES WITH
PASCAL

NEW ORLEANS

**GENERAL
DATA
SERVICES,
INC**

7230 CHADBOURNE DR
NEW ORLEANS LA 70126
(504) 241-9495

Software Support For HA-89-3
New Orleans General Data Services, Inc.

Micro-Doc, 3108 Jackson Street, Bellevue, Nebraska 68005, (402) 271-0795

Offers an enhanced version of Tiny Pascal with very fast curve drawing algorithm's. This product is priced very nicely. Micro-Doc also offers an excellent subroutine library for Pascal/MT+.

Polybytes, 325 19th St., S.E., Cedar Rapids, Iowa 52403, (319) 395-4131

Polybytes offers a graphics version of Lucidata Pascal. The curve drawing routines in this package are also quite good. Polybytes has been offering Lucidata Pascal for some time now. It is a proven and reliable product.

Far Field Software, 1031 Rue Verand, Slidell, La. 70458, (504) 255-3979

Far Field Software offers a CAD/CAM package for the HA-89-3. They also have a program which plots the TMS-9918A video RAM on an Epson MX-80 printer (in black and white).

COLORWORKS, 5337 E. Bellevue Ave., Tucson, Az. 85712, (602) 325-0096

The COLORWORKS offers two color editors and a slide show for the HA-89-3. The editors are conversational and very easy to use. They allow the preparation of graphs, figures, and text with no need for programming knowledge. They provide a demo disk for HA-89-3 owners. They have other products under development.

Hoyle & Hoyle Software, 716 South Elam Ave., Greensboro, NC 27403, (919) 378-1050

Hoyle & Hoyle Software offers a game called Brick Break. It is an action game which makes good use of the sound effects as well as the color graphics ability of the HA-89-3. This game does not require a joy stick.

Future plans include a release of the Raiders program from the Heath Users Group and a FORTRAN support library.

H A - 8 9 - 3
C o l o r G r a p h i c s B o a r d
New Orleans General Data Services, Inc.

The HA-89-3 Color Graphics Board uses the TMS-9918A Color Video Display Generator to add color graphics to the H/Z-89 computer. Also included is a Programmable Sound Generator (PSG) for generating sound effects. An 8-input analog-to-digital converter allows for 4 X-Y joystick consoles (not included). Two 8-bit parallel I/O ports are provided which may be used for controlling lights and switches. In addition, it includes a three channel counter/timer (8253) and a priority interrupt controller (8259A). Three options are also available. These include an arithmetic processor (9511A or 9512), a pair of 12 bit analog to digital converters and a Votrax[®] phoneme speech synthesizer.

The HA-89-3 Color Graphics Board was designed with as many functions as possible on one PC board because of the limited number of I/O slots on the H/Z-89 backplane. To accomplish the required circuit density, a considerable amount of logic was placed in two Programmable Array Logic (PAL[®]) chips. In order to reduce the cost of the board, some of the functions are provided as options which could either be obtained with the purchase of the board or added later when the need arises.

The board comes with a shunt to select which card slot is to be used and is supplied with sample user routines, diagnostic software, and demonstration programs. Routines to provide software support for this board will be available for both HDOS and CP/M[®] versions of Pascal/MT+[®] thru NOGDS.

Each HA-89-3 Color Graphics Board is "burned-in" for 100 hours, tested before shipping and warranted against defects for a period of 90 days from the date of purchase. The minimum recommended configuration is a 32k H/Z-89 with 200K of disk.

Enclosed is a photograph of the board, which has been fully configured with all three options. Should you need additional information concerning features of the board or software support, please contact us at:

New Orleans General Data Services, Inc.
7230 Chadbourne Drive
New Orleans, La. 70126
(504) 241-9495

Votrax[®] is a registered trademark of Federal Screw Works
PAL[®] is a registered trademark of Monolithic Memories
CP/M[®] and Pascal/MT+[®] are registered trademarks of Digital Research

NOGDS HA-89-3 Color Graphics Board

SPECIFICATIONS

Power Requirements for HA-89-3 board with all three options

+5 Volt Supply	1000 ma (typ)	1300 ma (max)
+12 Volt Supply	100 ma (typ)	150 ma (max)
-12 Volt Supply	60 ma (typ)	100 ma (max)

Note: Power requirements without Am9511A/8231A

+12 Volt Supply	50 ma (typ)	75 ma (max)
-----------------	-------------	-------------

Temperature

Operating Temperature	0 C (min)	40 C (max)
-----------------------	-----------	------------

Interfaces

H89 Bus Data Bus Transceivers	74LS245
H89 Bus Other Inputs	2 LS TTL loads or less
H89 Bus Open Collector Driver	7406

Video Amplifier Output Impedance	75 Ohms
Video Output	Adjustable to NTSC composite video

Audio Output Impedance	1K Ohms (min)
Audio Output (PSG and PSS)	1 Volt peak-to-peak (typ)
Audio output (Mixer)	1 Volt peak-to-peak (typ)
Note: Mixer output nominally 1 Volt, adjustable from 0 to 2 Volts	

A/D Input Impedance	10 K (typ)
---------------------	------------

Parallel Inputs	Internal Pullups (From AY-3-8910)
-----------------	--------------------------------------

Parallel Output	1 TTL Load
-----------------	------------

NOGDS HA-89-3 Color Graphics Board

VDP Section

Video Display Processor	TMS-9918A
RAM	16K X 8
Resolution	256 X 192 pixels
Colors	15 + Transparent
Number of X,Y Movable Sprites	32
Text Mode	24 X 40 characters

PSG Section

Programmable Sound Generator	AY-3-8910
Tone Channels	3
Noise Channels	1
Envelope Generator	1
Parallel I/O Ports	2 - 8 bit
Output Impedance	1K Ohms (min)

A/D Section

Analog Multiplexer/Converter	ADC0809
Analog Input Channels	8
A/D Resolution	8 bits, binary, unipolar
A/D Accuracy	+/-1 LSB
A/D Convert Time	83 us (max)
Analog Mux Delay Time	2.5 us (max)

CTC Section

Counter/Timer Chip	8253
Counter Resolution	16 bits
Number of Counters	3
Counter 0 input	1.79 MHz
Counters 1 and 2 input	external

PIC Section

Programmable Interrupt Controller	8259A
Number of Priority Interrupts supported	7
Number of Interrupts Maskable	All
CPU Interrupt Vector	Selectable

NOGDS HA-89-3 Color Graphics Board

APU Section (optional)

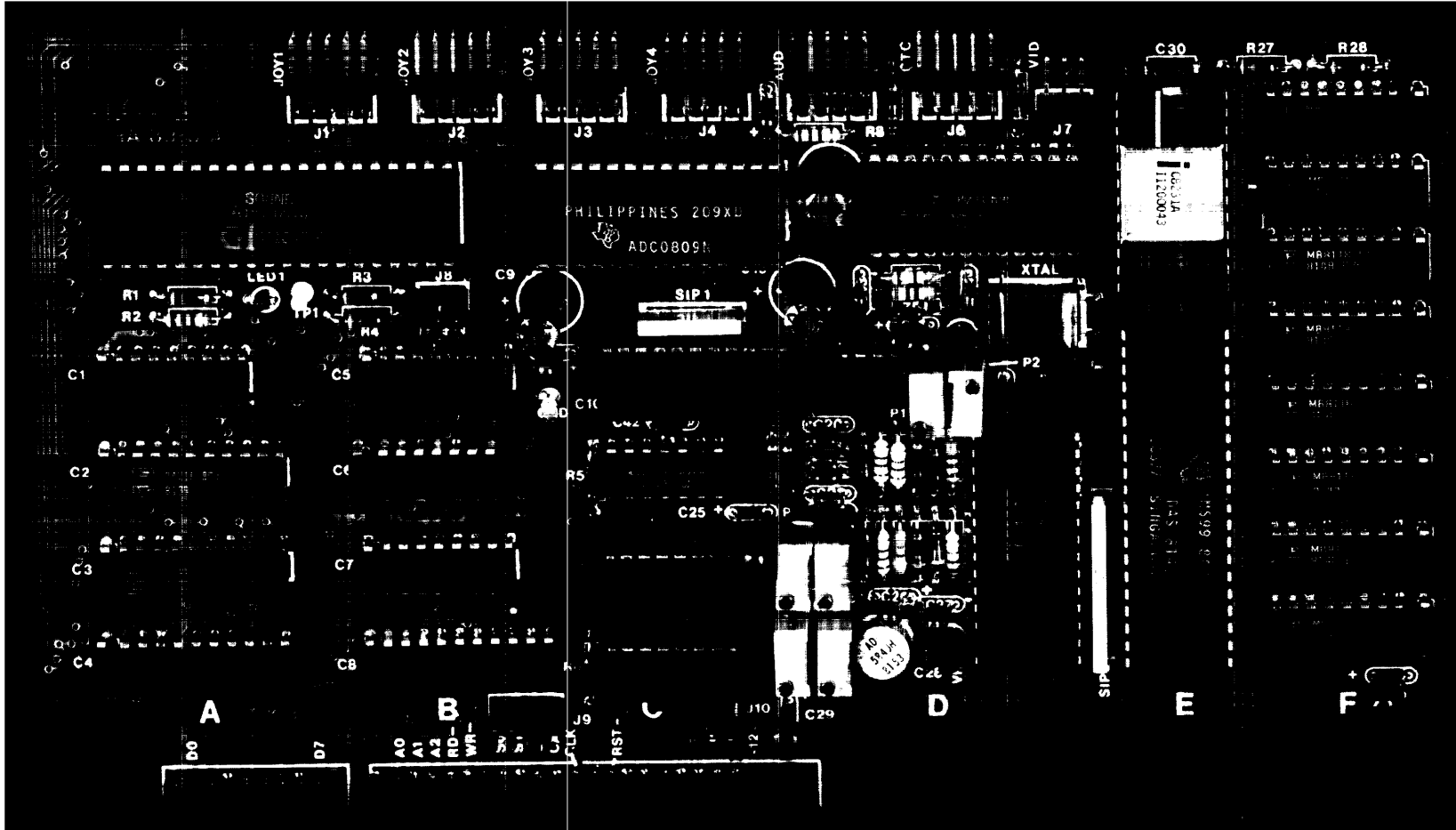
Arithmetic Processing Unit	Am9511A/8231A
Fixed Point Operations	16 and 32 bit
Floating Point Operations	32 bit
Arithmetic Operations	Add, Subtract, Multiply and Divide
Other Operations	Trig, Inv-Trig, Sqrt, Log, Exp, Float-fix conv
Floating point Divide, 32 bit with 3.58MHz Clock	43 to 51 us
Cosine, 32 bit Floating Point with 3.58MHz Clock	1150 us

D/A Section (optional)

Digital to Analog Converter	AD7542
Number of D/A Converters	2
Resolution	12 bits
Accuracy (Tmin to Tmax)	+/- 1 LSB
Reference Voltage Source (5.0 +/- 15mv)	AD584J
Output Impedance	10K Ohms (min)

PSS Section (optional)

Phoneme Speech Synthesizer	Votrax SC-01
Number of Phonemes	64
Master Clock Rate	Adjustable
Software setable Master Clock Rates	2
Software setable Inflection levels	4
Output Impedance	1K ohms (min)



GRAPH PLOTTING ON TV

ORG 0100H
JMP PLOT

```
      'HA-89-3 Board address equates'
```

```
HA-89-3 Board address equates.  All other routines use these equates  
to do I/O to the HA-89-3.  Therefore if you change the address of  
the HA-89-3 only the equate labeled "HAB93L" must be changed.  All  
other routines will use this address.
```

```
HAB93L EQU      3200           ;Address of device latch on HA-89-3  
HAB93D EQU      HAB93L+4     ;All devices on the HA-89-3 will be here
```

'Am9511A Definitions'

AMD Am9511A Definitions

I/O Address

APUADR EQU 7 ;APU is device 7 on HA-89-3

Status Register

AP\$BSY EQU 10000000B ;Busy
AP\$NEG EQU 01000000B ;TOS nesative
AP\$ZER EQU 00100000B ;TOS zero
AP\$ERC EQU 00011110B ;Error code mask
AP\$CRY EQU 00000001B ;Carry/borrow

Error codes

AP\$DZE EQU 00010000B ;Divide by zero
AP\$SRN EQU 00001000B ;Sqrt of nes number
AP\$ARG EQU 00011000B ;Invalid arsument to function
AP\$UFL EQU 00000100B ;Underflow
AP\$DFL EQU 00000010B ;Overflow

16-Bit Fixed-point operations

A\$SADD EQU 6CH
A\$SSUB EQU 6DH
A\$SMUL EQU 6EH
A\$SMUU EQU 76H
A\$SDIV EQU 6FH

32-Bit Fixed-point operations

A\$DADD EQU 2CH
A\$DSUB EQU 2DH
A\$DMUL EQU 2EH
A\$DMUU EQU 36H
A\$DDIV EQU 2FH

32-Bit Floatins-point Primary Operations

A\$FADD EQU 10H
A\$FSUB EQU 11H
A\$FMUL EQU 12H
A\$FDIV EQU 13H

32-Bit Floatins-point Derived Operations

A\$SQRT EQU 01H
A\$SIN EQU 02H
A\$COS EQU 03H
A\$TAN EQU 04H
A\$ASIN EQU 05H
A\$ACOS EQU 06H
A\$ATAN EQU 07H
A\$LOG EQU 08H
A\$LN EQU 09H
A\$EXP EQU 0AH
A\$PWR EQU 0BH

Data and stack manipulation operations

```
;  
A$NOP EQU 00H  
A$FIXS EQU 1FH  
A$FIXD EQU 1EH  
A$FLTS EQU 1DH  
A$FLTD EQU 1CH  
A$CHSS EQU 74H  
A$CHSD EQU 34H  
A$CHSF EQU 15H  
A$PTOS EQU 77H  
A$PTOD EQU 37H  
A$PTOF EQU 17H  
A$POPS EQU 78H  
A$POPD EQU 38H  
A$POPF EQU 18H  
A$XCHS EQU 79H  
A$XCHD EQU 39H  
A$XCHF EQU 19H  
A$PUPI EQU 1AH  
;  
; Service Request flag  
;  
A$SVRQ EQU 80H  
;
```

'Am9511A Subroutines'

AP\$CMD - Execute a command

Registers:

A: Command to be executed

All registers preserved

```
AP$CMD EQU $
        PUSH PSW ;Save PSW
        CALL AP$WAT ;Wait until APU not busy
        POP PSW ;Restore PSW
        OUT HAB93D+1 ;Issue command
        RET ;Return to caller
```

AP\$LSD - Load (from stack) 16-bit operand

Registers:

HL: Pointer to low order byte of operand

All registers preserved

```
AP$LSD EQU $
        PUSH PSW ;Save PSW
        CALL AP$WAT ;Wait until APU not busy
        INX H ;Point to high order byte
        IN HAB93D ;Pop high order byte
        MOV M,A ;Move high order byte to memory
        DCX H ;Point to low order byte
        IN HAB93D ;Pop low order byte
        MOV M,A ;Move low order byte to memory
        POP PSW ;Restore PSW
        RET ;Return to caller
```

AP\$LDD - Load (from stack) 32-bit operand

Registers:

HL: Pointer to low order byte

All registers preserved

```
AP$LDD EQU $
        PUSH PSW ;Save PSW
        PUSH B ;Save BC
        CALL AP$WAT ;Wait until APU not busy
        LXI B,4 ;4 bytes in 32-bit operand
        DAD B ;Point 1 past high order byte
```

Popins loop

```
        DCX H ;Point to byte
        IN HAB93D ;Pop byte for stack
        MOV M,A ;Move to memory
        DCR C ;Count this byte
        JNZ $-5 ;Loop 4 times

        POP B ;Restore BC
        POP PSW ;Restore PSW
        RET ;Return to caller
```


AP\$LSR - Load (from stack) 16-bit operand to HL

Registers:

HL: 16-bit operand stored here

Register HL is destroyed

```
AP$LSR EQU $
        PUSH   PSW           ;Save PSW
        CALL   AP$WAT        ;Wait until APU not busy
        IN     HAB93D        ;Pick up low order byte
        MOV    L,A           ;Store low order byte
        IN     HAB93D        ;Pick up high order byte
        MOV    H,A           ;Store high order byte
        POP    PSW          ;Restore PSW
        RET                    ;Return to caller
```

AP\$SSD - Store (on stack) 16-bit operand

Registers:

HL: Pointer to low order byte of operand

All registers preserved

```
AP$SSD EQU $
        PUSH   PSW           ;Save PSW
        CALL   AP$WAT        ;Wait until APU not busy
        MOV    A,M           ;Pick up low order byte
        OUT    HAB93D        ;Push low order byte
        INX    H             ;Point to high order byte
        MOV    A,M           ;Pick up high order byte
        OUT    HAB93D        ;Push high order byte
        DCX    H             ;Restore HL
        POP    PSW          ;Restore PSW
        RET                    ;Return to caller
```

AP\$SDO - Store (on stack) 32-bit operand

Registers:

HL: Pointer to low order byte of operand

All registers preserved

```
AP$SDO EQU $
        PUSH   PSW           ;Save PSW
        PUSH   H             ;Save HL
        PUSH   B             ;Save BC
        CALL   AP$WAT        ;Wait until APU not busy
        MVI    B,4           ;4 bytes in 32-bit operand
```

Stacking loop

```
        MOV    A,M           ;Pick up byte
        OUT    HAB93D        ;Stack byte
        INX    H             ;Point to next byte
        DCR    B             ;Count this byte
        JNZ    $-5           ;Loop for 4 bytes
```

```
        POP    B             ;Restore BC
```

```
POP      H           ;Restore HL
POP      PSW         ;Restore PSW
RET      ;Return to caller
```

```
AP$SSR - Store (on stack) 16-bit operand in HL
```

```
Registers:
```

```
HL:      16-bit operand to be stacked
```

```
All registers preserved
```

```
AP$SSR EQU $
PUSH   PSW           ;Save PSW
CALL   AP$WAT        ;Wait until APU not busy
MOV    A,L           ;Pick up low order byte
OUT    HA893D        ;Stack low order byte
MOV    A,H           ;Pick up high order byte
OUT    HA893D        ;Stack high order byte
POP    PSW           ;Restore PSW
RET    ;Return to caller
```

```
AP$STS - Get APU status
```

```
Registers:
```

```
A:      Status of APU
```

```
Register A is destroyed
```

```
AP$STS EQU $
MVI    A,APUADR      ;Pick up APU device number
OUT    HA893L        ;Latch device number
IN     HA893D+1      ;Get status
RET    ;Return to caller
```

```
AP$WAT - Wait until APU not busy
```

```
Registers: none
```

```
Register A is destroyed
```

```
AP$WAT EQU $
MVI    A,APUADR      ;Pick up APU device number
OUT    HA893L        ;Latch device number
AP$WT2 IN    HA893D+1 ;Get APU status
ANI    AP$BSY        ;Check for busy
RZ     ;Return if not busy
JMP    AP$WT2        ;Loop until not busy
```

'Definitions for TMS-9918 VDP'

Definitions for TMS-9918 Video Display Processor

I/O address for VDP

VDPADR EQU 1 ;VDP device number

Registers

External Video Register

VP\$EVR EQU 0 ;External video register

VP\$NEV EQU 00000000B ;No external video

VP\$EV EQU 00000001B*256 ;Enable external video

VP\$G2M EQU 00000010B*256 ;Graphics 2 mode (TMS-9918A only)

Option Control Register

VP\$OCR EQU 1 ;Option control register

VP\$4K EQU 00000000B ;4K RAMs

VP\$16K EQU 10000000B ;16K RAMs

VP\$DDP EQU 00000000B ;Blank display

VP\$EDP EQU 01000000B ;Enable display

VP\$DI EQU 00000000B ;Disable interrupts

VP\$EI EQU 00100000B ;Enable interrupts

VP\$PM EQU 00000000B ;Pattern mode

VP\$G1M EQU VP\$PM ;Graphics 1 mode (TMS-9918A only)

VP\$MCM EQU 00001000B ;Multicolor mode

VP\$TM EQU 00010000B ;Text mode

VP\$S0 EQU 00000000B ;Size 0 (8X8 bit) sprites

VP\$S1 EQU 00000010B ;Size 1 (16X16 bit) sprites

VP\$M0 EQU 00000000B ;Magnification 0

VP\$M1 EQU 00000001B ;Magnification 1

Name Table Base Address Register

VP\$NTR EQU 2 ;Name table base address register

VP\$NTB EQU 1024 ;Name table base divisor

Color Table Base Address Register

VP\$CTR EQU 3 ;Color table base address register

VP\$CTB EQU 64 ;Color table base divisor

Pattern Generator Table Base Address Register

VP\$PGR EQU 4 ;Pattern generator table base address register

VP\$PGB EQU 2048 ;Pattern generator table base divisor

Sprite Name Table Base Address Register

VP\$SNR EQU 5 ;Sprite name table base address register

VP\$SNB EQU 128 ;Sprite name table base divisor

VP\$SOF EQU 208 ;Sprite terminator flag

Sprite Pattern Generator Base Address Register

VP\$SGR EQU 6 ;Sprite pattern generator base address register

```

VP$SGB EQU 2048 ;Sprite pattern generator base divisor
;
; Text/Backdrop Color Resister
;
VP$TBR EQU 7 ;Text/backdrop color resister
;
; Colors
;
VC$CLR EQU 0 ;Transparent
VC$BLK EQU 1 ;Black
VC$MGR EQU 2 ;Medium sreen
VC$LGR EQU 3 ;Light sreen
VC$DBL EQU 4 ;Dark blue
VC$LBL EQU 5 ;Light blue
VC$DRD EQU 6 ;Dark red
VC$CYN EQU 7 ;Cyan
VC$MRD EQU 8 ;Medium red
VC$LRD EQU 9 ;Light red
VC$DYL EQU 10 ;Dark yellow
VC$LYL EQU 11 ;Light yellow
VC$DGR EQU 12 ;Dark sreen
VC$MAG EQU 13 ;Masenta
VC$GRY EQU 14 ;Gray
VC$WHT EQU 15 ;White
VC$LFT EQU 16 ;Multiplier for color in left nibble
;
; Status Resister
;
VP$IF EQU 10000000B ;Interrupt flas
VP$5S EQU 01000000B ;Fifth sprite flas
VP$C EQU 00100000B ;Coincidence flas
VP$FSN EQU 00011111B ;Fifth sprite number mask
;
; Sprite flass
;
VP$EC EQU 10000000B ;Early clock flas
VP$NEC EQU 00000000B ;No early clock
;

```

'VDP I/O Routines'

VDP I/O Routines

VP\$WRR - Write to VDP Register

Registers:

L: Register to be written
A: Data byte to be written to register

All registers preserved.

```
VP$WRR EQU $  
PUSH PSW ;Save PSW  
PUSH PSW ;Save PSW  
MVI A,VDPADR ;Pick up VDP device number  
OUT HAB93L ;Latch device number  
POP PSW ;Restore PSW  
OUT HAB93D+1 ;Write to control register  
MOV A,L ;Pick up register number to be written  
ANI 00000111B ;Mask out everything except register address  
ORI 10000000B ;High order bit must be on  
OUT HAB93D+1 ;Write to control register  
POP PSW ;Restore PSW  
RET ;Return to caller
```

VP\$SWA - Set VRAM write address

Registers:

HL: VRAM address to be written

All registers preserved.

```
VP$SWA EQU $  
PUSH PSW ;Save PSW  
MVI A,VDPADR ;Pick up VDP device number  
OUT HAB93L ;Latch device number  
MOV A,L ;Pick up low order byte of RAM address  
OUT HAB93D+1 ;Write to control port  
MOV A,H ;Pick up high order byte of RAM address  
ANI 00111111B ;Mask out unused address bits  
ORI 01000000B ;Set VRAM address code  
OUT HAB93D+1 ;Write to control port  
POP PSW ;Restore PSW  
RET ;Return to caller
```

VP\$WVD - Write to VRAM direct (From HL)

Registers:

HL: VRAM address to be written
A: Byte to be written

All registers preserved.

```
VP$WVD EQU $  
CALL VP$SWA ;Set write address  
CALL VP$WRV ;Write to VRAM  
RET ;Return to caller
```

VP\$WRV - Write to VRAM

Registers:

A: Data to be written

All registers preserved.

```
VP$WRV EQU $
        PUSH PSW           ;Save PSW
        MVI A,VDPADR       ;Pick up VDP device number
        OUT HAB93L         ;Latch device number
        POP PSW            ;Restore PSW
        OUT HAB93D         ;Write to data port
        RET                ;Return to caller
```

VP\$RDR - Read VDP register

Registers:

A: Status register returned here

Register A is destroyed.

```
VP$RDR EQU $
        MVI A,VDPADR       ;Pick up VDP device number
        OUT HAB93L         ;Latch device number
        IN HAB93D+1        ;Read status register
        RET                ;Return to caller
```

VP\$SRA - Set VRAM read address

Registers:

HL: VRAM address to be read

All registers preserved.

```
VP$SRA EQU $
        PUSH PSW           ;Save PSW
        MVI A,VDPADR       ;Pick up VDP device number
        OUT HAB93L         ;Latch device number
        MOV A,L             ;Pick up low order byte of RAM address
        OUT HAB93D+1       ;Write to control port
        MOV A,H             ;Pick up high order byte of RAM address
        ANI 00111111B      ;Set flas
        OUT HAB93D+1       ;Write to control port
        POP PSW            ;Restore PSW
        RET                ;Return to caller
```

VP\$RVD - Read from VRAM direct (from HL)

Registers:

HL: VRAM address to be read

A: Data from VRAM stored here

Register A destroyed.

```
VP$RVD EQU $
        CALL VP$SRA        ;Set read address
        CALL VP$RDR        ;Read from VRAM
        RET                ;Return to caller
```

;
;
VP\$RDV - Read from VRAM

Registers:

A: Data from VRAM stored here

Register A is destroyed.

;
;
VP\$RDV EQU \$
MVI A, VDPADR ;Pick up VDP device number
OUT HA893L ;Latch device number
IN HA893D ;Read from data port
RET ;Return to caller
;
;

'VDP Register maintenance routines'

Register maintenance routines

In all of the following routines, register HL contains the address of the appropriate table.

These routines preserve all registers

VP\$SOP - Set options

```
VP$SOP EQU $
PUSH PSW ;Save PSW
SHLD VP$OPT ;Save VDP options
MOV A,L ;Move in options control register
MVI L,VP$OCR ;Select option control register
CALL VP$WRR ;Write option control register
MOV A,H ;Move in external video register
MVI L,VP$EVR ;Select external video register
CALL VP$WRR ;Write external video register
POP PSW
RET
```

Options

```
VP$OPT DW 0 ;VDP options stored here
```

VP\$STB- Set text/backdrop

```
VP$STB EQU $
PUSH H ;Save HL
STA VP$TBC ;Save text/backdrop color
MVI L,VP$TBR ;Select text/backdrop register
CALL VP$WRR ;Write text/backdrop color
POP H ;Restore HL
RET
```

Text/backdrop

```
VP$TBC DB 0 ;Text/backdrop color stored here
```

VP\$SPN - Set pattern name table base address register

```
VP$SPN EQU $
PUSH PSW ;Save PSW
PUSH H ;Save HL
SHLD VP$PNT ;Store address of pattern name table
```

Move high order 4 bits of address to register A

```
MOV A,H ;Pick up high order address
RRC ;High order 5 bits
RRC ;High order 4 bits
```

Write pattern name table register

```
MVI L,VP$NTR ;Pattern name table register
CALL VP$WRR ;Write it to VDP
```

Restore registers and return


```

POP      H          ;Restore HL
POP      PSW        ;Restore PSW
RET      ;Return to caller
;
; Address of pattern name table in VRAM
;
VP$PNT  DW      0          ;address of pattern name table in VRAM
;
; VP$SPG - Set pattern generator table base address register
;
VP$SPG  EQU      $
        PUSH    PSW        ;Save PSW
        PUSH    H          ;Save HL
        SHLD   VP$PGT      ;Store address of pattern generator table
;
; Move high order 3 bits to register A
;
        MOV     A,H        ;Pick up high order address
        RRC    ;High order 5 bits
        RRC    ;High order 4 bits
        RRC    ;High order 3 bits
;
        PUSH   PSW
        PUSH   H
        LHLD  VP$OPT
        XRA   A
        CMP   H
        JZ    NOTG2M1
        ORI   3          ;SET LSB'S FOR G2 MODE
NOTG2M1:
        POP   H
        MOV   H,A
        POP   PSW
        ORA   H
;
; Write pattern generator table register
;
        MVI   L,VP$PGR    ;Pattern generator table register
        CALL  VP$WRR      ;Write it to VDP
;
; Restore registers and return
;
        POP   H          ;Restore HL
        POP   PSW        ;Restore PSW
        RET   ;Return to caller
;
; Address of pattern generator table in VRAM
;
VP$PGT  DW      0          ;Address of pattern generator table in VRAM
;
; VP$SSN - Set sprite name table base address register
;
VP$SSN  EQU      $
        PUSH   PSW        ;Save PSW
        PUSH   H          ;Save HL
        SHLD  VP$SNT      ;Store address of sprite name table
;
; Move high order 7 bits of address to register A
;
        MOV   A,L        ;Low order byte
        RLC   ;Get seventh bit out of low order byte
        MOV   A,H        ;High order byte
        RAL   ;Put seventh bit in register A

```

```

;
; Write sprite name table register
;
MVI    L,VP$SNR      ;Sprite name table register
CALL   VP$WRR        ;Write it to VDP
;
; Restore registers and return
;
POP    H              ;Restore HL
POP    PSW            ;Restore PSW
RET                                         ;Return to caller
;
; Address of sprite name table in VRAM
;
VP$SNT DW    0        ;Address of sprite name table in VRAM
;
; VP$SSG - Set sprite generator table base address register
;
VP$SSG EQU    $
PUSH   PSW           ;Save PSW
PUSH   H              ;Save HL
SHLD   VP$SGT        ;Store address of sprite generator table
;
; Move high order 3 bits of address to register A
;
MOV    A,H           ;Pick up high order address
RRC                                         ;High order 5 bits
RRC                                         ;High order 4 bits
RRC                                         ;High order 3 bits
;
; Write sprite generator table register
;
MVI    L,VP$SGR      ;Sprite generator table register
CALL   VP$WRR        ;Write it
;
; Restore registers and return
;
POP    H              ;Restore HL
POP    PSW            ;Restore PSW
RET                                         ;Return to caller
;
; Address of sprite generator table in VRAM
;
VP$SGT DW    0        ;Address of sprite generator table in VRAM
;
; VP$SCG - Set color generator table base address register
;
VP$SCG EQU    $
PUSH   PSW           ;Save PSW
PUSH   H              ;Save HL
SHLD   VP$CGT        ;Store address of color generator table
;
; Move high order 8 bits to register A
;
MOV    A,L           ;Low order address
RLC
RLC
MOV    L,A           ;Register L now set up
MOV    A,H           ;High order address
RLC
RLC
;
PUSH   PSW

```

```

    PUSH    H
    LHLD   VP$OPT
    XRA    A
    CMP    H
    JZ     NOTG2M2
    ORI    7FH      ;SET LSB'S FOR G2 MODE
    MVI    L,0
NOTG2M2:
    POP    H
    MOV    H,A
    POP    PSW
    ORA    H
;
    ADD    L          ;High order 8 bits now in register A
;
;   Write color generator table register
;
    MVI    L,VP$CTR    ;Color generator table register
    CALL   VP$WRR      ;Write it
;
;   Restore registers and return
;
    POP    H          ;Restore HL
    POP    PSW        ;Restore PSW
    RET              ;Return to caller
;
;   Address of color generator table in VRAM
;
VP$CGT DW    0          ;Address of color generator table in VRAM
;
;

```

'VRAM BLOCK MOVE SUBROUTINES'

VP\$MTV - MOVE BLOCK TO VRAM

REGISTERS:

HL: VRAM STARTING ADDRESS
DE: START MEMORY ADDRESS
BC: NUMBER OF BYTES TO MOVE

ALL REGISTERS PRESERVED

VP\$MTV EQU \$
PUSH PSW ;SAVE PSW
PUSH B ;SAVE BC
PUSH D ;SAVE DE
CALL VP\$SWA ;SET WRITE ADDRESS

MOVE LOOP

LDAX D ;PICK UP A BYTE
CALL VP\$WRV ;WRITE TO VRAM
INX D ;POINT TO NEXT MEMORY LOCATION
DCX B ;COUNT ONE BYTE
MOV A,B
ORA C ;COUNT ZERO?
JNZ \$-8 ;NO, THEN LOOP

EXIT

POP D ;RESTORE DE
POP B ;RESTORE BC
POP PSW ;RESTORE PSW
RET ;RETURN TO CALLER

VP\$VTM - BLOCK MOVE FROM VRAM

REGISTERS:

HL: STARTING VRAM ADDRESS
DE: STARTING MEMORY ADDRESS
BC: BYTE COUNT

ALL REGISTERS PRESERVED

VP\$VTM EQU \$
PUSH PSW ;SAVE PSW
PUSH B ;SAVE BC
PUSH D ;SAVE DE
CALL VP\$SRA ;SET (HL) AS READ ADDRESS

MOVE LOOP

CALL VP\$RDV ;GET BYTE FROM VRAM
STAX D ;STORE IN MEMORY
INX D ;POINT TO NEXT BYTE IN MEMORY
DCX B ;COUNT ONE BYTE
MOV A,B
ORA C ;COUNT EQUAL ZERO?
JNZ \$-8 ;NO, THEN LOOP

EXIT

POP
POP
POP
RET

D
B
PSW

;RESTORE DE
;RESTORE BC
;RESTORE PSW
;RETURN TO CALLER

;

'VDP SPRITE PROCESSING SUBROUTINES'

VP\$SSA - GET SPRITE ATTRIBUTE ADDRESS

REGISTERS:

A: SPRITE NUMBER
HL: BASE VRAM ADDRESS OF SPRITE ATTRIBUTES

REGISTER HL IS DESTROYED

```
VP$SAA EQU $
PUSH PSW ;SAVE PSW
ADD A ;SPRITE # * 2
ADD A ;SPRITE # * 4
LHLD VP$SNT ;BASE ADDRESS OF SPRITE NAME TABLE
ORA L ;ADD IN OFFSET
MOV L,A ;AND RETURN TO VRAM ADDRESS
POP PSW ;RESTORE PSW
RET ;RETURN TO CALLER
```

VP\$SSC - SET SPRITE COORDINATES

REGISTERS:

A: SPRITE NUMBER
H: X (HORIZONTAL) VALUE
L: Y (VERTICAL) VALUE

ALL REGISTERS PRESERVED

```
VP$SSC EQU $
PUSH PSW ;SAVE PSW
PUSH H ;SAVE HL
CALL VP$SAA ;GET BASE ADDRESS OF SPRITE ATTRIBUTE
CALL VP$SWA ;THIS IS STARTING WRITE ADDRESS
POP H ;RESTORE HL
MOV A,L ;PICK UP VERTICAL VALUE
CALL VP$WRV ;WRITE TO SPRITE ATTRIBUTE
MOV A,H ;PICK UP HORIZONTAL VALUE
CALL VP$WRV ;WRITE TO SPRITE ATTRIBUTE
POP PSW ;RESTORE PSW
RET ;RETURN TO CALLER
```

VP\$SSA - SET SPRITE ATTRIBUTES

REGISTERS:

A: SPRITE NUMBER
H: X (HORIZONTAL) VALUE
L: Y (VERTICAL) VALUE
D: COLOR/EC
E: PATTERN NAME

ALL REGISTERS PRESERVED

```
VP$SSA EQU $
PUSH PSW ;SAVE PSW
PUSH H ;SAVE HL
CALL VP$SAA ;GET SPRITE ADDRESS
CALL VP$SWA ;SET THIS AS WRITE ADDRESS
POP H ;RESTORE HL
```


A: SPRITE NUMBER
HL: ADDRESS OF FIRST BYTE OF PATTERN STORED HERE

REGISTER HL DESTROYED

```
UP$SGA EQU $  
PUSH PSW ;SAVE PSW  
PUSH D ;SAVE DE  
CALL UP$SAA ;GET ATTRIBUTE ADDRESS  
INX H ;X ADDRESS  
INX H ;PATTERN NAME  
CALL UP$RVD ;READ PATTERN NAME  
MOV L,A ;MOVE NAME TO L  
MVI H,0 ;MOVE NAME TO HL  
DAD H ;NAME * 2  
DAD H ;NAME * 4  
DAD H ;NAME * 8  
XCHG ;MOVE OFFSET TO DE  
LHLD UP$SGT ;PICK UP BEGINNING ADDRESS OF SPRITE GENERATOR  
DAD D ;BASE ADDRESS OF THIS SPRITE  
POP D ;RESTORE DE  
POP PSW ;RESTORE PSW  
RET ;RETURN TO CALLER
```



```

;           'PSG DEFINITIONS'
;
;           PSG I/O ADDRESS
;
PSGADR EQU      2           ;PSG DEVICE NUMBER
;
;           BIT MASKS FOR ENABLE REGISTER
;
PS$PBI EQU      10000000B   ;PORT B INPUT
PS$PBD EQU      00000000B   ;PORT B OUTPUT
PS$PAI EQU      01000000B   ;PORT A INPUT
PS$PAD EQU      00000000B   ;PORT A OUTPUT
PS$ENC EQU      00100000B   ;ENABLE NOISE ON CHANNEL C
PS$DNC EQU      00000000B   ;DISABLE NOISE ON CHANNEL C
PS$ENB EQU      00010000B   ;ENABLE NOISE ON CHANNEL B
PS$DNB EQU      00000000B   ;DISABLE NOISE ON CHANNEL B
PS$ENA EQU      00001000B   ;ENABLE NOISE ON CHANNEL A
PS$DNA EQU      00000000B   ;DISABLE NOISE ON CHANNEL A
PS$ETC EQU      00000100B   ;ENABLE TONE ON CHANNEL C
PS$DTC EQU      00000000B   ;DISABLE TONE ON CHANNEL C
PS$ETB EQU      00000010B   ;ENABLE TONE ON CHANNEL B
PS$DTB EQU      00000000B   ;DISABLE TONE ON CHANNEL B
PS$ETA EQU      00000001B   ;ENABLE TONE ON CHANNEL A
PS$DTA EQU      00000000B   ;DISABLE TONE ON CHANNEL A
;
;           BIT MASKS FOR AMPLITUDE CONTROL REGISTERS
;
PS$FLA EQU      00000000B   ;FIXED LEVEL AMPLITUDE
PS$VLA EQU      00010000B   ;VARIABLE LEVEL AMPLITUDE
;
;           BIT MASKS FOR ENVELOPE CONTROL REGISTER
;
PS$CNT EQU      00001000B   ;CONTINUE
PS$ATT EQU      00000100B   ;ATTACK
PS$ALT EQU      00000010B   ;ALTERNATE
PS$HLD EQU      00000001B   ;HOLD
;
;           PSG REGISTER DEFINITIONS
;
PS$ATF EQU      0           ;CHANNEL A TONE FINE
PS$ATC EQU      1           ;CHANNEL A TONE COURSE
PS$BTF EQU      2           ;CHANNEL B TONE FINE
PS$BTC EQU      3           ;CHANNEL B TONE COURSE
PS$CTF EQU      4           ;CHANNEL C TONE FINE
PS$CTC EQU      5           ;CHANNEL C TONE COURSE
PS$NPR EQU      6           ;NOISE PERIOD REGISTER
PS$ENR EQU      7           ;ENABLE REGISTER
PS$AAR EQU      8           ;CHANNEL A AMPLITUDE REGISTER
PS$BAR EQU      9           ;CHANNEL B AMPLITUDE REGISTER
PS$CAR EQU      10          ;CHANNEL C AMPLITUDE REGISTER
PS$EPF EQU      11          ;ENVELOPE PERIOD FINE
PS$EPC EQU      12          ;ENVELOPE PERIOD COURSE
PS$ECR EQU      13          ;ENVELOPE CONTROL REGISTER
PS$PAR EQU      14          ;PARALLEL PORT A
PS$PBR EQU      15          ;PARALLEL PORT B
;

```

'PSG I/O ROUTINES'

PSG I/O ROUTINES

PS\$WRR - WRITE TO PSG REGISTER

REGISTERS:

L: REGISTER TO BE WRITTEN
A: BYTE TO BE WRITTEN

ALL REGISTERS PRESERVED

```
PS$WRR EQU $  
PUSH H ;SAVE HL  
MOV H,A ;SAVE A  
MVI A,PSGADR ;PICK UP PSG DEVICE NUMBER  
OUT HAB93L ;LATCH DEVICE  
MOV A,L ;GET REGISTER NUMBER TO BE WRITTEN  
DUT HAB93D+1 ;WRITE REGISTER NUMBER  
MOV A,H ;GET BYTE TO BE WRITTEN  
OUT HAB93D ;WRITE TO PSG  
POP H ;RESTORE HL  
RET ;RETURN TO CALLER
```

PS\$RDR - READ FROM PSG REGISTER

REGISTERS:

L: REGISTER TO BE READ
A: BYTE FROM REGISTER

REGISTER A IS DESTROYED

```
PS$RDR EQU $  
MVI A,PSGADR ;PICK UP PSG DEVICE NUMBER  
OUT HAB93L ;LATCH DEVICE  
MOV A,L ;GET REGISTER TO BE READ  
OUT HAB93D+1 ;WRITE REGISTER NUMBER  
IN HAB93D ;READ FROM REGISTER  
RET ;RETURN TO CALLER
```

PS\$WTA - WRITE TONE PERIOD A

REGISTERS:

HL: TONE PERIOD FOR CHANNEL A

ALL REGISTERS PRESERVED

```
PS$WTA EQU $  
PUSH PSW ;SAVE PSW  
PUSH H ;SAVE HL  
MOV A,L ;GET FINE VALUE  
MVI L,PS$ATF ;CHANNEL A FINE REGISTER  
CALL PS$WRR ;WRITE TO REGISTER  
MOV A,H ;GET COURSE VALUE  
MVI L,PS$ATC ;CHANNEL A COURSE REGISTER
```

```
CALL    PS$WRR           ;WRITE TO REGISTER
POP     H                ;RESTORE HL
POP     PSW              ;RESTORE PSW
RET     ;RETURN TO CALLER
```

```
PS$RTA - READ TONE PERIOD A
```

```
REGISTERS:
```

```
HL:     MOST RECENT CHANNEL A TONE PERIOD RETURNED HERE
```

```
REGISTER HL DESTROYED
```

```
PS$RTA EQU    $
PUSH   PSW           ;SAVE PSW
MVI    L,PS$ATC     ;CHANNEL A COURSE TONE
CALL   PS$RDR       ;READ COURSE TONE
MOV    H,A          ;STORE IN REGISTER H
MVI    L,PS$ATF     ;CHANNEL A FINE TONE
CALL   PS$RDR       ;READ FINE TONE
MOV    L,A          ;STORE IN REGISTER L
POP    PSW           ;RESTORE PSW
RET    ;RETURN TO CALLER
```

```
PS$WTB - WRITE TONE PERIOD B
```

```
REGISTERS:
```

```
HL:     TONE PERIOD FOR CHANNEL B
```

```
ALL REGISTERS PRESERVED
```

```
PS$WTB EQU    $
PUSH   PSW           ;SAVE PSW
PUSH   H             ;SAVE HL
MOV    A,L           ;GET FINE VALUE
MVI    L,PS$BTF     ;CHANNEL B FINE REGISTER
CALL   PS$WRR       ;WRITE TO REGISTER
MOV    A,H           ;GET COURSE VALUE
MVI    L,PS$BTC     ;CHANNEL B COURSE REGISTER
CALL   PS$WRR       ;WRITE TO REGISTER
POP    H             ;RESTORE HL
POP    PSW           ;RESTORE PSW
RET    ;RETURN TO CALLER
```

```
PS$RTB - READ TONE PERIOD B
```

```
REGISTERS:
```

```
HL:     MOST RECENT CHANNEL B TONE PERIOD RETURNED HERE
```

```
REGISTER HL DESTROYED
```

```
PS$RTB EQU    $
PUSH   PSW           ;SAVE PSW
MVI    L,PS$BTC     ;CHANNEL B COURSE TONE
CALL   PS$RDR       ;READ COURSE TONE
MOV    H,A          ;STORE IN REGISTER H
MVI    L,PS$BTF     ;CHANNEL B FINE TONE
CALL   PS$RDR       ;READ FINE TONE
MOV    L,A          ;STORE IN REGISTER L
POP    PSW           ;RESTORE PSW
RET    ;RETURN TO CALLER
```

PS\$WTC - WRITE TONE PERIOD TO CHANNEL C

REGISTERS:

HL: CHANNEL C TONE PERIOD

ALL REGISTERS PRESERVED

```
PS$WTC EQU    $
        PUSH   PSW           ;SAVE PSW
        PUSH   H             ;SAVE HL
        MOV    A,L           ;GET FINE VALUE
        MVI    L,PS$CTF      ;CHANNEL C FINE REGISTER
        CALL   PS$WRR        ;WRITE TO REGISTER
        MOV    A,H           ;GET COURSE VALUE
        MVI    L,PS$CTC      ;CHANNEL C COURSE REGISTER
        CALL   PS$WRR        ;WRITE TO REGISTER
        POP    H             ;RESTORE HL
POP      PSW                 ;RESTORE PSW
        RET                    ;RETURN TO CALLER
```

PS\$RTC - READ TONE PERIOD C

REGISTERS:

HL: MOST RECENT CHANNEL C TONE PERIOD RETURNED HERE

REGISTER HL DESTROYED

```
PS$RTC EQU    $
        PUSH   PSW           ;SAVE PSW
        MVI    L,PS$CTC      ;CHANNEL C COURSE TONE
        CALL   PS$RDR        ;READ COURSE TONE
        MOV    H,A           ;STORE IN REGISTER H
        MVI    L,PS$CTF      ;CHANNEL C FINE TONE
        CALL   PS$RDR        ;READ FINE TONE
        MOV    L,A           ;STORE IN REGISTER L
        POP    PSW          ;RESTORE PSW
        RET                    ;RETURN TO CALLER
```

PS\$WNP - WRITE NOISE PERIOD

REGISTERS:

A: NOISE PERIOD VALUE

ALL REGISTERS PERSERVED

```
PS$WNP EQU    $
        PUSH   H             ;SAVE HL
        MVI    L,PS$NPR      ;NOISE PERIOD REGISTER
        CALL   PS$WRR        ;WRITE NOISE PERIOD
        POP    H             ;RESTORE HL
        RET                    ;RETURN
```

PS\$RNP - READ NOISE PERIOD VALUE

REGISTERS:

A: NOISE PERIOD VALUE RETURNED HERE

REGISTER A DESTROYED

```

;
; PS$RNP EQU $
; PUSH H ;SAVE HL
; MVI L,PS$NPR ;NOISE PERIOD REGISTER
; CALL PS$RDR ;READ NOISE PERIOD REGISTER
; POP H ;RESTORE HL
; RET ;RETURN TO CALLER

```

```

;
; PS$WER - WRITE ENABLE REGISTER
;
;
;
;
;
;
;
;

```

```

; REGISTERS:
;
;
;
;
;
;
;
;

```

```

; A: ENABLE VALUES (COMPLEMENTED)
;
;
;
;
;
;
;
;

```

```

; ALL REGISTERS PRESERVED
;
;
;
;
;
;
;
;

```

```

; PS$WER EQU $
; PUSH H ;SAVE HL
; MVI L,PS$ENR ;ENABLE REGISTER
; CMA ;COMPLEMENT VALUES
; CALL PS$WRR ;WRITE ENABLE VALUES
; CMA ;RESTORE A
; POP H ;RESTORE HL
; RET ;RETURN TO CALLER

```

```

; PS$RER - READ MOST RECENT ENABLE VALUE
;
;
;
;
;
;
;
;

```

```

; REGISTERS:
;
;
;
;
;
;
;
;

```

```

; A: MOST RECENT ENABLE VALUE RETURNED HERE
;
;
;
;
;
;
;
;

```

```

; REGISTER A DESTROYED
;
;
;
;
;
;
;
;

```

```

; PS$RER EQU $
; PUSH H ;SAVE HL
; MVI L,PS$ENR ;ENABLE REGISTER
; CALL PS$RDR ;READ ENABLE REGISTER
; CMA ;CONVERT TO EXTERNAL FORMAT
; POP H ;RESTORE HL
; RET ;RETURN TO CALLER

```

```

; PS$WAA - WRITE CHANNEL A AMPLITUDE
;
;
;
;
;
;
;
;

```

```

; REGISTERS:
;
;
;
;
;
;
;
;

```

```

; A: AMPLITUDE FOR CHANNEL A
;
;
;
;
;
;
;
;

```

```

; ALL REGISTERS PRESERVED
;
;
;
;
;
;
;
;

```

```

; PS$WAA EQU $
; PUSH H ;SAVE HL
; MVI L,PS$AAR ;CHANNEL A AMPLITUDE REGISTER
; CALL PS$WRR ;WRITE AMPLITUDE
; POP H ;RESTORE HL
; RET ;RETURN

```

```

; PS$RAA - READ CHANNEL A AMPLITUDE
;
;
;
;
;
;
;
;

```

```

; REGISTERS:
;
;
;
;
;
;
;
;

```

```

; A: MOST RECENT CHANNEL A AMPLITUDE RETURNED HERE
;
;
;
;
;
;
;
;

```

```

; REGISTER A DESTROYED
;
;
;
;
;
;
;
;

```

```
PS$RAA EQU $
        PUSH H ;SAVE HL
        MVI L,PS$AAR ;CHANNEL A AMPLITUDE REGISTER
        CALL PS$RDR ;READ AMPLITUDE
        POP H ;RESTORE HL
        RET ;RETURN TO CALLER
```

```
PS$WBA - WRITE CHANNEL B AMPLITUDE
```

```
REGISTERS:
```

```
A: AMPLITUDE FOR CHANNEL B
```

```
ALL REGISTERS PRESERVED
```

```
PS$WBA EQU $
        PUSH H ;SAVE HL
        MVI L,PS$BAR ;CHANNEL B AMPLITUDE REGISTER
        CALL PS$WRR ;WRITE AMPLITUDE
        POP H ;RESTORE HL
        RET ;RETURN
```

```
PS$RBA - READ CHANNEL B AMPLITUDE
```

```
REGISTERS:
```

```
A: MOST RECENT CHANNEL B AMPLITUDE RETURNED HERE
```

```
REGISTER A DESTROYED
```

```
PS$RBA EQU $
        PUSH H ;SAVE HL
        MVI L,PS$BAR ;CHANNEL B AMPLITUDE REGISTER
        CALL PS$RDR ;READ AMPLITUDE
        POP H ;RESTORE HL
        RET ;RETURN TO CALLER
```

```
PS$WCA - WRITE CHANNEL C AMPLITUDE
```

```
REGISTERS:
```

```
A: AMPLITUDE FOR CHANNEL C
```

```
ALL REGISTERS PRESERVED
```

```
PS$WCA EQU $
        PUSH H ;SAVE HL
        MVI L,PS$CAR ;CHANNEL C AMPLITUDE REGISTER
        CALL PS$WRR ;WRITE AMPLITUDE
        POP H ;RESTORE HL
        RET ;RETURN
```

```
PS$RCA - READ CHANNEL C AMPLITUDE
```

```
REGISTERS:
```

```
A: MOST RECENT CHANNEL C AMPLITUDE RETURNED HERE
```

```
REGISTER A DESTROYED
```

```
PS$RCA EQU $
        PUSH H ;SAVE HL
```

```
MVI    L,PS$CAR      ;CHANNEL C AMPLITUDE REGISTER
CALL   PS$RDR        ;READ AMPLITUDE
POP    H              ;RESTORE HL
RET                                ;RETURN TO CALLER
```

```
;
;
; PS$WEP - WRITE ENVELOPE PERIOD
```

```
;
;
;     REGISTERS:
```

```
;
;           HL:    ENVELOPE PERIOD
```

```
;
; ALL REGISTERS PRESERVED
```

```
;
; PS$WEP EQU    $
; PUSH   PSW      ;SAVE PSW
; PUSH   H        ;SAVE HL
; MOV    A,L      ;ENVELOPE PERIOD FINE VALUE
; MVI    L,PS$EPF ;ENVELOPE PERIOD FINE REGISTER
; CALL   PS$WRR   ;WRITE FINE VALUE
; MOV    A,H      ;ENVELOPE PERIOD COURSE VALUE
; MVI    L,PS$EPC ;ENVELOPE PERIOD COURSE REGISTER
; CALL   PS$WRR   ;WRITE COURSE VALUE
; POP    H        ;RESTORE HL
; POP    PSW      ;RESTORE PSW
; RET                                ;RETURN TO CALLER
```

```
;
; PS$REP - READ ENVELOPE PERIOD REGISTER
```

```
;
;     REGISTERS:
```

```
;
;           HL:    MOST RECENT ENVELOPE PERIOD VALUE RETURNED HERE
```

```
;
; REGISTER HL DESTROYED
```

```
;
; PS$REP EQU    $
; PUSH   PSW      ;SAVE PSW
; MVI    L,PS$EPC ;ENVELOPE PERIOD COURSE REGISTER
; CALL   PS$RDR   ;READ COURSE VALUE
; MOV    H,A      ;STORE IN REGISTER H
; MVI    L,PS$EPF ;ENVELOPE PERIOD FINE REGISTER
; CALL   PS$RDR   ;READ FINE VALUE
; MOV    L,A      ;STORE IN REGISTER L
; POP    PSW      ;RESTORE PSW
; RET                                ;RETURN TO CALLER
```

```
;
; PS$WEC - WRITE ENVELOPE CONTROL REGISTER
```

```
;
;     REGISTERS:
```

```
;
;           A:    ENVELOPE CONTROL VALUE
```

```
;
; ALL REGISTERS PRESERVED
```

```
;
; PS$WEC EQU    $
; PUSH   H        ;SAVE HL
; MVI    L,PS$ECR ;ENVELOPE CONTROL REGISTER
; CALL   PS$WRR   ;WRITE ENVELOPE CONTROL REGISTER
; POP    H        ;RESTORE HL
; RET                                ;RETURN TO CALLER
```

```
;
; PS$REC - READ ENVELOPE CONTROL REGISTER
```

```
;
;     REGISTERS:
```

A: MOST RECENT ENVELOPE CONTROL VALUE RETURNED HERE

ALL REGISTERS PRESERVED

```
PS$REC EQU $
        PUSH H ;SAVE HL
        MVI L,PS$ECR ;ENVELOPE CONTROL REGISTER
        CALL PS$RDR ;READ ENVELOPE CONTROL REGISTER
        POP H ;RESTORE HL
        RET ;RETURN
```

PS\$WPA - WRITE PARALLEL PORT A

REGISTERS:

A: VALUE WRITTEN TO PORT A

ALL REGISTERS PRESERVED

```
PS$WPA EQU $
        PUSH H ;SAVE HL
        MVI L,PS$PAR ;PORT A REGISTER
        CALL PS$WRR ;WRITE TO PORT A
        POP H ;RESTORE HL
        RET ;RETURN TO CALLER
```

PS\$RPA - READ PARALLEL PORT B

REGISTERS:

A: VALUE FROM PORT A RETURNED HERE

REGISTER A DESTROYED

```
PS$RPA EQU $
        PUSH H ;SAVE HL
        MVI L,PS$PAR ;PORT A REGISTER
        CALL PS$RDR ;READ PORT A
        POP H ;RESTORE HL
        RET ;RETURN TO CALLER
```

PS\$WPB - WRITE PARALLEL PORT B

REGISTERS:

A: VALUE WRITTEN TO PORT B

ALL REGISTERS PRESERVED

```
PS$WPB EQU $
        PUSH H ;SAVE HL
        MVI L,PS$PBR ;PORT B REGISTER
        CALL PS$WRR ;WRITE TO PORT B
        POP H ;RESTORE HL
        RET ;RETURN TO CALLER
```

PS\$RPB - READ PARALLEL PORT B

REGISTERS:

A: VALUE FROM PORT B RETURNED HERE


```
; REGISTER A DESTROYED
;
PS$RPB EQU $
        PUSH    H           ;SAVE HL
        MVI     L,PS$PBR    ;PORT B REGISTER
        CALL    PS$RDR     ;READ PORT B
        POP     H           ;RESTORE HL
        RET          ;RETURN TO CALLER
;
```

```

; PIC I/O ADDRESS
;
PICADR EQU 0 ;PIC DEVICE NUMBER
;
; ICW1 DEFINITIONS
;
PI$LTM EQU 00001000B ;LEVEL TRIGGERED MODE
PI$ETM EQU 00000000B ;EDGE TRIGGERED MODE
PI$IN4 EQU 00000100B ;4 BYTE INTERVAL
PI$IN8 EQU 00000000B ;8 BYTE INTERVAL
PI$SIN EQU 00000010B ;SINGLE PIC MODE
PI$CAS EQU 00000000B ;CASCADE MODE (NOT SUPPORTED)
PI$IC4 EQU 00000001B ;ICW4 NEEDED
PI$NC4 EQU 00000000B ;ICW4 NOT NEEDED
;
; OCW2 DEFINITIONS
;
PI$RAC EQU 00000000B ;ROTATE IN AUTOMATIC EOI MODE (CLEAR)
PI$NSE EQU 00010000B ;NON-SPECIFIC EOI COMMAND
PI$NOP EQU 00100000B ;NO OPERATION
PI$SPE EQU 00110000B ;SPECIFIC EOI COMMAND
PI$RAS EQU 01000000B ;ROTATE IN AUTOMATIC EOI MODE (SET)
PI$RNS EQU 01010000B ;ROTATE ON NON-SPECIFIC EOI COMMAND
PI$SPC EQU 01100000B ;SET PRIORITY COMMAND
PI$RSP EQU 01110000B ;ROTATE ON SPECIFIC EOI COMMAND
;
; OCW3 DEFINITIONS
;
PI$PLL EQU 00000100B ;POLL COMMAND
PI$IRR EQU 00000010B ;READ IR ON NEXT READ COMMAND
@I$ISR EQU 00000011B ;READ IS ON NEXT READ COMMAND
;

```

'PIC I/O ROUTINES'

PIC I/O ROUTINES

PI\$IW1 - LOAD INITIALIZATION COMMAND WORD 1
PI\$OW2 - LOAD OPERATION COMMAND WORD 2
PI\$OW3 - LOAD OPERATION COMMAND WORD 3

REGISTERS:

A: BYTE TO BE WRITTEN

ALL REGISTERS PRESERVED

PI\$IW1 EQU \$
PI\$OW2 EQU \$
PI\$OW3 EQU \$
PUSH PSW ;SAVE PSW
MVI A,PICADR ;PICK UP PIC DEVICE NUMBER
OUT HAB93L ;LATCH DEVICE
POP PSW ;RESTORE PSW
OUT HAB93D ;WRITE 8259 (A0=0)
RET ;RETURN TO CALLER

PI\$IW2 - LOAD INITIALIZATION COMMAND WORD 2
PI\$IW3 - LOAD INITIALIZATION COMMAND WORD 3
PI\$IW4 - LOAD INITIALIZATION COMMAND WORD 4
PI\$OW1 - LOAD OPERATION COMMAND WORD 1

REGISTERS:

A: BYTE TO BE WRITTEN

ALL REGISTERS PRESERVED

PI\$IW2 EQU \$
PI\$IW3 EQU \$
PI\$IW4 EQU \$
PI\$OW1 EQU \$
PUSH PSW ;SAVE PSW
MVI A,PICADR ;PICK UP PIC DEVICE NUMBER
OUT HAB93L ;LATCH DEVICE
POP PSW ;RESTORE PSW
OUT HAB93D+1 ;WRITE TO 8259 (A0=1)
RET ;RETURN TO CALLER

PI\$RIR - READ INTERRUPT REQUEST REGISTER

REGISTERS:

A: BYTE TO BE RETURNED

ALL REGISTERS PRESERVED

PI\$RIR EQU \$
MVI A,PICADR ;PICK UP PIC DEVICE NUMBER
OUT HAB93L ;LATCH DEVICE
IN HAB93D ;READ 8259
RET ;RETURN TO CALLER

PI\$RIM - READ INTERRUPT MASK REGISTER

REGISTERS:

A: BYTE TO BE RETURNED

ALL REGISTERS PRESERVED

```

PI$RIM EQU $
MVI A,PICADR ;PICK UP PIC DEVICE NUMBER
OUT HA893L ;LATCH DEVICE
IN HA893D+1 ;READ 8259
RET ;RETURN TO CALLER

```

PI\$POL - POLL 8259

REGISTERS:

A: BYTE TO BE RETURNED

ALL REGISTERS PRESERVED

```

@I$POL EQU $
MVI A,PICADR ;PICK UP PIC DEVICE NUMBER
OUT HA893L ;LATCH DEVICE
MVI A,00001100B ;BIT PATTERN FOR OCW3 POLL REQUEST
CALL PI$OW3 ;ISSUE OCW3
CALL PI$RIR ;READ THE RESULTS OF THE POLL
RET ;RETURN TO CALLER

```

PI\$IPL - INITIALIZE 8259 FOR POLLING. THIS ROUTINE INITIALIZES THE 8259 SO THAT ALL INTERRUPTS ARE MASKED. THIS ROUTINE IS DESIGNED SO THAT A CALL TO PI\$RIR CAN BE ISSUED TO CHECK FOR DEVICES REQUESTING SERVICE.

ALL REGISTERS PRESERVED

```

PI$IPL EQU $
PUSH PSW ;SAVE PSW
MVI A,00011010B ;SET 8259 IN LEVEL MODE.
CALL PI$IW1 ;WRITE ICW1
XRA A ;VALUE FOR ICW2
CALL PI$IW2 ;WRITE ICW2
CMA ;ALL BITS ON TO MASK ALL INTERRUPTS
CALL PI$OW1 ;WRITE OCW1 TO MASK ALL INTERRUPTS
POP PSW ;RESTORE PSW
RET ;RETURN TO CALLER

```

'A/D DEFINITIONS'

A/D DEFINITIONS

ADCADR EQU 4 ;A/D CONVERTER IS DEVICE 4

'A/D I/O ROUTINES'

A/D I/O ROUTINE

AD\$RD - READ A/D CHANNEL

NOTE - THIS ROUTINE REQUIRES THAT THE 8259 BE INITIALIZED
SO THAT PI\$RIR CAN BE USED TO POLL THE DEVICES
REQUESTING AN INTERRUPT. THE ROUTINE PI\$IPL WILL
INITIALIZE THE 8259 IN THIS MODE.

REGISTERS:

L: A/D CHANNEL TO BE READ
A: VALUE FROM A/D CHANNEL (L)

REGISTER A DESTROYED

```
AD$RD EQU $
MVI A,ADCADR ;A/D CONVERTER DEVICE NUMBER
OUT HAB93L ;LATCH DEVICE NUMBER
MOV A,L ;GET CHANNEL TO BE READ
OUT HAB93D ;LATCH CHANNEL TO BE READ
OUT HAB93D+1 ;START CONVERSION

;
; WAIT FOR A/D TO FINISH CONVERSION
;
AD$LP1 EQU $
CALL PI$RIR ;READ 8259 INTERRUPT REQUEST REGISTER
ANI 00010000B ;IGNORE ALL REQUEST EXCEPT A/D REQUEST
JZ AD$LP1 ;LOOP TILL A/D CONVERSION COMPLETE

;
; SELECT A/D AND READ BACK CONVERTED VALUE
;
MVI A,ADCADR ;A/D CONVERTER DEVICE NUMBER
OUT HAB93L ;LATCH DEVICE NUMBER
IN HAB93D ;READ VALUE FROM A/D
RET ;RETURN TO CALLER

; INITIALIZE VP REGISTERS
;
PATTERN: EQU 0000H
PATTERNNAME: EQU 1800H
SPRITENAME: EQU 1B00H
PATTERNCOLOR: EQU 2000H
SPRITEPATTERN: EQU 3800H
```

PLOT:

```
LXI H,VP$NEV+VP$G2M+VP$1GK+VP$EDP+VP$DI+VP$PM+VP$SO+VP$MO
CALL VP$SOP ;SET OPTION REGISTER

LXI H,PATTERNNAME
CALL VP$SPN

LXI H,PATTERNCOLOR
CALL VP$SCG

LXI H,PATTERN
CALL VP$SPG

LXI H,SPRITENAME
CALL VP$SSN
```

LXI H,SPRITEPATTERN
CALL VP\$SSG

MVI A,UC\$LGR
CALL VP\$STB

INITNAMES:

LHLD VP\$PNT
CALL VP\$SWA
MVI B,0

MN1:

XRA A

MN2:

CALL VP\$WRV
INR A
JNZ MN2

INR B
MVI A,3
CMP B
JNZ MN1

INITSPRITE:

LHLD VP\$SNT
LXI D,SPRITE0
LXI B,32*4
CALL VP\$MTV

LHLD VP\$SGT
LXI D,SPRITETABLE
LXI B,SPRITELENGTH
CALL VP\$MTV

INITCOLOR:

LHLD VP\$CGT
CALL VP\$SWA
LXI D,6144

IC1:

MVI A,UC\$WHT*UC\$LFT+UC\$BLK
CALL VP\$WRV
DCX D
MOV A,D
ORA E
JNZ IC1

LXI H,BITARRAY
CALL INITPATTERN

LHLD VP\$PGT
LXI D,BITARRAY
CALL MOVE6K

MVI A,PS\$PBI+PS\$PAI ;ENABLE BOTH PORTS FOR INPUT
CALL PS\$WER ;WRITE ENABLE BYTE

CALL PI\$IPL

LOOP:

CALL JOY1
CALL PS\$RPA ;READ SWITCH

JMP LOOP

```

        JMP      0000H

INITPATTERN:
        PUSH    H          ;SAVE STARTING ADDRESS
        LXI    D,6144

IP1:
        MVI    A,0
        MOV    M,A
        DCX    D
        INX    H
        MOV    B,A
        MOV    A,D
        ORA    E
        JNZ    IP1

        POP    D          ;RETRIEVE STARTING ADDRESS
        PUSH    D

        LXI    H,15
        DAD    D
        LXI    D,32
MVI     A,1
        MVI    B,192

IP3:
        MOV    M,A
        DAD    D          ;H+32
        DCR    B
        JNZ    IP3       ;LOOP 192 TIMES

        POP    D
        LXI    H,3072
        DAD    D
        MVI    A,0FFH
        MVI    B,32

IP2:
        MOV    M,A
        INX    H
        DCR    B
        JNZ    IP2
        RET

;
;
;
MOVE6K:
        CALL   VP$SWA
        LXI    B,0        ;B=0,C=0
        LXI    H,0        ;H=0

MGK1:
        PUSH    H
        DAD    D
        MOV    A,M        ;A=DATA @POINTER
        CALL   VP$WRV     ;WRITE
        POP    H
        MOV    A,L
        ADI    32
        MOV    L,A
        CMP    C
        JNZ    MGK1

```



```

INR    C
INR    L
MVI    A,32
CMP    C      ;C=32?
JNZ    MGK1
INR    H      ;HL + 256
MVI    L,0
MVI    C,0
MVI    A,24
CMP    H
JNZ    MGK1
RET

```

JOY1:

```

MVI    L,0      ;A/D CHANNEL 0
CALL   AD$RD    ;READ FROM A/D
MOV    D,A      ;SAVE X VALUE
INR    L        ;A/D CHANNEL 1
CALL   AD$RD    ;READ A/D
SBI    31       ;CORRECT FOR BLEED-IN
MOV    E,A      ;SAVE Y VALUE
XCHG                      ;MOVE X,Y TO HL
MVI    A,0      ;SPRITE 0
CALL   UP$SSC   ;SET SPRITE COORDINATES
RET

```

;
;
;
;

SPRITETABLE:

```

DB      18H,24H,42H,42H,42H,42H,24H,18H ; ZERO
DB      08H,18H,08H,08H,08H,08H,08H,1CH ; ONE
DB      1CH,22H,02H,02H,04H,18H,20H,7EH ; TWO
DB      38H,44H,04H,18H,04H,04H,44H,38H ; THREE
DB      04H,0CH,14H,24H,44H,7EH,04H,04H ; FOUR
DB      7EH,40H,40H,7CH,02H,02H,44H,38H ; FIVE
DB      14H,24H,40H,40H,5CH,62H,22H,1CH ; SIX
DB      7EH,42H,02H,04H,08H,10H,20H,20H ; SEVEN
DB      18H,24H,24H,18H,24H,42H,42H,3CH ; EIGHT
DB      18H,24H,42H,4CH,4AH,32H,04H,78H ; NINE
DB      00H,18H,18H,7EH,7EH,18H,18H,00H ; PLUS
DB      00H,00H,00H,3EH,3EH,00H,00H,00H ; MINUS
DB      00H,00H,00H,00H,00H,00H,18H,18H ; DECIMAL
DB      0C3H,0E7H,7EH,3CH,3CH,7EH,0E7H,0C3H ; X
DB      0C3H,0E7H,7EH,3CH,18H,18H,18H,18H ; Y
DB      10H,10H,10H,0CCH,10H,10H,10H,0H ; CURSOR
DB      00H,00H,00H,00H,00H,00H,00H,00H ; SPARE
DB      00H,00H,00H,00H,00H,00H,00H,00H ; SPARE
DB      00H,00H,00H,00H,00H,00H,00H,00H ; SPARE
DB      00H,00H,00H,00H,00H,00H,00H,00H ; SPARE

```

SPRITELENGTH: EQU \$-SPRITETABLE

;
;
;
;

SPRITE0:

```
DB      50,50,0FH,VC$LRD
```

SPRITE1:

```
DB      120,0,0BH,VC$LBL      ;LEFT SIGN
```

SPRITE2:

```

        DB          120,8,0DH,VC$LBL          ;LEFT X
SPRITE3:
        DB          120,235,0AH,VC$LBL        ;RIGHT SIGN
SPRITE4:
        DB          120,243,0DH,VC$LBL        ;RIGHT X
SPRITE5:
        DB          0DOH,0,0,0
SPRITE6:
        DB          0DOH,0,0,0
SPRITE7:
        DB          0DOH,0,0,0
SPRITE8:
        DB          0DOH,0,0,0
SPRITE9:
        DB          0DOH,0,0,0
SPRITE10:
        DB          0DOH,0,0,0
SPRITE11:
        DB          0DOH,0,0,0
SPRITE12:
        DB          0DOH,0,0,0
SPRITE13:
        DB          0DOH,0,0,0
SPRITE14:
        DB          0DOH,0,0,0
SPRITE15:
        DB          0DOH,0,0,0
SPRITE16:
        DB          0DOH,0,0,0
SPRITE17:
        DB          0DOH,0,0,0
SPRITE18:
        DB          0DOH,0,0,0
SPRITE19:
        DB          0DOH,0,0,0
SPRITE20:
        DB          0DOH,0,0,0
SPRITE21:
        DB          0DOH,0,0,0
SPRITE22:
        DB          0DOH,0,0,0
SPRITE23:
        DB          0DOH,0,0,0
SPRITE24:
        DB          0DOH,0,0,0
SPRITE25:
        DB          0DOH,0,0,0
SPRITE26:
        DB          0DOH,0,0,0
SPRITE27:
        DB          0DOH,0,0,0
SPRITE28:
        DB          0DOH,0,0,0
SPRITE29:
        DB          0DOH,0,0,0
SPRITE30:
        DB          0DOH,0,0,0
SPRITE31:
        DB          0DOH,0,0,0
;
;
;
BITARRAY:

```

DS 6144

END PLOT