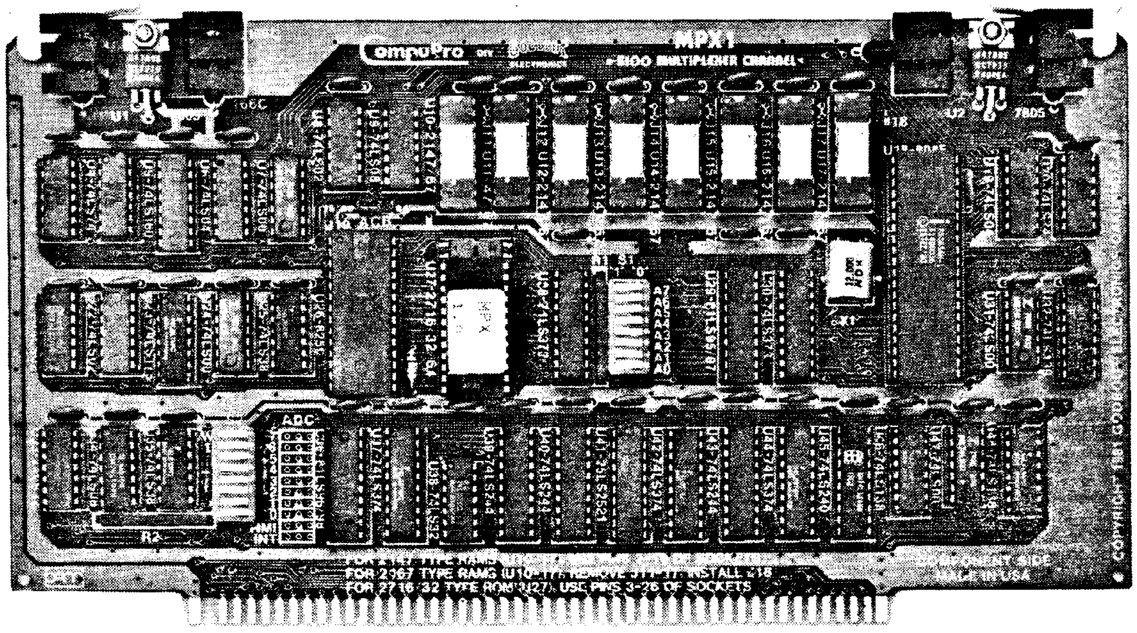
MPX-1™
TECHNICAL MANUAL



IEEE 696 / S-100

MULTIPLEXER CHANNEL
4K or 16K RAM
6 MHz 8085 Processor

166C
mac

CompuPro®

BULLDOG COMPUTER
IBM PC - XT - AT COMPUPRO
1334 Chapel Street
New Haven, CT 06511
(203) 777-1476 or -7763

2

TABLE OF CONTENTS

This document was proofread with the aid of SpellGuard™ from ISA, Menlo Park, CA.

FIRST - A WORD OR TWO OF WARNING!

This manual is intended to guide the sophisticated systems integrator or OEM through the hardware features of the MPX-1. **This manual is not intended for novice or inexperienced users.** If you are an end-user who has purchased an MPX-1 all by itself, you should have a thorough knowledge of hardware and software as **CompuPro** or your local dealer is not prepared to provide applications assistance for this product, (beyond the contents of this manual). If you are an end-user who has purchased a system with an MPX-1 integrated into it, your systems integrator should have done all the work for you. This means that they should have provided the operating system software with the MPX-1 integrated into it already. If this is the case, feel free to read this manual for your information, but you should never have to worry about this manual's contents.

If you don't feel that you are sophisticated enough to handle programming the MPX-1 (be honest with yourself now!), please return it to the place of purchase for a full refund.

ABOUT THE MPX-1

The MPX-1 from **CompuPro** is a very useful addition to any multi-user or interrupt intensive environment. Its extreme versatility and flexibility makes it a very bewildering product on the surface - it can do so much, where do you start? The basic function of the MPX-1 is that of an interrupt pre-processor. It takes the interrupt servicing and handling load off of the main processor in a system. Interrupts in the system are now processed in parallel with other processing resulting in higher system throughput, which is a paramount consideration in multi-user, multi-tasking situations.

The MPX-1 was designed for the IEEE 696/S-100 Bus because of that bus' modularity and its unique ability to handle multiple temporary bus masters (of which the MPX-1 is one). The IEEE 696/S-100 bus is the choice of professionals for business, industrial and scientific applications.

TECHNICAL OVERVIEW

The MPX-1 has its own on-board processor, an Intel 8085AH-1 running at 6 MHz. This processor is supported by either 4K or 16K of fast static RAM and up to 8K of EPROM. In addition, a flexible 8259A interrupt controller monitors the eight vectored interrupt lines on the S-100 Bus. None of these local resources take up any address space on the S-100 Bus.

When an interrupt request occurs on one of the vectored interrupt lines, the interrupt response is handled by the on-board interrupt controller and CPU, taking up no processing time on the bus.

When needed, the MPX-1 can access any system resource on the bus. This means that it can talk to any I/O port or memory location on the bus. It does this by requesting the bus from the permanent master on a cycle-by-cycle basis. This request/grant procedure is fully arbitrated to 16 levels as provided for by the IEEE 696/S-100 bus standard. This allows up to 16 such devices to exist in a system at any given time. Multiple MPX-1s could even be used.

The uses of the MPX-1 are vast. Obvious uses would include terminal interrupt handlers, print spoolers, task allocation and management, and resource control. A unique feature of the MPX-1 allows it to load execution code from the system memory so that the MPX-1's function can be altered dynamically.

HOW TO USE THE MPX-1

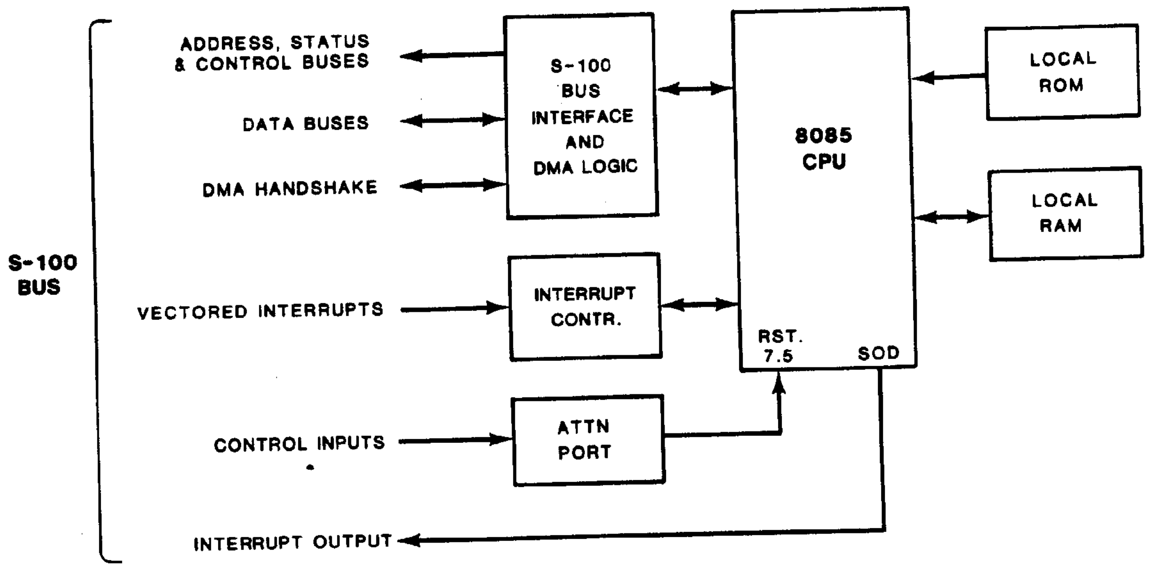
A simplified block diagram of the MPX-1 appears in Figure 1. In the following discussions, it may be useful to refer to the block diagram.

Here is a quick-and-dirty overview of how the MPX-1 works: When the on-board 8085 powers up, it begins executing code from its on-board EPROM. Usually it will be sitting in a loop waiting for something to happen, an external interrupt for example. Suppose an interrupt occurs. The 8085 will be interrupted and vectored to a service routine by the 8259A interrupt controller. The MPX-1 can access any I/O port on the S-100 Bus by simply doing an INPUT or OUTPUT instruction. Since the "ports" on the MPX-1 are all memory mapped, no conflicts occur and all 256 ports can be accessed. The on-board logic takes care of the DMA request, arbitration and the actual running of the bus cycle. The 8085 will "stall" until the operation is completed allowing program execution to continue.

The MPX-1 can also access any memory location on the S-100 Bus. To do this, the program first writes the upper 16 address bits (A8-23) to two registers. Then if a memory reference to address C000 to F000 is made, the lower 8 bits from the reference are used to make up the lower 8 bits of the desired memory location on the bus. If the memory reference is a read operation, then the bus access will also be a read operation. If the memory reference is a write operation, then the bus access will be a write operation. This address space from C000 to F000 is known as the "external window". As in the case of I/O accesses, the on-board logic takes care of the DMA request, arbitration and the actual running of the bus cycle. The 8085 will "stall" until the operation is completed allowing program operation to continue.

Sounds simple, doesn't it? Well, it is. The following sections discuss the above overview in greater detail. First is a local address map showing where everything lives on the MPX-1.

Figure 1. MPX-1 BLOCK DIAGRAM



LOCAL ADDRESS MAP

FROM	TO	SIZE	FUNCTION	NOTES
0000	3FFF	4-16K	RAM	1,4
4000	5FFF	2-8K	EPROM	1,2,5
8000	8001	2	8259A Registers	
8002		1	Set Interrupts Latch	2
8004		1	DMA Address Bits 8-15	3
8005		1	DMA Address Bits 16-23	3
8007		1	Interrupt Response Byte	3
C000	FFFF	16K	External Window	6
All I/O Ports		256	External Window	6

Notes:
1. RAM and EPROM exchange locations for power-on-jump.
2. Read Only.
3. Write Only.
4. 4K or 16K depending on chips used.
5. 2K for 2716, 4K for 2732, 8K for 2764.
6. Lower 8 address bits (A0-7) pass through.

ACCESSING MEMORY ON THE BUS

An access to memory that resides on the S-100 Bus will be performed if the CPU makes an access to the range noted as the external window in the address map above (addresses C000 to FFFF). The low order 8 bits of the address (A0-7) will be taken from the CPU's address lines directly, while the high order 16 bits (A8-23) will come from the address that has been previously written to the DMA address registers. So the procedure for accessing memory on the bus would be:

1. Write desired A8-15 bits to memory location 8004.

2. Write desired A16-23 bits to memory location 8005.

3. Perform a memory reference to C0XX to F0XX, where XX represents the low order 8 bits (A0-7) of the desired address.

If the memory reference to the external window is a read operation, then a memory read will occur on the bus. If the memory reference is write operation, data will be written to the memory on the bus. Note that M1 cycles may also be executed on the bus meaning that the MPX-1 may actually execute a small amount of code (less than 256 bytes) directly from the bus.

Note that the high order byte of the external window is insignificant. Any address in the range C000 to FFFF will be treated identically. For example, C083, BD83, A983, and FF83 would all access the same external memory location - XXXX83, where XXXX are the two bytes from the DMA address registers.

ACCESSING I/O PORTS ON THE BUS

All of the "I/O Ports" local to the MPX-1 are "memory mapped". This means they are decoded in the memory address space rather than in the I/O space. This leaves all 256 I/O addresses free. Any input or output cycle performed by the CPU will cause a corresponding cycle to be executed on the S-100 Bus. Since the lower 8 bits of the DMA Address pass through from the actual lower 8 address bits from the CPU, the port address specified in the I/O instruction will be the one accessed on the bus.

The high order 8 bits of the I/O access will come from the DMA address register as in a memory reference. This allows the port address to be "mirrored" in A8-15 as early S-100 (8080) processors did, or this byte may be loaded with different data to emulate Z-80 I/O modes (the Z-80 passes the accumulator contents on A8-15). This also allows the MPX-1 to emulate the current generation of 16 bit processors such as the CPU 8085/88, CPU 86/87 and the CPU 68K, which can put out 16 bit I/O addresses.

If an input instruction is executed, then an input cycle will be performed on the S-100 Bus. If an output instruction is executed, then an output cycle will be performed on the bus.

GETTING THE MPX-1's ATTENTION

In any system it will be necessary for the main CPU in the system to get the attention of the MPX-1. This can be for initial start-up of the MPX-1, or to "interrupt" its current task to be given another. This is done through a mechanism called the ATTN port. This port is on the S-100 Bus and its address is selected by switch S1. When the system CPU executes an output to the ATTN port, a RST 7.5 will be generated to the on-board 8085. Note that no data is accepted by the MPX-1.

GETTING THE MAIN SYSTEM CPU'S ATTENTION

The MPX-1 may need to get the attention of the system CPU to tell it that a task is complete, a buffer is nearing full, or many other reasons. The MPX-1 may signal the main CPU by causing an interrupt on the bus. This interrupt may occur on the INT*, NMI* or any of the vectored interrupt lines. A hardware jumper is used to select which of the ten possible lines are used.

Two methods of causing this interrupt are available, again selected by a jumper. The first type uses the Serial Output Data (SOD) line from the 8085 to cause the interrupt. The state of this line is set and reset by the Set Interrupt Mask (SIM) instruction. The state of the interrupt request must be reset in software.

The second method uses a one bit latch that is set by performing a read from address 8002 (Set Interrupt Latch in the address map above). This latch is automatically reset by the occurrence of an interrupt acknowledge cycle.

Note that if one of the vectored interrupt lines is selected to cause the system interrupt, the corresponding interrupt input to the 8259A should be masked, unless you want the MPX-1 to interrupt itself.

INTERRUPT ACKNOWLEDGE RESPONSE ON THE BUS

The MPX-1 may provide a single byte of data during bus interrupt acknowledge cycles. This response must be enabled by a switch. The data to be passed during interrupt acknowledge cycles is written to the latch at address 8007 (Interrupt Response Byte in the address map above).

Note that this single byte response may cause an 8080/Z-80 RESTART instruction, or is compatible with the vector information required by 8088/86 or 68000 CPUs. Note that if the response is desired and the CPU is an 8088/86 or 68000, the SOD interrupt call method (described above) should be used. This is because the 8086 and 68000 run two interrupt acknowledge cycles (the first byte of data is ignored) and the interrupt latch used in the second method would be reset prematurely. This may cause a system problem.

HARDWARE SWITCH SETTINGS AND JUMPER OPTIONS

SWITCH SETTINGS

There are two dip-switches on the MPX-1. Switch S1 selects the address of the ATTN port on the S-100 bus. Switch S2 is used to select the various board options and the DMA arbitration address of the MPX-1.

S1 - ATTN PORT ADDRESS SELECT

PADDLE #	ADDRESS BIT	
1	A7	
2	A6	
3	A5	"ON" = "0"
4	A4	
5	A3	
6	A2	"OFF" = "1"
7	A1	
8	A0	

STANDARD ATTN PORT ADDRESS SELECTION

The **CompuPro** "standard" port address for MPX ATTN calls is F1 hex. To set the MPX-1 to respond to ATTN calls on port F1 hex, set S1 as follows: Paddles 1-4 and paddle 8 should be OFF. Paddles 5-7 should ON.

S2

PADDLE #	FUNCTION	
1	"ON" enables EPROM wait state.	
2	"ON" enables interrupt response byte.	
3	DMA priority address 3	
4	DMA priority address 2	"ON = "0"
5	DMA priority address 1	
6	DMA priority address 0	"OFF" = "1"
7	"ON" enables SLAVE CLR* to reset MPX-1.	
8	not used	

INTERRUPT "CALL" JUMPERS

The MPX-1 "calls" the system CPU by causing an interrupt on the bus. There are ten possible interrupt lines that the MPX-1 may assert. They are: INT*, NMI* or any of the eight vectored interrupt lines (VI0*-VI7*). There are also two methods by which the MPX-1 can assert the interrupt request - the SOD line or by setting a hardware latch (described above in the section entitled "Getting the Main System CPU's Attention").

The interrupt line asserted and the method of asserting it are selected by Jumpers J1-10. These jumpers are implemented with push-on shorting plugs and pins soldered into the board. Each jumper has three pins labeled A, B and C. The "A" pin of each jumper is connected to the SOD interrupt source. The "C" pin of each jumper is connected to the interrupt response latch. The "B" pin of each jumper is connected to an interrupt line on the bus according to the chart below:

JUMPER #	LEGEND MARKING	"B" POSITION CONNECTION
1	7	VI7*
2	6	VI6*
3	5	VI5*
4	4	VI4*
5	3	VI3*
6	2	VI2*
7	1	VI1*
8	0	VI0*
9	NMI	NMI*
10	INT	INT*

EXAMPLE: To connect the MPX-1 interrupt request output to the INT* line on the bus with the interrupt source from the SOD line, a shorting plug should be installed at J10 from the "A" to "B" position (left of center).

EXAMPLE: To connect the MPX-1 interrupt request output to the VI3* line on the bus with the interrupt source from the interrupt latch, a shorting plug should be installed at J5 from the "B" to "C" position (right of center).

NOTE: The software supplied with the MPX-1 assumes the use of the SOD interrupt mode, so if you wish to use the MPX in an interrupt driven mode with the standard software, use only the SOD interrupt source.

SELECTING AND USING 4K OR 16K RAM CHIPS

The MPX-1 can use either 2147 4Kx1 RAM chips, or 2167 16Kx1 RAM chips. The MPX-1 should have come from the factory already jumpered correctly for the type of chip that was originally ordered with the board. Should it become necessary to change these jumpers once the board is in the field, here is how different RAMs are jumpered and inserted:

USING 2147 4Kx1 TYPE RAM CHIPS

To use 2147 type RAM chips, jumpers J11 through J17 should be installed and J18 should be open. J11 through J17 are located in-between the RAM array (U10-

17), and J18 is located at the right-hand side of the RAM array.. The RAM chips come in 18 pin packs, but the sockets are 20 pin to accomodate 2167 type RAMs. When using 2147 type RAMs, plug them in so that the chips are in the bottom-most part of the socket, that is pins 1 and 20 are blank.

USING 2167 16Kx1 TYPE RAM CHIPS

To use 2167 type RAM chips, jumper J18 should be installed and jumpers J11-17 should be open. Jumper J18 is located at the far right-hand side of the RAM array and jumpers J11-17 are located in-between the RAM chips.

CONFIGURING THE MPX-1 FOR DIFFERENT SIZE EPROMS

The MPX-1 has a JEDEC 28 pin socket for the EPROM (U27). With the use of one jumper, this socket can accomodate a 2716, 2732 or a 2764 type EPROM. This gives 2K, 4K or 8K of storage, respectively.

USING A 2716 or 2732 EPROM

To use a 2716 or 2732 type EPROM, jumper J19 should have a shorting plug installed connecting pins "A" and "C" (left of center). J19 is located just above U26. The 2716 or 2732 should be installed at location U27 such that it uses the bottom-most pins of the socket, that is pins 1,2,27 and 28 are left blank.

USING A 2764 EPROM

To use a 2764 type EPROM, jumper J19 should have a shorting plug installed connecting pins "C" and "B" (right of center). J19 is located just above U26. The 2764 should be installed at location U27, and all the pins of the socket are used.

STANDARD SOFTWARE SUPPLIED WITH MPX-1

The MPX-1 is supplied with an EPROM that contains some general purpose utility routines. It contains code to initialize the interrupt controllers (to a benign state), several useful subroutines and a general purpose command interpreter that implements a "channel protocol". Included are several built-in commands to perform useful tasks such as loading and executing programs from system memory, changing the interrupt controller parameters and block memory moves on system RAM. The command structure includes a sophisticated "link" protocol that allows chaining of command sequences and recursion.

Note that no representation is made that this is the most efficient way to program or use an MPX board. Rather, it is intended as partly tutorial and partly a useful way to get "up and running" with the MPX in a minimum amount of time.

What follows is a discussion of the basic command structure and then descriptions of the actual commands. Following that is a discussion of the code itself that explains how to add custom commands and describes several useful subroutines.

BASIC COMMAND STRUCTURE AND PROTOCOL

When the MPX-1 powers up, it masks all its interrupt inputs, does some internal initialization and waits quietly for an ATTN on its ATTN port. When it receives an ATTN it will read in 16 bytes from the system memory starting at address 50 hex. The meaning of the bytes follows:

Byte 0:	Opcode Byte
Byte 1:	Status Indication Byte
Byte 2:	General Purpose Parameter Byte 0
•	
•	
•	
Byte 10:	General Purpose Parameter Byte 8
Byte 11:	Link Address (least significant byte)
Byte 12:	Link Address
Byte 13:	Link Address (most significant byte)
Byte 14:	Result 1 byte
Byte 15:	Result 2 byte

The following is a more detailed description of the bytes shown above:

OPCODE BYTE

The opcode byte contains the information that tells the MPX what command to execute, and also contains two bits that control the completion interrupt and link structures. The actual bit coding of the opcode byte is shown below:

```
 Bit 7                                                    Bit 0
 ---------------------------------------------------------------
| CONT |  INT |   0   | BIT 4 | BIT 3 | BIT 2 | BIT 1 | BIT 0 |
 ---------------------------------------------------------------
```

Bits 0 through 4 of the opcode contain the "command number" of which there are a possible 31 (one command is reserved). The first 10 commands have already been defined and the remaining 21 may be implemented by you.

Command "OF" is reserved, and if invoked will be translated internally to a NOP command.

The INT bit (bit 6) of the opcode byte is the INTERRUPT ENABLE bit. If this bit is set to one in the opcode, the MPX will cause an interrupt (SOD interrupt) when it has completed the command. If this bit zero, an interrupt will not be generated when the command has completed execution. Note that in order for the interrupt to make it to the bus, it must be jumpered to an interrupt line (see the hardware configuration section of this document for further details).

The CONT bit (bit 7) of the opcode byte is the CONTINUE bit and is associated with the link structure. If the CONT bit is set high, execution of the next command (pointed to by the link address) will commence immediately after execution of the current command is complete. If the CONT bit is zero, the MPX will stop when execution of the current command is complete, and wait for another ATTN before executing the next command.

There are two considerations when using the CONT bit. The first is that the INT bit is ignored if the CONT bit is set. This means that an interrupt will only occur if no command is to follow, which is the way one would normally want things to happen when executing a sequence of commands.

Secondly, if the link address points to the beginning of this same instruction (pointing to itself) and the CONT bit is set, endless execution of the instruction will occur. This could be useful. The way to stop execution would be to change the opcode so that the CONT bit is zero. The opcode could be changed to a NOP, or merely the same opcode with the CONT bit zeroed. Either the system CPU or the MPX could change the opcode. DO NOT try to change the link address on the fly!

Bit 5 is unused and is really a "don't care" bit, but a good practice would be to always set this bit to zero.

STATUS INDICATION BYTE

This byte is used to "handshake" with the MPX when the use of a completion interrupt is not desired, or they may be used in conjunction. This byte should be set to zero before an ATTN is sent to the MPX. When the MPX is done executing the command, it will set this byte to FF hex. In a non-interrupt environment, this byte should be checked before another command is sent to the MPX.

If the CONT bit is set in the command opcode, the status byte for that particular command will NOT be set to FF hex. This means that in a sequence of commands, only the status byte in the last command (the one with CONT = 0) will be set to FF hex.

GENERAL PURPOSE PARAMETER BYTES 0 through 8

These bytes are used to send parameters to the MPX along with the command. The parameter(s) sent vary with the command. Only the block move commands use all nine bytes, and they would contain the starting, ending and destination addresses for the block move (3 bytes each). Sometimes only a few bytes are used and sometimes none are used at all. The exact usage of these bytes is detailed in each individual command description.

LINK ADDRESS BYTES

These three bytes are a pointer to the place at which the next command line is resident in the system memory. When not executing multiple commands (CONT bit = 0), this address would normally point to the beginning of the same command. When executing a sequence of commands, this address would point to the address of the next command.

The address is stored low byte first and is a full 24 bit address.

The initial link address is 50 hex, but the NOP instruction may be used to change the link address to any other system address.

Note that the link address is read only once and at the start of each command, not at the end. This means that the command itself may modify the link address, but it will only affect the following command (not the where the next command will be fetched). The main system CPU should not modify the current link address unless the MPX is not active.

RESULT 1 AND RESULT 2 BYTES

Sometimes it is desirable to have the MPX return parameters to the caller, and that is the purpose of these two bytes. Only two of the built-in commands return data to these locations, but user generated commands should use these bytes for that purpose as well.

GENERAL NOTES

Commands are assumed to be resident on 16 byte boundaries ie: 50H, 60H, 180H, etc.

The only bytes in the command line that the MPX modifies are the status indication and result bytes. All others are left intact.

COMMAND DESCRIPTIONS

NOP - No Operation

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	0	0	0	0

PARAMETERS PASSED: Link Address.
PARAMETERS REURNED: None.

DESCRIPTION: This command seems useless on the surface, but in reality has many uses. This command may be used to change the link address if address 50 hex is not a good one for your system. It may also be used to reset the interrupt output from the MPX if it was set by the completion of a previous command (of course the INT bit should be zero). This command is also useful in debugging a command sequence since it may be used to cause execution to skip the command that is replaced with a NOP.

RESET - Reset the MPX-1

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	0	0	0	1

PARAMETERS PASSED: None.
PARAMETERS RETURNED: None.

DESCRIPTION: This command resets the MPX-1 to its initial starting state. The internal command table will be cleared (so any custom commands you have loaded into RAM will now be ignored). The address where the MPX picks up its first command line will be set to 50H. All interrupts will be masked and the interrupt controller will be re-initialized.

SET MASK - Mask or Unmask interrupt inputs to MPX-1

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	0	0	1	0

PARAMETERS PASSED: Mask Byte, Link Address.
PARAMETERS RETURNED: None.

DESCRIPTION: This command is used to mask or unmask interrupt inputs to the MPX-1. The byte passed in Parameter Byte 0 is written to the mask register of the 8259A interrupt controller. If a bit in the mask byte is set to one, the corresponding interrupt will be masked. Conversely, if a bit is zero, that interrupt will be unmasked. Bit 0 of the mask byte corresponds to VI0* on the bus, and Bit 7 corresponds to VI7* on the bus. This is the same as sending OCW1 to the interrupt controller (see the 8259A application note in the appendix of this document for more information).

SENDEOI - SEND END-OF-INTERRUPT COMMAND TO INTERRUPT CONTROLLER

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	0	0	1	1

PARAMETERS PASSED: EOI Command Byte, Link Address.
PARAMETERS RETURNED: None.

DESCRIPTION: This command is used to send an End-of-Interrupt Command to the 8259A interrupt controller. It is also useful for rotating the interrupt priority levels. The byte to be sent to the 8259A is passed in Parameter Byte 0. This is equivalent to sending OCW2 to the 8259A. For more information on what this byte does to the 8259A, refer to the 8259A application note contained in the appendix of this document.

READREG - READ INTERRUPT CONTROLLER REGISTERS

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	0	1	0	0

PARAMETERS PASSED: Link Address.
PARAMETERS RETURNED: Contents of IS and IR registers in 8259A.

DESCRIPTION: This command is used to read the contents of the Interrupt Request (IR) and In Service (IS) registers in the 8259A. It returns the contents of the IR register in the Result 1 Byte location and the contents of the IS register in the Result 2 Byte location. For more information on the meaning of the IS and IR registers, see the 8259A application note in the appendix of this document.

SETRESPONSE - SET INTERRUPT RESPONSE BYTE

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	0	1	0	1

PARAMETERS PASSED: Response Byte, Link Address.
PARAMETERS RETURNED: None.

DESCRIPTION: The MPX-1 is capable of putting an 8 bit value on the system data bus during system interrupt acknowledge cycles. The value is called the interrupt response byte and may be set as desired with this command. Note that this response will only appear on the bus if this feature in enabled by a hardware switch. See the hardware section of this document for more information.

SIZE - INDICATE WHETHER 4K OR 16K MPX-1

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	0	1	1	0

PARAMETERS PASSED: Link Address.
PARAMETERS RETURNED: Size indication.

DESCRIPTION: This command is used by the system to determine the amount of memory installed in this particular MPX-1. The Result 1 Byte is set to 00 hex if this is a 4K MPX-1 and is set to FFH if this is a 16K MPX-1.

LOADRAM - LOAD MPX LOCAL RAM FROM SYSTEM RAM

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	0	1	1	1

PARAMETERS PASSED: Starting Address in System RAM (3 bytes)
Ending Address in System RAM (3 bytes)
Destination Address in Local RAM (2 bytes)
Link Address

PARAMETERS RETURNED: None.

DESCRIPTION: This command is used to load the MPX local RAM from system RAM. Three addresses are passed to the MPX: The starting address of the block to be loaded, its ending address, and the starting address of the place to put it in local RAM. The starting and ending addresses are 3 bytes long and the destination address is only two bytes long. All addresses are stored low byte first and the starting address is at Parameter Bytes 0-2, the ending address is at Parameter Bytes 3-5, and the destination address is at bytes 6-7.

EXRAM - EXECUTE A PROGRAM IN LOCAL RAM

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	1	0	0	0

PARAMETERS PASSED: Execution Address (2 bytes), Link Address.

PARAMETERS RETURNED: None.

DESCRIPTION: This command is used to cause a routine stored in local RAM on the MPX-1 to be executed. A "call" is made to the routine, so when it has finished executing a RETURN instruction will pass control back to the interpreter (assuming a clean stack). The starting address of the routine is two bytes long and is stored low byte first at Parameter Bytes 0 and 1.

BLKMOV - MOVE A BLOCK OF RAM ON THE SYSTEM BUS

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	1	0	0	1

PARAMETERS PASSED: Starting Address in System RAM (3 bytes).
Ending Address in System RAM (3 bytes).
Destination Address in System RAM (3 bytes).
Link Address.

PARAMETERS RETURNED: None.

DESCRIPTION: This command is used to move a block of RAM (of any size) on the S-100 bus. Three addresses are passed: The starting address of the block to be moved, its ending address and the destination address of where it's being moved to. Each address is three bytes long and is stored low byte first. The starting address should be at Parameter Bytes 0-2, the ending address should be at Parameter Bytes 3-5 and the destination address should be at Parameter Bytes 6-8. Note that this command is useful for putting code in extended pages that normal 8 bit processors cannot talk to without a lot of effort.

FASTMOV - MOVE A BLOCK OF RAM ON THE SYSTEM BUS - FAST

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	1	0	1	0

PARAMETERS PASSED: Starting Address in System RAM (3 bytes).
Ending Address in System RAM (3 bytes).
Destination Address in System RAM (3 bytes).
Link Address.

PARAMETERS RETURNED: None.

DESCRIPTION: This command is almost the same as above, except that all blocks must be a multiple of 256 bytes, and begin on 256 byte boundaries. This allows the transfer to occur at a much higher rate. Three addresses are passed: The starting address of the block to be moved, its ending address and the destination address of where it's being moved to. Each address is three bytes long and is stored low byte first. The starting address should be at Parameter Bytes 0-2, the ending address should be at Parameter Bytes 3-5 and the destination address should be at Parameter Bytes 6-8. Note that even though each address is three bytes long, the low byte is assumed to be 00 hex.

RESERVED - DO NOT USE

OPCODE BIT CODING:

Bit 7							Bit 0
CONT	INT	0	0	1	1	1	1

Note: This opcode is reserved and should not be used by custom command routines. Attempts to call this opcode will be translated internally to NOP opcodes (link address will still be valid, however). The reason this opcode is reserved is because its jump table address is used by the ATTN routine.

ABOUT THE ROM CODE

This is a brief description of what goes on in the standard software routine.

The first thing in the code is a jump to the actual beginning of the code. The reason this is there is because of the way the MPX-1 does its "power-on-jump" sequence. The ROM appears at address 0000H for the first three cycles and then appears at 4000H thereafter.

INITIALIZATION

The first thing that happens is to set up the interrupt controller with all interrupts masked, and the interrupt vector table at address 80H (4 byte interval).

Starting at 0000H in RAM are two tables: the command jump table and the interrupt vector table. The first thing that happens is that all these jump locations are initialized to point to a routine called DUMMY, which is nothing more than a RET instruction. This is where all interrupts and all unused commands go to, (until you change the table). The next thing that happens is to patch the command jump table with the addresses of the ten command routines that are supplied in the ROM. Next the stack pointer is initialized leaving about 77 bytes free for stack usage. That should be more than enough, the routines in the ROM never get more than about 6 bytes deep.

Next the initial link address is set up to point address 50H and then the ATTN interrupt (RST 7.5) is armed and the MPX sits quietly waiting for an ATTN.

ATTN RESPONSE

When an ATTN occurs, first the interrupt output is cleared, the ATTN interrupt input is masked and the last link address is copied into CURRENT. Then 16 bytes are read from the system memory (pointed to by CURRENT) into a buffer called CMNDBUF. The opcode byte is read from the buffer and decoded. A call is then made to the address pointed to by the lower 5 bits of the opcode. The opcode is first checked to see if it is the same as the ATTN jump address. If it is, a NOP command is substituted instead. Control has now been transfered to the actual command routine. When the command is done executing, all it need do is a RET instruction to get back to the main interpreter loop.

When a command is finished, the program branches to CHECK. This routine checks to see if the CONT bit was set in that opcode. If it was, a jump occurs to the ATTN routine and the next command is executed. If the CONT bit is not set, the INT bit is checked. If it is set, the status byte is set to FFH and the SOD interrupt output is set and the MPX then waits for the next ATTN. If the INT bit is not set, the status byte is set to FFH and the MPX waits for the next ATTN.

GENERAL PURPOSE SUBROUTINES

The ROM contains several useful subroutines that may be utilized by your own custom commands. They include functions such as managing the address pointers, storing them in appropriate registers for bus accesses, and reading and writing bytes on the system bus. Their functions are pretty well documented in the assembly listing that follows.

WRITING YOUR OWN CUSTOM COMMANDS

Ten commands are used by the standard ROM and one is reserved, so that leaves a total of 21 command opcodes that are available for your custom usage. Less than 1/4th of the 2K bytes in the ROM are used by the standard routines, so custom commands could be added to the ROM if you have access to an EPROM burner. If not, commands can be assembled to run from the ample free RAM space and loaded using the LDRAM command. The command jump table is kept in RAM (from 0000H to 007FH) so it may also be changed with the LDRAM command. Control may be passed to the new commands by use of the opcode protocol or by the EXRAM command.

Some considerations about writing into the internal RAM: Be careful not to overwrite the buffer, stack or ATTN jump locations (the ATTN jump resides at 3C - 3FH). The program does not use RAM above 200H at any time, so all RAM above this address is free for your use. It is a good idea to issue a SIZE command to determine the amount of RAM available (4K or 16K) before issuing a LDRAM command. This is because the RAM in a 4K MPX-1 "wraps around" and appears in eack 4K block in the 16K RAM space.

LISTING OF STANDARD SOFTWARE SUPPLIED WITH THE MPX-1

```
                 ;MPX PROM VERSION 1.0
                 ;WRITTEN 1-7-82 BY MARK GARETZ
                 ;COPYRIGHT 1982 BY GODBOUT ELECTRONICS

                 ;VERSION 1.1   Multiple command linking bug fixed-llo

                 ;EQUATES

8000 =           INTCLA  EQU     8000H          ;Interrupt Controller
8001 =           INTCLB  EQU     8001H          ;   "       "
8004 =           DMALO   EQU     8004H          ;DMA Addr. Bits A8-15
8005 =           DMAHI   EQU     8005H          ;DMA Addr. Bits A16-23
8007 =           INTRSP  EQU     8007H          ;Interrupt Response Byte Reg.
00C0 =           WINDOW  EQU     0C0H           ;External Window
0000 =           CMNDTBL EQU     0000H          ;Command Jump Table
0080 =           INTTBL  EQU     0080H          ;Interrupt Response Jump Table
00A0 =           CMNDBUF EQU     00A0H          ;Temporary Command Buffer
00A2 =           STRTAD  EQU     CMNDBUF+2      ;Starting Address, 3 bytes
00A5 =           ENDAD   EQU     STRTAD+3       ;Ending Address, 3 bytes
00A8 =           DESTAD  EQU     ENDAD+3        ;Destination Address, 3 bytes
00AB =           LINKAD  EQU     DESTAD+3       ;Link Address, 3 bytes
00AE =           RESULT1 EQU     LINKAD+3       ;Result byte 1
00AF =           RESULT2 EQU     RESULT1+1      ;Result byte 2
00B0 =           CURRENT EQU     00B0H          ;Current Command Address, 3 bytes
00FF =           STACK   EQU     00FFH          ;Stack Space
0100 =           BUFFER  EQU     0100H          ;Fast Block Move Buffer
0030 =           SIM     EQU     30H            ;SIM INSTRUCTION

                 ;Beginning of Code

4000                     ORG     4000H

4000 C33140              JMP     START          ;3 Init Bytes
4003 564552532E          DB      'VERS. 1.0'
400C 434F505952          DB      'COPYRIGHT 1982 BY GODBOUT ELECTRONICS'

4031 3E9E        START:  MVI     A,9EH          ;ICW1
4033 320080              STA     INTCLA
4036 3E00                MVI     A,0            ;ICW2
4038 320180              STA     INTCLB
403B 320180              STA     INTCLB         ;ICW3 too
403E 3EFF                MVI     A,0FFH         ;OCW1 Mask all interrupts
4040 320180              STA     INTCLB         ;Done with interrupt controllers

                 ;The following code initializes the Command Jump Table and
                 ;Interrupt Jump Table.  Unused entries jump to DUMMY
                 ;Note: all entries are first initialized to DUMMY, then
                 ;patched for commands and ATTN.

4043 210000              LXI     H,CMNDTBL
4046 117B42              LXI     D,DUMMY        ;D gets DUMMY addr.
4049 0100C3      LOOP1:  LXI     B,0C300H       ;JUMP and a NOP
404C 7D                  MOV     A,L
```

```
404D FEA0              CPI     0A0H            ;check for end
404F CA5D40            JZ      MORE
4052 70                MOV     M,B             ;the JUMP
4053 23                INX     H
4054 73                MOV     M,E             ;the DUMMY addr.
4055 23                INX     H
4056 72                MOV     M,D
4057 23                INX     H
4058 71                MOV     M,C             ;the NOP
4059 23                INX     H
405A C34940            JMP     LOOP1
405D 21E040    MORE:   LXI     H,ATTN          ;addr. of ATTN routine
4060 223D00            SHLD    003DH

               ;This part of the code writes the addresses of the command
               ;routines into the command table.

4063 21CE41            LXI     H,NOPR          ;address of NOP routine
4066 220100            SHLD    CMNDTBL+1
4069 213140            LXI     H,START         ;RESET jump
406C 220500            SHLD    CMNDTBL+5
406F 21CF41            LXI     H,SETMSK        ;set mask routine
4072 220900            SHLD    CMNDTBL+9
4075 21D641            LXI     H,SETEOI        ;EOI routine
4078 220D00            SHLD    CMNDTBL+13
407B 21DD41            LXI     H,READRG        ;INT regester read routine
407E 221100            SHLD    CMNDTBL+17
4081 21F841            LXI     H,SETRSP        ;set response byte routine
4084 221500            SHLD    CMNDTBL+21
4087 21FF41            LXI     H,SIZE          ;return size routine
408A 221900            SHLD    CMNDTBL+25
408D 213342            LXI     H,LDRAM         ;load RAM routine
4090 221D00            SHLD    CMNDTBL+29
4093 214B42            LXI     H,EXRAM         ;execute RAM routine
4096 222100            SHLD    CMNDTBL+33
4099 214F42            LXI     H,BLKMOV        ;block move routine
409C 222500            SHLD    CMNDTBL+37
409F 216542            LXI     H,FSTMOV        ;fast block move routine
40A2 222900            SHLD    CMNDTBL+41

40A5 31FF00            LXI     SP,STACK
40A8 21AB00            LXI     H,LINKAD
40AB 3650              MVI     M,50H           ;low order initial link addr.
40AD 23                INX     H
40AE AF                XRA     A
40AF 77                MOV     M,A
40B0 23                INX     H
40B1 77                MOV     M,A

               ;This routine arms the attention interrupt and waits.

40B2 3E1B      REARM:  MVI     A,1BH
40B4 30                DB      SIM
40B5 FB                EI
40B6 76                HLT
```

```
                    ;This routine checks the CONT and INT bits of the opcode
                    ;and acts accordingly.

40B7 3AA000         CHECK:  LDA     CMNDBUF
40BA E680                   ANI     80H             ;Check the CONT bit
40BC C2DC40                 JNZ     ATTNO           ;If set then a successive command
40BF 3AA000                 LDA     CMNDBUF         ;get it again
40C2 E640                   ANI     40H             ;Check the INT bit
40C4 CAD340                 JZ      DONE            ;if not set
40C7 CD1041                 CALL    PUTCMD          ;put current addr into regs
40CA 23                     INX     H               ;for status byte addr.
40CB 36FF                   MVI     M,OFFH          ;set status byte in memory
40CD 3ECB                   MVI     A,OCBH          ;set SOD high
40CF 30                     DB      SIM
40D0 C3B240                 JMP     REARM
40D3 CD1041         DONE:   CALL    PUTCMD
40D6 23                     INX     H
40D7 36FF                   MVI     M,OFFH
40D9 C3B240                 JMP     REARM

                    ;This is the main command interpreter routine.  It first copies
                    ;the LINK addr. into CURRENT and then reads in the command
                    ;bytes.  Then an indirect call is executed to the address of
                    ;the command. It also unsets the interrupt output (SOD).

40DC 21B740         ATTNO:  LXI     H,CHECK         ;ADDRESS TO RETURN TO
40DF E5                     PUSH    H               ;PUT ON STACK
40E0 3E5F           ATTN:   MVI     A,5FH           ;reset SOD and mask 7.5
40E2 30                     DB      SIM
40E3 3AAB00                 LDA     LINKAD          ;copy LINKAD into CURRENT
40E6 32B000                 STA     CURRENT
40E9 3AAC00                 LDA     LINKAD+1
40EC 32B100                 STA     CURRENT+1
40EF 3AAD00                 LDA     LINKAD+2
40F2 32B200                 STA     CURRENT+2
40F5 CD4941                 CALL    GETCMD          ;get the command line from memory
40F8 3AA000                 LDA     CMNDBUF         ;get the opcode byte
40FB E61F                   ANI     1FH             ;mask the INT and CONT bits
40FD 110F41                 LXI     D,BACK          ;put return addr. in D
4100 D5                     PUSH    D               ;and put in on the stack
4101 210000                 LXI     H,CMNDTBL       ;HL gets addr. of command table
4104 07                     RLC
4105 07                     RLC                     ;shift the opcode for pointer
4106 FE3C                   CPI     3CH             ;check for reserved op-code
4108 C20D41                 JNZ     ATTN1           ;skip next if OK
410B 3E00                   MVI     A,0             ;otherwise, do a nop command
410D 6F             ATTN1:  MOV     L,A             ;L gets low byte
410E E9                     PCHL                    ;and go there
410F C9             BACK:   RET                     ;we're done

                    ;General Purpose Subroutines Follow:

                    ;This subroutine puts: CURRENT+2 into DMAHI, CURRENT+1
                    ;into DMALO, CURRENT into L reg and WINDOW into H
```

```
4110 3AB200    PUTCMD: LDA     CURRENT+2
4113 320580            STA     DMAHI
4116 3AB100            LDA     CURRENT+1
4119 320480            STA     DMALO
411C 3AB000            LDA     CURRENT
411F 6F                MOV     L,A
4120 26C0              MVI     H,WINDOW
4122 C9                RET

               ;This subroutine puts: STRTAD+2 into DMAHI, STRTAD+1
               ;into DMALO, STRTAD into L reg and WINDOW into H

4123 3AA400    PUTST:  LDA     STRTAD+2
4126 320580            STA     DMAHI
4129 3AA300            LDA     STRTAD+1
412C 320480            STA     DMALO
412F 3AA200            LDA     STRTAD
4132 6F                MOV     L,A
4133 26C0              MVI     H,WINDOW
4135 C9                RET

               ;This subroutine puts: DESTAD+2 into DMAHI, DESTAD+1
               ;into DMALO, DESTAD into L reg and WINDOW into H

4136 3AAA00    PUTDST: LDA     DESTAD+2
4139 320580            STA     DMAHI
413C 3AA900            LDA     DESTAD+1
413F 320480            STA     DMALO
4142 3AA800            LDA     DESTAD
4145 6F                MOV     L,A
4146 26C0              MVI     H,WINDOW
4148 C9                RET

               ;This subroutine gets 16 bytes from system memory pointed to
               ;by CURRENT and puts them into CMNDBUF.

4149 CD1041    GETCMD: CALL    PUTCMD          ;set up the registers
414C 11A000            LXI     D,CMNDBUF
414F 0610              MVI     B,16
4151 7E        GET1:   MOV     A,M             ;get bus byte
4152 12                STAX    D               ;put in buffer
4153 23                INX     H
4154 13                INX     D
4155 05                DCR     B
4156 C25141            JNZ     GET1
4159 C9                RET

               ;This subroutine increments STRTAD (3 bytes) and DESTAD
               ;(3 bytes).  Entry at BUMP2 bumps only the upper two bytes.

415A 3AA200    BUMP:   LDA     STRTAD
415D 3C                INR     A
415E 32A200            STA     STRTAD
4161 3AA800            LDA     DESTAD
4164 3C                INR     A
```

```
4165 32A800             STA     DESTAD
4168 C0                 RNZ
4169 3AA300    BUMP2:   LDA     STRTAD+1
416C 3C                 INR     A
416D 32A300             STA     STRTAD+1
4170 3AA900             LDA     DESTAD+1
4173 3C                 INR     A
4174 32A900             STA     DESTAD+1
4177 C0                 RNZ
4178 3AA400             LDA     STRTAD+2
417B 3C                 INR     A
417C 32A400             STA     STRTAD+2
417F 3AAA00             LDA     DESTAD+2
4182 3C                 INR     A
4183 32AA00             STA     DESTAD+2
4186 C9                 RET

               ;This subroutine has two entry points: CMPAR3 checks all three
               ;bytes of both STRTAD and ENDAD for equality.  CMPAR2 checks
               ;only the upper two bytes.  Returns with Z flag set if equal.

4187 3AA200    CMPAR3:  LDA     STRTAD
418A 47                 MOV     B,A
418B 3AA500             LDA     ENDAD
418E B8                 CMP     B
418F C0                 RNZ
4190 3AA300    CMPAR2:  LDA     STRTAD+1
4193 47                 MOV     B,A
4194 3AA600             LDA     ENDAD+1
4197 B8                 CMP     B
4198 C0                 RNZ
4199 3AA400             LDA     STRTAD+2
419C 47                 MOV     B,A
419D 3AA700             LDA     ENDAD+2
41A0 B8                 CMP     B
41A1 C9                 RET

               ;This subroutine reads a byte from external memory pointed to
               ;by STRTAD.  The byte read returns in A.

41A2 CD2341    RDEXT:   CALL    PUTST
41A5 7E                 MOV     A,M
41A6 C9                 RET

               ;This subroutine writes a byte to exteral memory pointed to by
               ;DESTAD.  The byte to be written should be in A.

41A7 F5        WREXT:   PUSH    PSW
41A8 CD3641             CALL    PUTDST
41AB F1                 POP     PSW
41AC 77                 MOV     M,A
41AD C9                 RET

               ;This subroutine reads 256 bytes from external memory pointed
               ;to by STRTAD and puts them into BUFFER.
```

```
41AE CD2341    RD256:  CALL    PUTST
41B1 2E00              MVI     L,0               ;zero L reg.
41B3 110001            LXI     D,BUFFER          ;DE gets BUFFER address
41B6 7E        RD2:    MOV     A,M               ;get byte
41B7 12                STAX    D                 ;store it in buffer
41B8 1C                INR     E
41B9 2C                INR     L
41BA C2B641            JNZ     RD2
41BD C9                RET

               ;This subroutine writes 256 bytes from BUFFER to external
               ;memory pointed to by DESTAD.

41BE CD3641    WR256:  CALL    PUTDST
41C1 2E00              MVI     L,0
41C3 110001            LXI     D,BUFFER
41C6 1A        WR2:    LDAX    D                 ;A gets byte from buffer
41C7 77                MOV     M,A
41C8 1C                INR     E
41C9 2C                INR     L
41CA C2C641            JNZ     WR2
41CD C9                RET

               ;The actual commands start below:
               ;Command does nothing, but is useful anyway.

41CE C9        NOPR:   RET

               ;Command to set OCW1 (Mask Byte)

41CF 3AA200    SETMSK: LDA     STRTAD            ;Get the mask byte
41D2 320180            STA     INTCLB
41D5 C9                RET

               ;Command to send EOI to interrupt controller (OCW2)

41D6 3AA200    SETEOI: LDA     STRTAD            ;get the EOI byte
41D9 320080            STA     INTCLA
41DC C9                RET

               ;Command to read the IR and IS registers in the Interrupt
               ;Controller.  Puts IR in RESULT1 and IS in RESULT2.

41DD CD1041    READRG: CALL    PUTCMD
41E0 7D                MOV     A,L
41E1 C60E              ADI     0EH               ;offset of RESULT1
41E3 6F                MOV     L,A
41E4 3E0A              MVI     A,0AH             ;read IR command
41E6 320080            STA     INTCLA
41E9 3A0080            LDA     INTCLA            ;read it
41EC 77                MOV     M,A               ;Store it external
41ED 3E0B              MVI     A,0BH             ;read IS command
41EF 320080            STA     INTCLA
41F2 3A0080            LDA     INTCLA            ;read it
41F5 2C                INR     L                 ;offset of RESULT2
```

```
41F6 77                MOV     M,A             ;Store it external
41F7 C9                RET

              ;Command to set the interrupt response byte

41F8 3AA200   SETRSP:  LDA     STRTAD
41FB 320780            STA     INTRSP
41FE C9                RET

              ;Command to return size of MPX1 (4K or 16K).  If 4K MPX1,
              ;sets RESULT1 to 0, if 16K MPX1, sets RESULT1 to FFH

41FF 3AFF0F   SIZE:    LDA     0FFFH           ;top of 4K RAM
4202 F5                PUSH    PSW             ;save it on the stack
4203 3AFF3F            LDA     3FFFH           ;top of 16K RAM
4206 F5                PUSH    PSW             ;save it as well
4207 3EAA              MVI     A,0AAH
4209 32FF0F            STA     0FFFH
420C 3E55              MVI     A,55H
420E 32FF3F            STA     3FFFH
4211 3AFF0F            LDA     0FFFH
4214 FE55              CPI     55H             ;is it 4K?
4216 CA1E42            JZ      IS4K
4219 3EFF              MVI     A,0FFH
421B C32042            JMP     SIZE2
421E 3E00     IS4K:    MVI     A,0
4220 F5       SIZE2:   PUSH    PSW
4221 CD1041            CALL    PUTCMD
4224 7D                MOV     A,L
4225 C60E              ADI     0EH
4227 6F                MOV     L,A
4228 F1                POP     PSW
4229 77                MOV     M,A
422A F1                POP     PSW
422B 32FF3F            STA     3FFFH
422E F1                POP     PSW
422F 32FF0F            STA     0FFFH
4232 C9                RET

              ;Command to load local RAM from external memory.

4233 CDA241   LDRAM:   CALL    RDEXT           ;read the byte
4236 2AA800            LHLD    DESTAD
4239 77                MOV     M,A
423A CD5A41            CALL    BUMP            ;Bump the pointers
423D CD8741            CALL    CMPAR3          ;are they equal?
4240 C23342            JNZ     LDRAM
4243 CDA241            CALL    RDEXT           ;once more for last byte
4246 2AA800            LHLD    DESTAD
4249 77                MOV     M,A
424A C9                RET

              ;Command to execute program in local RAM.  A "call" is made
              ;to the execution address, so all the program has to do is a
              ;RET to get back to the main loop.
```

```
424B 2AA200     EXRAM:  LHLD    STRTAD            ;address in HL
424E E9                 PCHL                      ;go there

                ;Command to move a block of RAM on the external bus
                ;(slow version).

424F CDA241     BLKMOV: CALL    RDEXT
4252 CDA741             CALL    WREXT
4255 CD5A41             CALL    BUMP
4258 CD8741             CALL    CMPAR3
425B C24F42             JNZ     BLKMOV
425E CDA241             CALL    RDEXT
4261 CDA741             CALL    WREXT
4264 C9                 RET

                ;Command to move a block of RAM on the external bus
                ;(fast version).

4265 CDAE41     FSTMOV: CALL    RD256
4268 CDBE41             CALL    WR256
426B CD6941             CALL    BUMP2
426E CD9041             CALL    CMPAR2
4271 C26542             JNZ     FSTMOV
4274 CDAE41             CALL    RD256
4277 CDBE41             CALL    WR256
427A C9                 RET

                ;Dummy routine that does nothing but return.

427B C9         DUMMY:  RET
```

MPX - THEORY OF OPERATION

The MPX is designed around the Intel 8085 microprocessor. In this application, the 8085 may access resources which are local to the MPX without use of the S-100 bus. Resources external to the MPX may be accessed through a temporary master interface as defined in the IEEE 696/S-100 specification.

The 8085 is a single chip microprocessor which requires very few external support chips. The processor includes a built in oscillator. An external crystal is provided for oscillation at 12.000 Mhz which results in a 166 nsec "T" state. Use of this oscillator rather than the S-100 bus clock makes MPX internal operations asynchronous with activities on the S-100 bus, but allows the MPX to operate at 6 Mhz independent of the external bus speed. (i.e. the MPX may execute local code at 6 Mhz even though the external bus is 2 Mhz). An LS373 (U28) is used to latch the low order address byte from the multiplexed Address-/Data bus. This IC is controlled by the Address Latch Enable (ALE) signal provided by the 8085 for that purpose. A buffer (U29) is provided for a group of loads which are inputs only.

The MPX local environment includes both RAM and ROM memory. The processor Read and Write strobes are "OR"ed and qualified with MEMORY status (by U31) to produce a MEMORY STROBE signal. This signal is used to enable a one of four decoder (U32) which decodes the two high order address lines A14 and A15. The resulting outputs are RAM STB which is decoded for addresses in the range of 0000 through 3FFF, ROM STB which is decoded for addresses of 4000 through 7FFF and INT STB (INT for Internal) which is decoded for 8000 through BFFF.

The RAM consists of either eight 2147 or 2167 type ICs (U10-U17) which are each either 4K or 16K by 1 bit. The high order address inputs A8-A11 come directly from the 8085 while the lower eight lines come from the address latch described above. The data input and data output lines of each IC are tied together to provide a bidirectional connection to the internal data bus. The remaining inputs are Chip Enable (CE) which is driven by the RAM STB as described above and Write Enable (WE) which is controlled by 8085 S1 line. If WE is false, CE will cause the contents of the addressed location to be driven onto the data bus. If WE is true, the memory outputs are forced to their high Z state. CE will cause the data on the bus to be written at the addressed location.

ROM storage is provided by a 2716, 2732 or 2764 EPROM. The ROM STB described above is used to drive the Output Enable (OE). A ROM ENA signal is derived from address and status information (without strobes) by a decoder (U32). This "look ahead" signal is used to drive Chip Enable CE of the ROM and to generate a wait state.

The 8085 samples its RDY input at the rising edge of the clock to determine if a wait state is required. Because of set-up and hold time requirements, the RDY line is held low for a full "T" state from the falling edge of the clock preceding the ROM STB. This function is controlled by a pair of "D" flip-flops (U20) and enabled by S2-1.

The 8085 will fetch the first instruction following a Reset from location 0000. The interrupt flexibility of the MPX requires RAM memory at the interrupt vector locations. This apparent dilemma is resolved by having the ROM and RAM exchange locations during initialization. S-100 bus signals RESET* or SLAVE CLR* clear a counter (U5) which in turn generates INIT. INIT is "OR"ed with address line A14 (U21) so that memory accesses in the range of 0000 to 3FFF will actually access 4000 to 7FFF when INIT is active. The initial instruction fetch by the 8085 will actually come from location 4000 which is the start of the ROM. The ROM contains a jump to 4003 (or the actual start of the code) at this location. When the 8085 goes to fetch the next instruction, the initialization counter which has been counting ALEs counts to 4 removing INIT and hanging the counter. The 8085 address will be used directly to fetch the next address in the ROM.

The remaining features of the MPX hardware make up its interface to the external bus.

The primary function of the MPX is to assist the bus CPU in the servicing of interrupts. To facilitate the capability, the MPX data bus includes an 8259A interrupt controller (U26). The eight active low S-100 bus vectored interrupt lines (VI0* - VI7*) are tied through inverters (U36) to the eight active high interrupt request lines (IR0 - IR7) of the 8259A. (See the 8259A application note in the appendix of this document for a complete description of the 8259A.) The resulting INT line from the 8259A is tied directly to the INTR input of the 8085 and the 8085 Interrupt Acknowledge line (INTA) is tied back to the 8259A INTA input. Thus, bus interrupts that are not masked will interrupt the 8085 directly and 8085 interrupt acknowledge cycles will accept data from the 8259A. Programming the 8259A requires two eight bit ports.

The LS138 one of eight decoder (U48) decodes the three low order address lines (A0 - A2) qualified by Internal Strobe (INT STB) described above to obtain Interrupt Controller Enable (INT CTL ENA), Set Interrupt (SET INT), Interrupt Acknowledge Strobe (INTA STB), A8-15 STB and A16-23 STB making all of these facilities memory mapped within the 8000 to BFFF range. The INT CTL ENA is generated for a pair of addresses as required for programming the 8259A as described above.

An S-100 bus I/O port with a switch selectable address is decoded by the LS2521 (U41). No data is accepted by this port, however writing to the port will generate an attention signal (ATTN) to the 8085 by causing RST7.5 to be asserted.

Two methods are provided for the MPX to call the bus CPU. The SET INT signal described above may set a latch (U24). The output of the latch is buffered (U25). It may be jumpered to any of the S-100 interrupt lines, including NMI and INT. The latch is cleared by an interrupt acknowledge cycle on the S-100 bus which is decoded by (U7). The other CPU call option uses the 8085 Serial Output Data line (SOD). The buffered SOD output (U25), may also be jumpered to any of the S-100 bus interrupt lines.

For environments where the MPX is the only interrupt controller in the system, the MPX may provide a single byte response to the CPU interrupt acknowledge cycle which results from a call by the MPX to the bus CPU. (This response would normally be a RST instruction for 8080 type CPUs, or vector information for 8086/88 or 68000 type processors). Another of the addresses decoded by the LS138 is INTERRUPT RESPONSE. The resulting strobe loads an eight bit latch with the contents of the data bus. The buffered latch will enable its outputs onto the S-100 DI bus during a bus interrupt acknowledge cycle if the interrupt response enable switch is on.

The MPX may also communicate with the S-100 bus as a temporary master. As the name implies, the bus CPU or permanent master will give up the bus for a short time allowing a temporary master to take control. The protocol for transfer of the bus as defined in the IEEE 696/S-100 specification must be carefully adhered to if proper operation is to be obtained. Once the bus is obtained, the temporary master will generate all of the bus signals usually provided by the CPU (with the exception of INTA cycles or releasing the bus to other temporary masters).

The MPX will perform a DMA cycle if either a memory address in the range of C000 - FFFF is accessed or an I/O port is accessed. Since there are no port addresses used on the MPX, all port accesses must be external. The signal External Enable (EXT ENA) is generated on EXT MEM which was decoded from the high order address lines or on I/O status and not interrupt acknowledge. EXT ENA will assert a false level on the 8085 RDY line making the 8085 hang in a wait. If EXT ENA (which was decoded entirely from status signals) has remained until the leading edge of the 8085 strobe, the flip-flop I_WANT will be set. When the bus is available as determined by the signals HOLD* and Hold Acknowledge (pHLDA) both being inactive, I_WANT will set Assert Priority (APRIO). APRIO will assert HOLD* and enable the priority arbitration logic.

Priority arbitration is handled by the three ICs U33,34 and 35. The S-100 bus DMA address bus consists of four open collector lines which are active low. To understand this process, consider the arbitration of the most significant bit DMA3*. If a device has set APRIO and the most significant bit of its priority is a "1", it will assert DMA3* by pulling the line low. If a different device also has APRIO set, but the most significant bit of its priority is a "0", its open collector output will be unable to pull DMA3* high. Based on consideration of this single bit, the second device will see that some device on the bus has a priority bit of "1" where he has a "0" and will know that he is not the highest priority device on the bus at this time. The first device on the other hand will see his own address bit asserted and know that he is the highest priority device (based on consideration of this single bit only).

If a device has asserted a given bit of his priority and there are no other devices asserting a higher priority in that bit, it may enable the next most significant bit. The operation of the bits is cascaded. In a finite amount of time, the address of the highest priority device will have stabilized on the DMA address lines. The device asserting the least significant bit and not finding a higher priority bit on the least significant address line will generate the signal IMHI.

The time required for the arbitration to settle is provided by the bus CPU sensing the HOLD* line one "T" state before acknowledging the bus. This scheme would not work if additional devices could enter the arbitration just prior to the CPU asserting pHLDA, but this may not happen because a device asserts HOLD* once it sets APRIO. HOLD* will lockout other devices by preventing them from setting APRIO. Arbitration really only occurs when two devices set APRIO almost simultaneously.

On receipt of pHLDA, a device will clear APRIO if IMHI is false. If, on the other hand, the device had the highest priority, APRIO will remain set and the control of the bus will be received. This operation procedes as follows:

At the falling edge of the bus clock following pHLDA, the transfer flip-flop XFER will set. XFER will enable the Tri-State buffer (U39) which drives the S-100 bus "P" or processor control lines. At this time, the bus CPU is also driving the same lines. It is very important that both devices drive the lines in exactly the same directions as described by the S-100 bus specifications. The same IC will also assert the disable lines ADSB*, SDSB* and DODSB* which disable the CPU address, status and data output drivers respectively.

At the next rising edge of the bus clock, the signal Bus Cycle (BC) will set. BC enables the MPX drivers for the address, status and data output busses. It also causes Command Disable (CDSB*) to be asserted, turning off the CPU bus drivers for the processor control lines. This overlap in drive on the control bus is necessary to prevent spikes on the active high strobe signals.

The signal BC brackets the MPX cycle on the external bus. The signal pSYNC will go high for the first "T" state with pSTVAL* going low for the second half of pSYNC. The status which is asserted is determined by the 8085 cycle which is still held in a wait. An appropriate S-100 status is decoded for I/O or memory cycles (including M1 cycles) and asserted for the entire cycle. The twenty-four bit extended address is made up of two sections. The high order sixteen bits are driven from registers which are writable at memory locations decoded by the LS138 (U48). The least significant eight bits are taken directly taken from the 8085 address latch. For memory operations, the two address latches define a 256 byte "window" which may be accessed by the 8085. For I/O operations, the port address is asserted directly from the address latch. Since the high order address byte may be software controlled by writing to the latch, the 8085 may simulate Z80 or 16 bit CPU I/O instructions. The address is also asserted for the duration of BC.

At the end of the first "T" state which is signalled by the next rising edge of the bus clock, STB ENA will set inhibiting pSYNC and enabling the bus strobe. Either pDBIN or pWR* will be decoded, again depending on the state of the waiting 8085. At this same edge of the clock, the bus signals RDY and XRDY are sampled. If either signal is false, an additional strobe (wait) state will follow.

At the end of a strobe state which began with the ready lines high, STB INH will be set. This terminates the strobes providing one "T" state of hold time.

At the next clock, the presence of STB INH will clear BC. BC low with STB ENA still active will generate RELEASE. RELEASE clears I_WANT which then clears APRIO. The absence of BC marks the start of the bus transfer back to the CPU. All of the MPX bus drivers except for the control bus are disabled and the CPU control bus drivers are enabled providing the overlap period.

At the next falling edge of the clock, APRIO being low will cause XFER to clear, inhibiting the control bus driver. This completes the DMA cycle as seen from the bus however it is not until the following rising edge of the clock that BC being low clears STB ENA. The falling edge of STB ENA sets END WAIT which will release the 8085. If the 8085 had been in a memory or I/O write cycle, its data has already been transferred. If a memory or I/O read had been performed, the data received from the bus has been latched by the end of the bus strobe and is available to be accepted by the 8085 from the MPX internal data bus.

One more clock is required with STB ENA low to clear STB INH. This completes the DMA cycle returning all of the DMA hardware to its initial state.

APPENDIX

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.

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/Output (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 program 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.

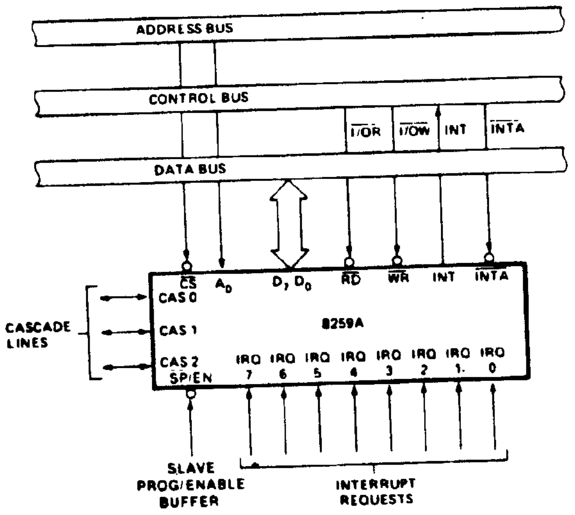


Figure 1. 8259A Interface to Standard System Bus

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 (IR0-IR7) 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 contents 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.

1.3 MCS-86/88™—8259A OVERVIEW

Operation of an MCS-86/88—8259A configuration has basic similarities of the MCS-80/85—8259A configurations. That is, a device can cause an interrupt by pulling one of the 8259A's interrupt request pins (IR0-IR7) high. If the 8259A honors the request, its INT pin will go high, driving the 8086/8088's INTR pin high. Like the 8080A and 8085A, the INTR pin of the 8086/8088 is asynchronous, thus it can receive an interrupt any time. The 8086/8088 can also accept or disregard requests on INTR under software control using the STI (Set Interrupt) or CLI (Clear Interrupt) instructions. These instructions set or clear the interrupt-enabled flag IF. Upon 8086/8088 reset the IF flag is cleared, disabling external interrupts on INTR. Beside the INTR pin, the 8086/8088 provides an NMI (Non-Maskable Interrupt) pin. The NMI functions similar to the 8085A's TRAP; it can't be disabled or masked. NMI has higher priority than INTR.

Although there are some basic similarities, the actual processing of interrupts with an 8086/8088 is different than an 8080A or 8085A. When an interrupt request is present and interrupts are enabled, the 8086/8088 enters its interrupt acknowledge machine cycle. The interrupt acknowledge machine cycle pushes the flag registers onto the stack (as in a PUSHF instruction). It then clears the IF flag which disables interrupts. The contents of 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 $\overline{\text{INTA}}$ 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 $\overline{\text{INTA}}$ signal is available from the 8086/8088 on its $\overline{\text{INTA}}$ pin. If the 8086/8088 is used in the "MAX Mode" the $\overline{\text{INTA}}$ signal is available via the 8288 Bus Controller $\overline{\text{INTA}}$ 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 $\overline{\text{INTA}}$ 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 $\overline{\text{INTA}}$ pulse is used only to signal the 8259A of the honored request. The second $\overline{\text{INTA}}$ 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.

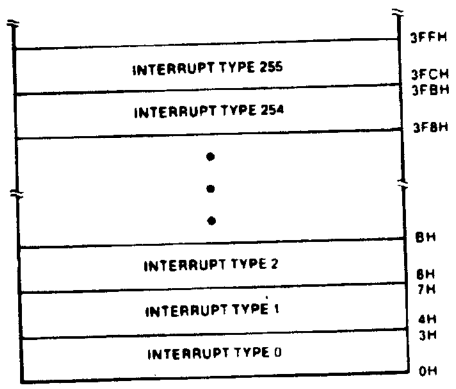


Figure 5. 8086/8088 Interrupt Vector Table

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 x 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.

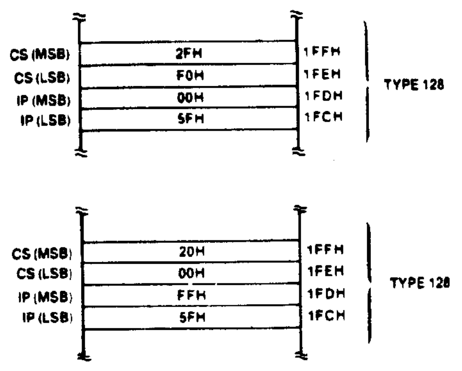


Figure 6. Two Examples of 8086/8088 Interrupt Type 128 Vectoring to Location 2FF5FH

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.

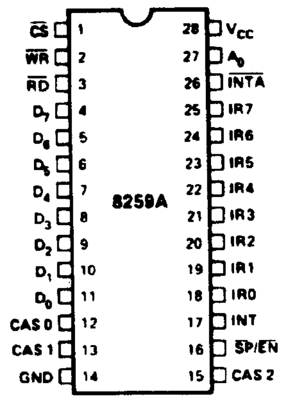
Beside external interrupt generation from the INTR pin, the 8086/8088 is also able to invoke interrupts by software. Three interrupt instructions are provided: INT, INT (Type 3), and INTO. INT is a two byte instruction, the second byte selects the interrupt type. INT (Type 3) is a one byte instruction 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.

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.

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
CAS1-CAS0	CASCADE LINES
SP/EN	SLAVE PROGRAM/ENABLE BUFFER
INT	INTERRUPT OUTPUT
INTA	INTERRUPT ACKNOWLEDGE INPUT
IR0-IR7	INTERRUPT REQUEST INPUTS

BLOCK DIAGRAM

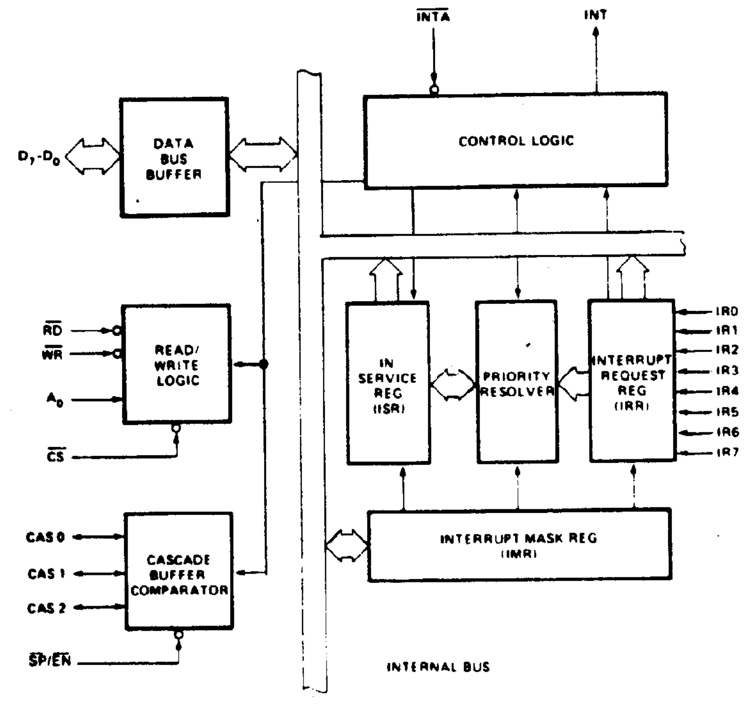


Figure 7. 8259A Block Diagram and Pin Configuration

2.1 INTERRUPT REGISTERS AND CONTROL LOGIC

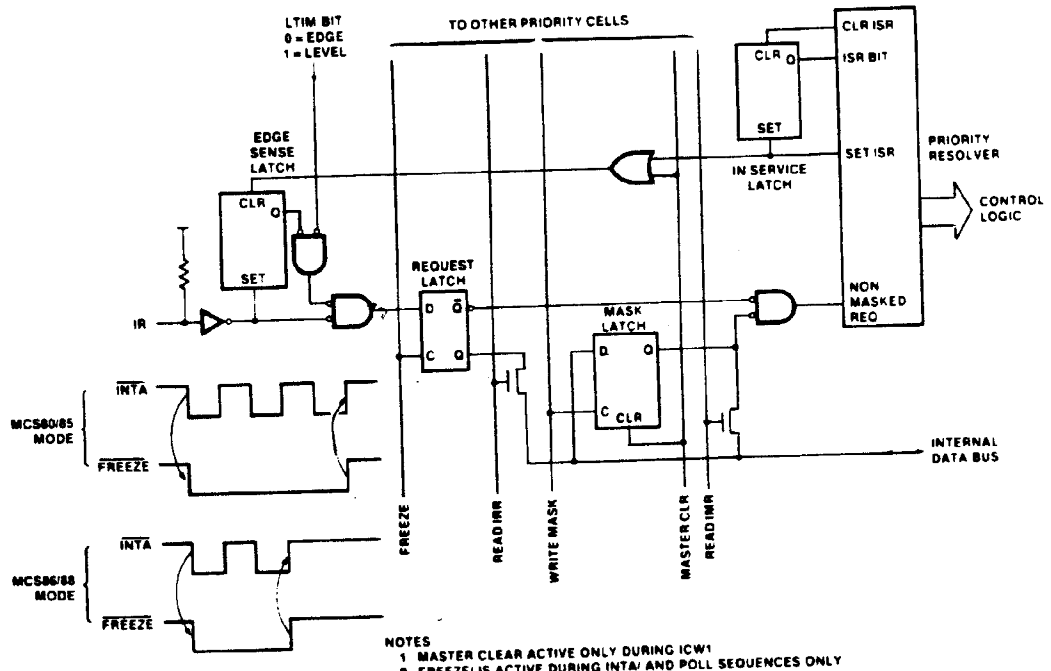
Basically, interrupt requests are handled by three "cascaded" registers: the Interrupt Request Register (IRR) is use to store all the interrupt levels requesting service; the In-Service Register (ISR) stores all the levels which are being serviced; and the Interrupt Mask Register (IMR) stores the bits 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 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 represents 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.



NOTES
1 MASTER CLEAR ACTIVE ONLY DURING ICW1
2 FREEZE/ IS ACTIVE DURING INTA/ AND POLL SEQUENCES ONLY
3 TRUTH TABLE FOR D-LATCH

C	D	Q	OPERATION
1	Di	Di	FOLLOW
0	X	Qn − 1	HOLD

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 affect 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 $\overline{SP}/\overline{EN}$ 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
$\overline{CS}$	1	I	*Chip Select:* A low on this pin enables $\overline{RD}$ and $\overline{WR}$ communication between the CPU and the 8259A. $\overline{INTA}$ functions are independent of $\overline{CS}$.
$\overline{WR}$	2	I	*Write:* A low on this pin when $\overline{CS}$ is low enables the 8259A to accept command words from the CPU.
$\overline{RD}$	3	I	*Read:* A low on this pin when $\overline{CS}$ is low enables the 8259A to release status onto the data bus for the CPU.
D7-D0	4-11	I/O	*Bidirectional Data Bus:* Control, status and interrupt-vector information is transferred via this bus.
CAS0-CAS2	12,13, 15	I/O	*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.
$\overline{SP}/\overline{EN}$	16	I/O	*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 ($\overline{EN}$). When not in the buffered mode it is used as an input to designate a master ($\overline{SP}$ = 1) or slave ($\overline{SP}$ = 0).
INT	17	O	*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.
IR0-IR7	18-25	I	*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).
$\overline{INTA}$	26	I	*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.
A0	27	I	*A0 Address Line:* This pin acts in conjunction with the $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ 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 $\overline{\text{INTA}}$ 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 $\overline{\text{INTA}}$ pulse. Contents of the third interrupt-vector byte is shown in Figure 9C.

A. FIRST INTERRUPT VECTOR BYTE, MCS80/85 MODE

	D7	D6	D5	D4	D3	D2	D1	D0
CALL CODE	1	1	0	0	1	1	0	1

B. SECOND INTERRUPT VECTOR BYTE, MCS80/85 MODE

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

C. THIRD INTERRUPT VECTOR BYTE, MCS80/85 MODE

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

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 $\overline{\text{INTA}}$ pulses issued by the 8086/8088 after the 8259A has raised INT high.

The first $\overline{\text{INTA}}$ 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 $\overline{\text{INTA}}$ 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 $\overline{\text{INTA}}$ pulse and is shown in Figure 10.

	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

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.

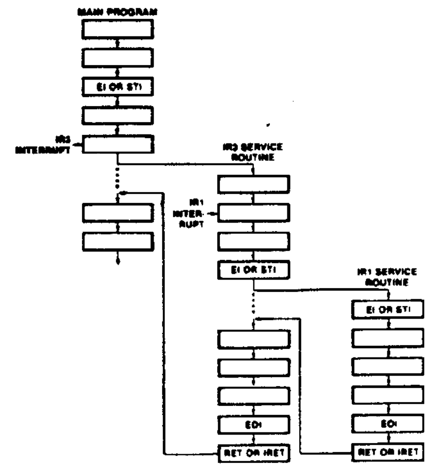


Figure 12. Fully Nested Mode Example (MCS 80/85™ or MCS 86/88™)

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 $\overline{\text{INTA}}$ 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 sevice 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.

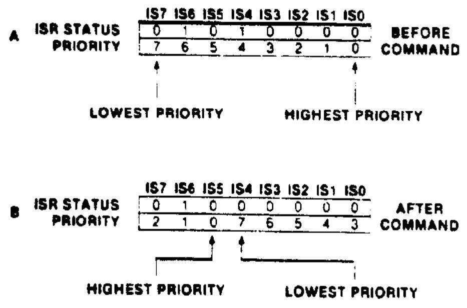


Figure 13. A-B. Rotate on Non-specific EOI Command Example

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 $\overline{\text{INTA}}$ 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.

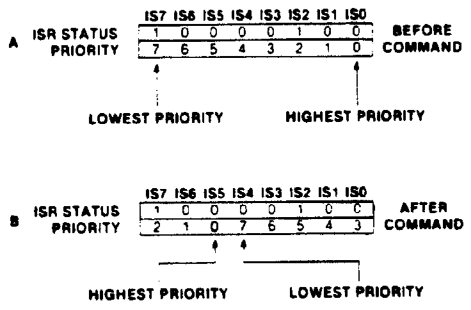


Figure 14. A-B. Set Priority Command Example

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.

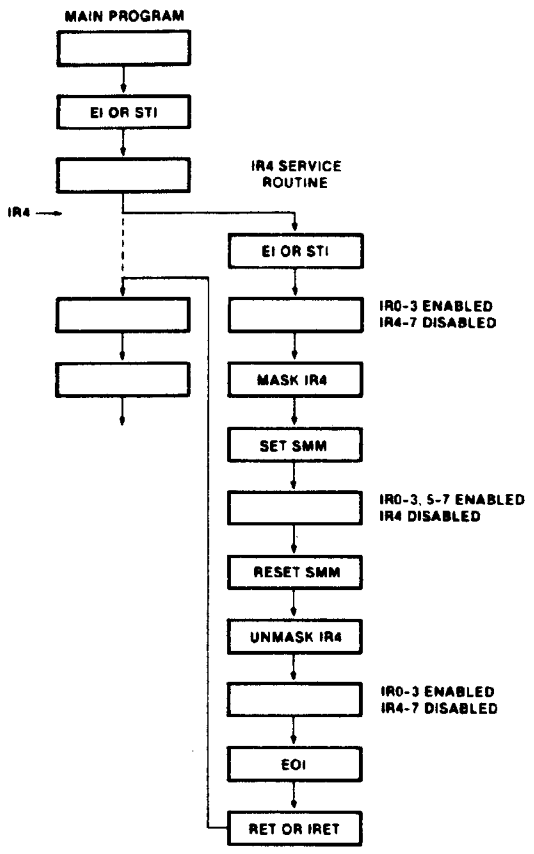


Figure 15. Special Mask Mode Example (MCS 80/85™ or MCS 86/88™)

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 $\overline{\text{INTA}}$ pulse. If on any IR input, the request goes inactive before the first $\overline{\text{INTA}}$ 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 actual 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.

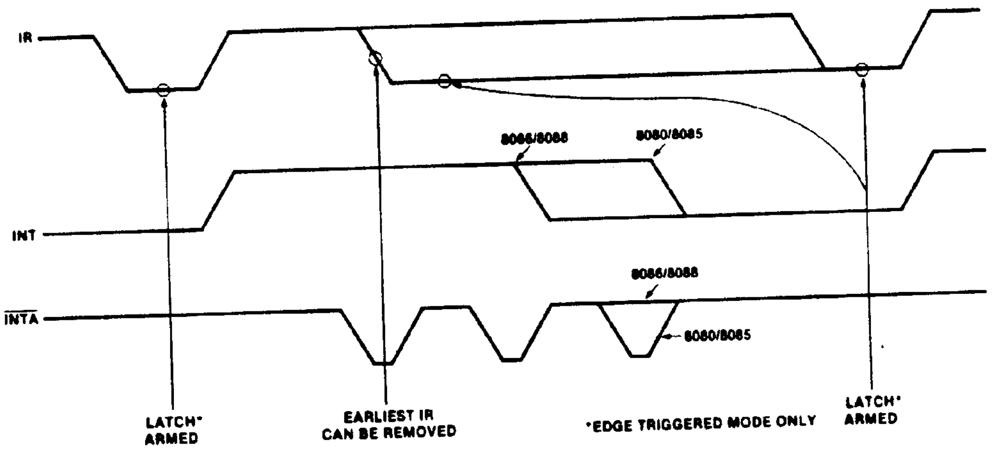


Figure 16. IR Triggering Timing Requirements

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 $\overline{\text{INTA}}$ 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 $\overline{\text{RD}}$ 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 $\overline{\text{RD}}$ 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 $\overline{\text{RD}}$ 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 acknowledge. 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.

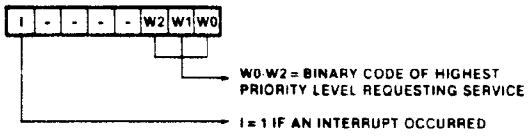

I
-
-
-
-
W2
W1
W0
W0-W2 = BINARY CODE OF HIGHEST PRIORITY LEVEL REQUESTING SERVICE
I = 1 IF AN INTERRUPT OCCURRED

Figure 17. 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".

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 $\overline{SP}/\overline{EN}$ pin, while the $\overline{SP}/\overline{EN}$ 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 INTA pulse).

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 inputs 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

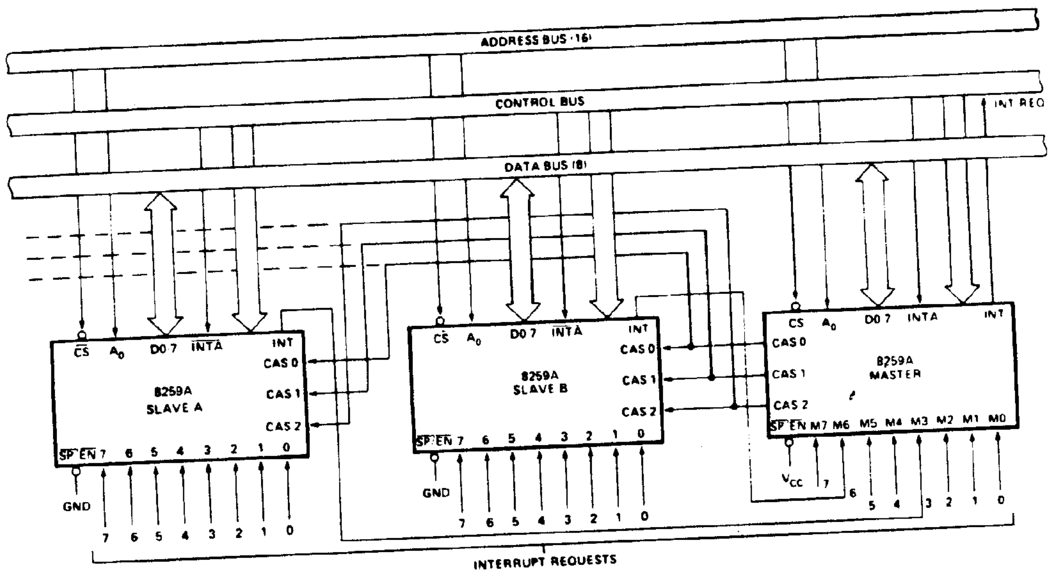


Figure 18. Cascaded 8259A'S 22 Interrupt Levels

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 $\overline{\text{INTA}}$ 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 $\overline{\text{INTA}}$ 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 IR0 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 IR0 (slave IR0). If a slave is connected to the master's IR0 when a non-slave interrupt occurs on another master IR input, erroneous conditions may result. Thus IR0 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 programmed 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.

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 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 intialized, 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 remain ng 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).

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.

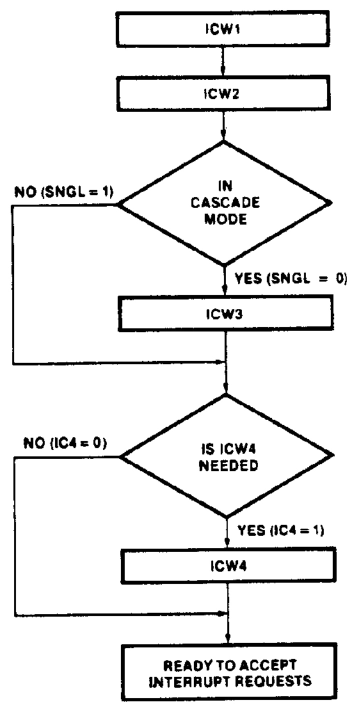
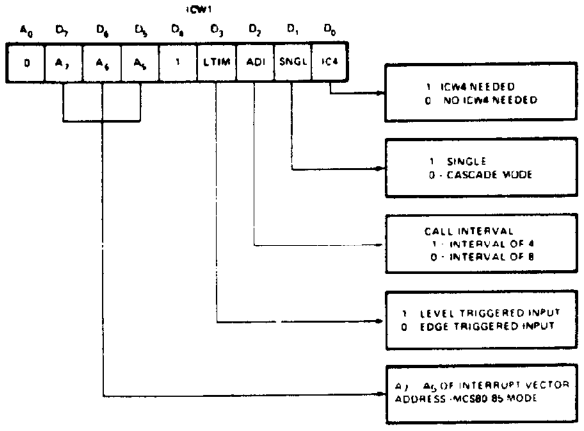
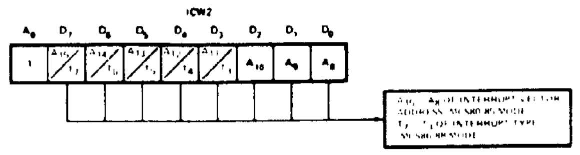
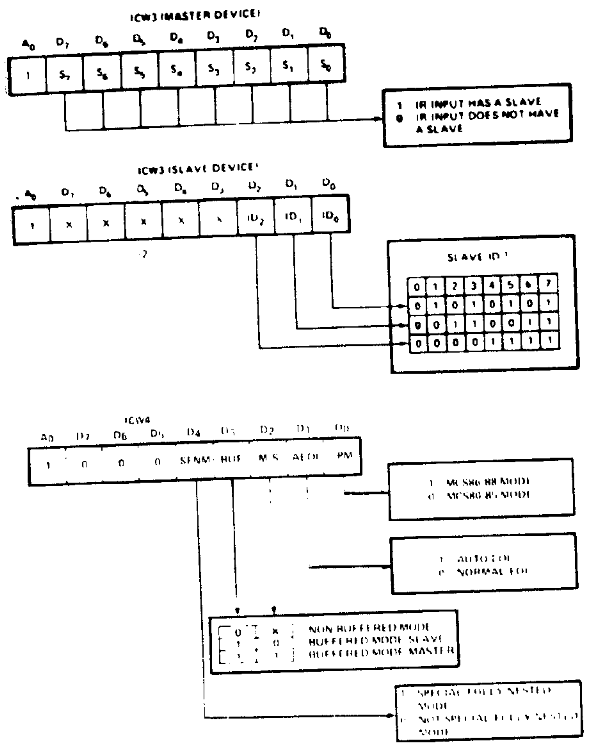


Figure 20. Initialization Flow

Initialization Command Word Format




ICW2
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 A15/T7 A14/T6 A13/T5 A12/T4 A11/T3 A10 A9 A8
A15 - A8 OF INTERRUPT VECTOR ADDRESS MCS80/85 MODE
T7 - T3 OF INTERRUPT TYPE 8086/8088 MODE



NOTE 1
SLAVE ID IS EQUAL TO THE CORRESPONDING MASTER IR INPUT

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.

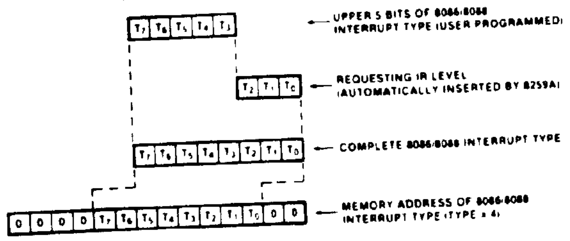
SNGL: The SNGL bit is used to designate whether or not the 8259A is to be used alone or in the cascade mode. If the cascade mode is desired, SNGL must equal 0. In doing this, the 8259A will accept ICW3 for further cascade mode programming. If the 8259A is to be used as the single 8259A within a system, 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.



T7 T6 T5 T4 T3
UPPER 5 BITS OF 8086/8088 INTERRUPT TYPE (USER PROGRAMMED)
T2 T1 T0
REQUESTING IR LEVEL (AUTOMATICALLY INSERTED BY 8259A)
T7 T6 T5 T4 T3 T2 T1 T0
COMPLETE 8086/8088 INTERRUPT TYPE
0 0 0 0 T7 T6 T5 T4 T3 T2 T1 T0 0 0
MEMORY ADDRESS OF 8086/8088 INTERRUPT TYPE (TYPE x 4)

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 $\overline{SP}/\overline{EN}$ 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 $\overline{SP}/\overline{EN}$ 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:

μPM: The μPM 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 $\overline{SP}/\overline{EN}$ pin. If BUF is set to a 1, the buffered mode is programmed and $\overline{SP}/\overline{EN}$ is used as a transceiver enable output. If BUF is 0, the buffered mode isn't programmed and $\overline{SP}/\overline{EN}$ is used for master/slave selection. Note if ICW4 isn't programmed, $\overline{SP}/\overline{EN}$ 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:

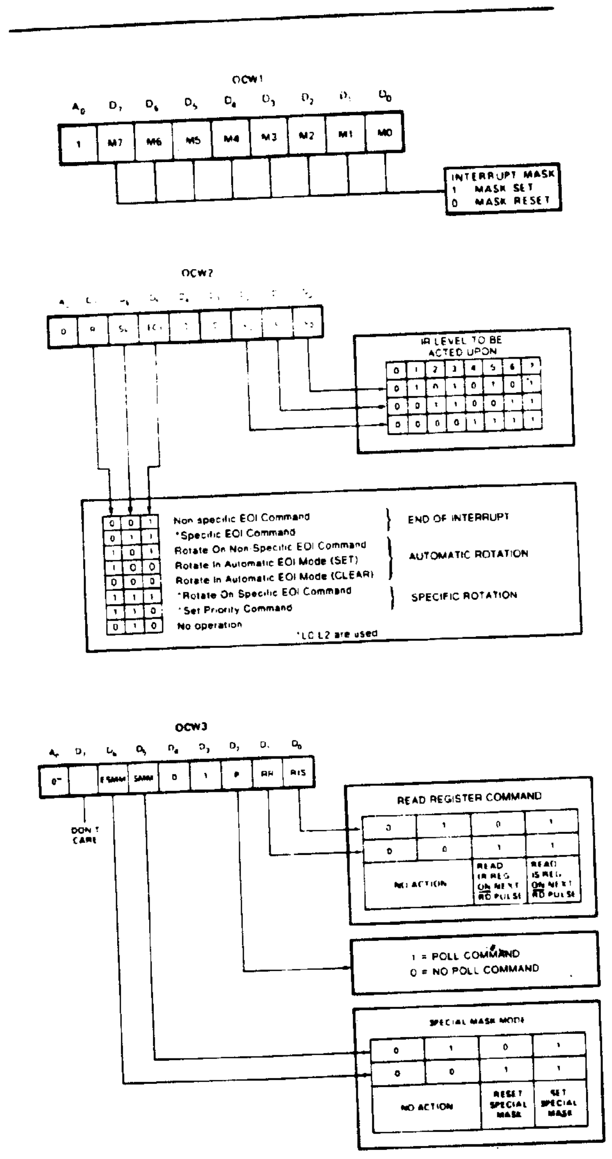
M0-M7: The M0-M7 bits are used to control the masking of IR inputs. If an M bit is set to a 1, it will mask the corresponding IR input. A 0 clears the mask, thus enabling the IR input. These bits convey the same meaning when being read by the processor for status update.

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.

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.



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

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.

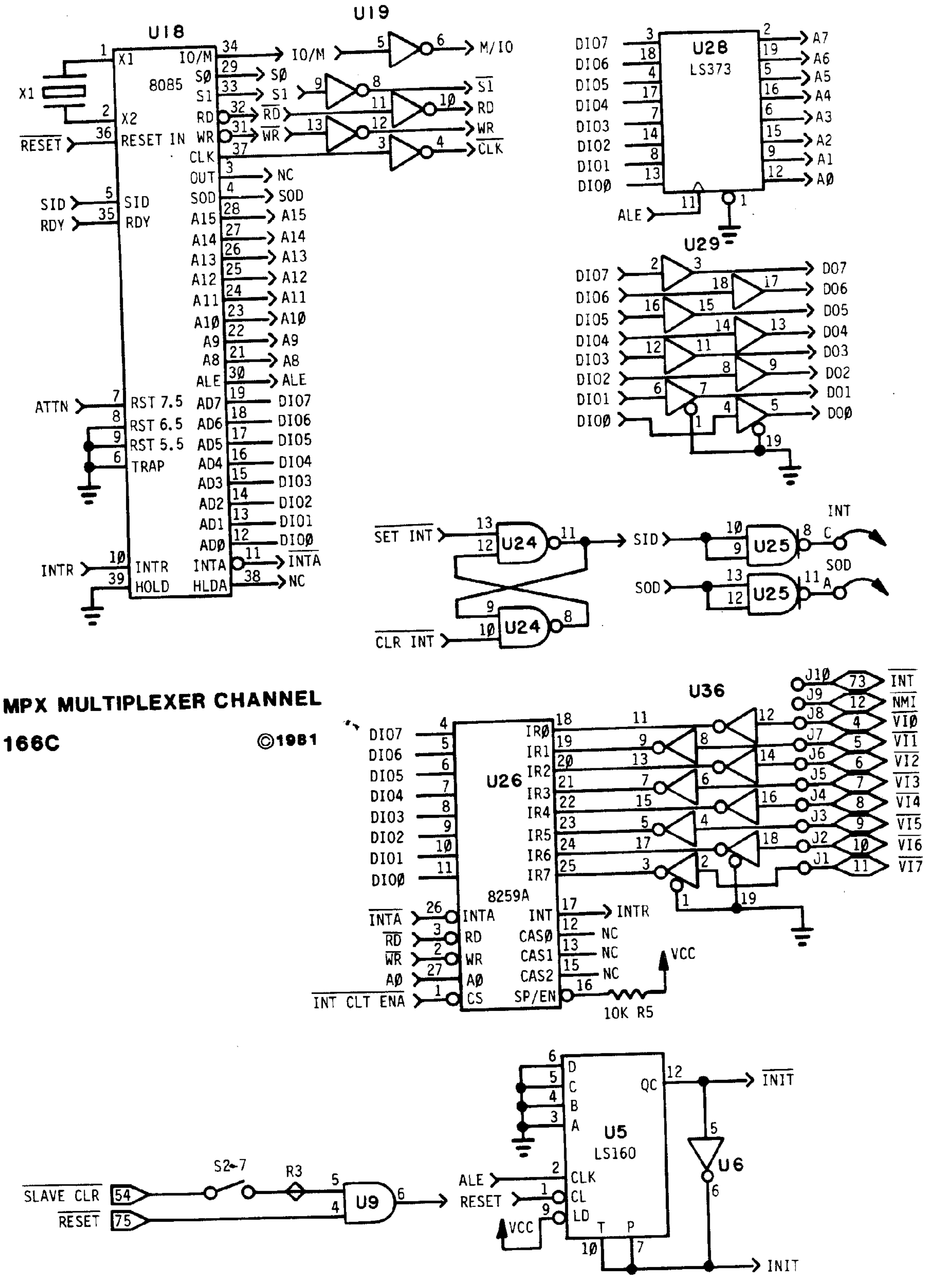
P: The P bit is used to issue the poll command. If P is set to a 1, the poll command is issued. If it is 0, the poll command isn't issued. The poll command will override a read register command if set simultaneously.

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.

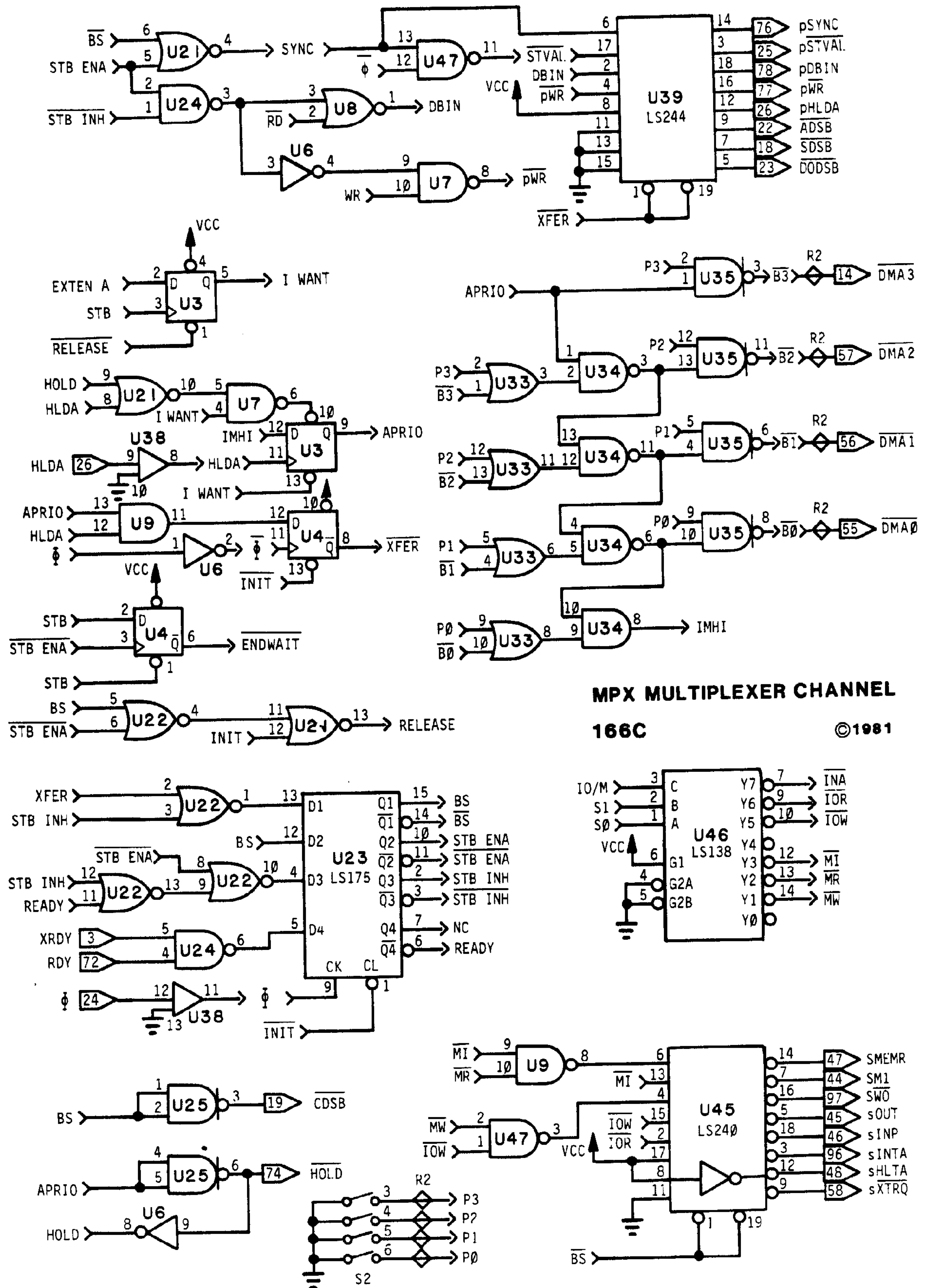
SUMMARY OF 8259A INSTRUCTION SET

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
2	ICW1	B	0	A7	A6	A5	1	1	1	1	0		Format = 4 single level triggered
3	ICW1	C	0	A7	A6	A5	1	0	1	0	0	Byte 1 Initialization	Format = 4 not single edge triggered
4	ICW1	D	0	A7	A6	A5	1	1	1	0	0		Format = 4 not single level triggered
5	ICW1	E	0	A7	A6	0	1	0	0	1	0	No ICW4 Required	Format = 8 single edge triggered
6	ICW1	F	0	A7	A6	0	1	1	0	1	0		Format = 8 single leve triggered
7	ICW1	G	0	A7	A6	0	1	0	0	0	0		Format = 8. not single. edge triggered
8	ICW1	H	0	A7	A6	0	1	1	0	0	0		Format = 8 not single. level triggered
9	ICW1	I	0	A7	A6	A5	1	0	1	1	1		Format = 4, single, edge triggered
10	ICW1	J	0	A7	A6	A5	1	1	1	1	1		Format = 4, single, level triggered
11	ICW1	K	0	A7	A6	A5	1	0	1	0	1	Byte 1 Initialization	Format = 4, not single, edge triggered
12	ICW1	L	0	A7	A6	A5	1	1	1	0	1		Format = 4, not single, level triggered
13	ICW1	M	0	A7	A6	0	1	0	0	1	1	ICW4 Required	Format = 8, single, edge triggered
14	ICW1	N	0	A7	A6	0	1	1	0	1	1		Format = 8, single, level triggered
15	ICW1	O	0	A7	A6	0	1	0	0	0	1		Format = 8, not single, edge triggered
16	ICW1	P	0	A7	A6	0	1	1	0	0	1		Format = 8, not single, level triggered
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	1	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	
37	ICW4	NB	1	0	0	0	1	0	0	0	1	ICW4 NB through ICW4 ND are identical to	
38	ICW4	NC	1	0	0	0	1	0	0	1	0	ICW4 B through ICW4 D with the addition of	
39	ICW4	ND	1	0	0	0	1	0	0	1	1	Fully Nested Mode	
40	ICW4	NE	1	0	0	0	1	0	1	0	0	Fully Nested Mode, MCS-80/85 non buffered no AEOI	
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		
46	ICW4	NK	1	0	0	0	1	1	0	1	0	ICW4 NF through ICW4 NP are identical to	
47	ICW4	NL	1	0	0	0	1	1	0	1	1	ICW4 F through ICW4 P with the addition of	
48	ICW4	NM	1	0	0	0	1	1	1	0	0	Fully Nested Mode	
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	Load mask register, read mask register	
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	C	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	



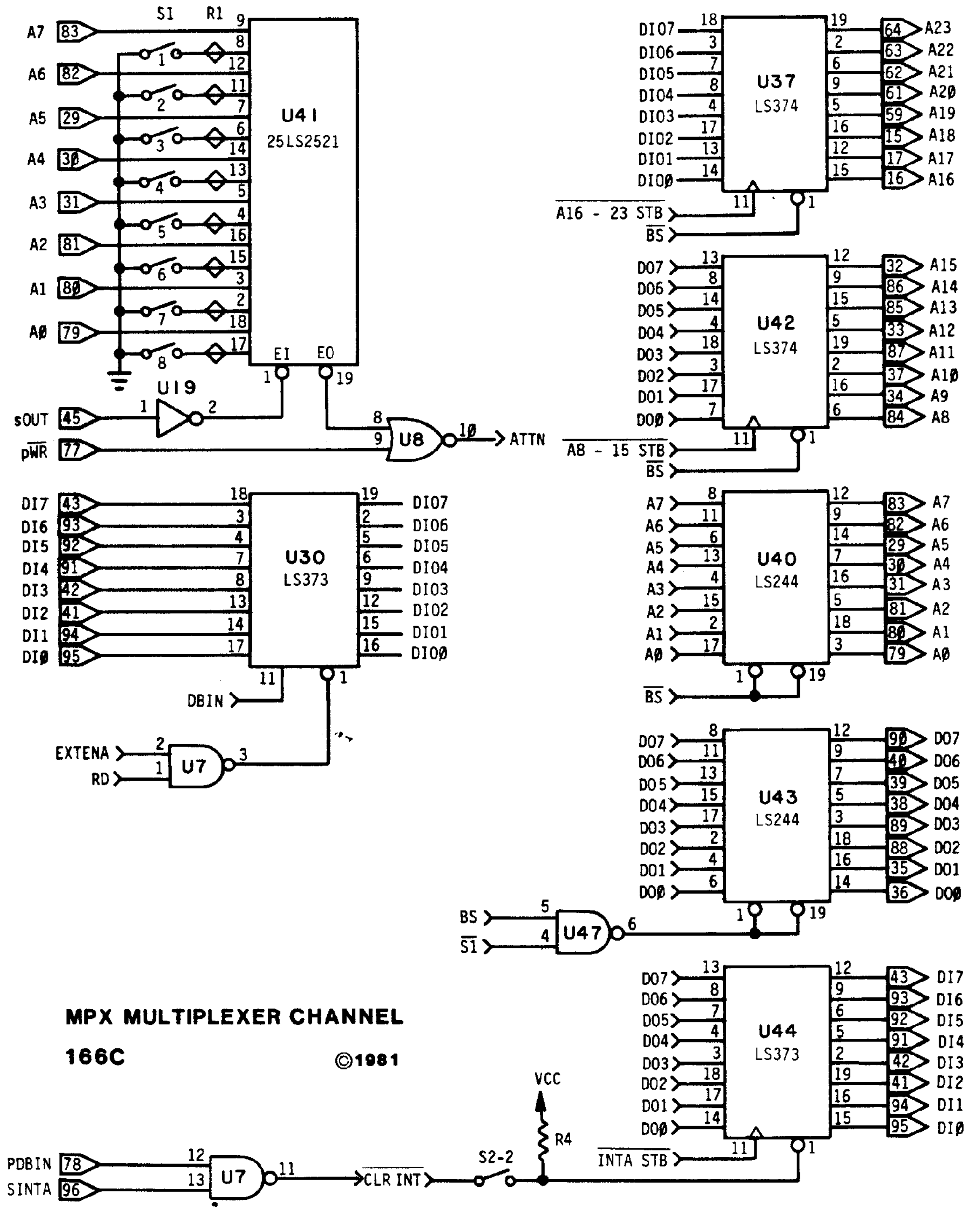
MPX MULTIPLEXER CHANNEL

166C ©1981



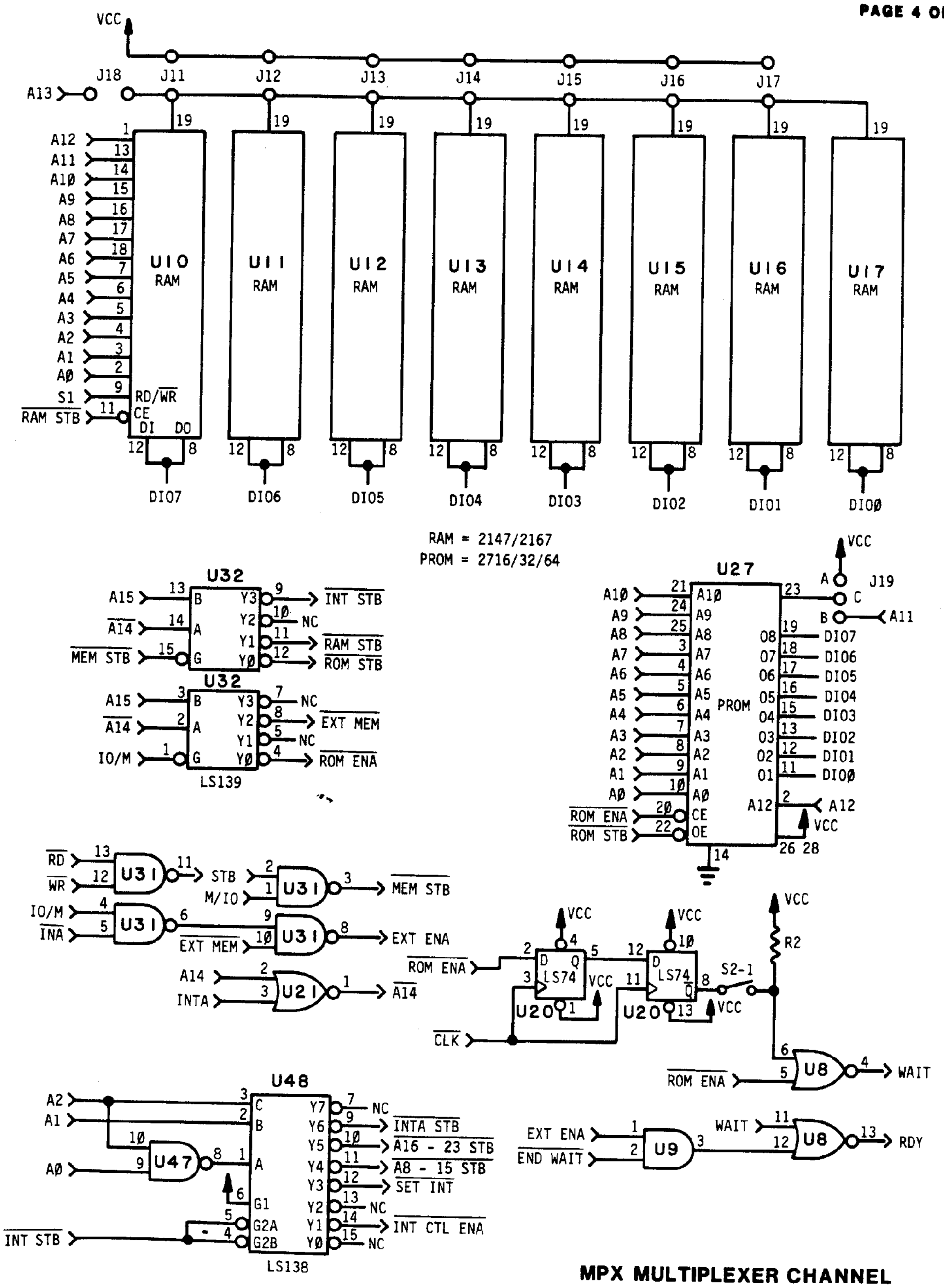
MPX MULTIPLEXER CHANNEL

166C ©1981



MPX MULTIPLEXER CHANNEL

166C ©1981



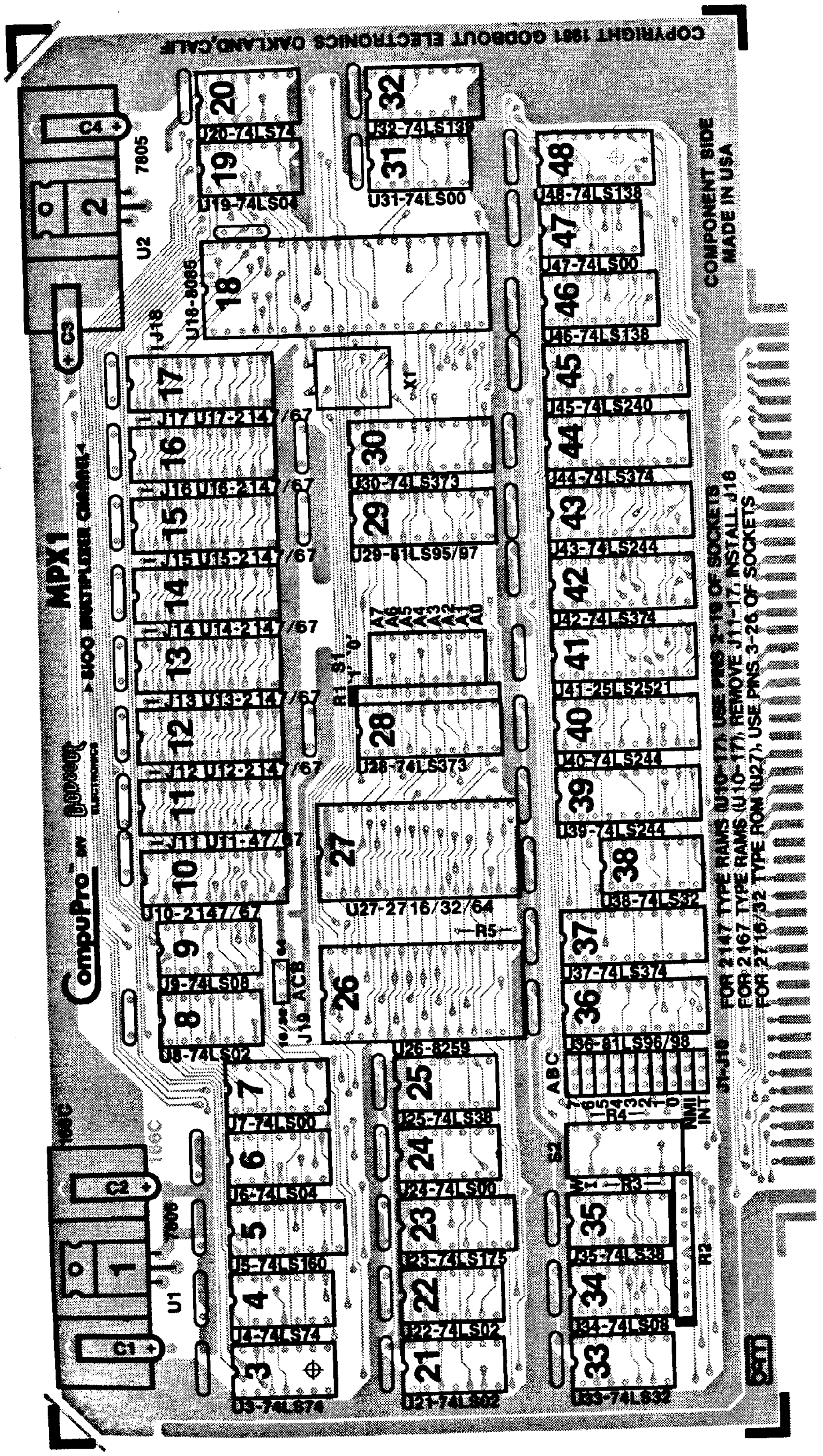
MPX MULTIPLEXER CHANNEL

166C

©1981

PARTS LIST FOR MPX-1

QTY	DESCRIPTION	
SEMICONDUCTORS		
4	74LS00	Quad two input NAND (U7,24,31,47)
3	74LS02	Quad two input NOR (U8,21,22)
2	74LS04	Hex Inverter (U6,19)
2	74LS08	Quad two input AND (U9,34)
2	74LS32	Quad two input OR (U33,38)
2	74LS38	Quad two input NAND O.C. (U25,35)
3	74LS74	Dual D Flip-Flop (U3,4,20)
2	74LS138	One of eight decoder (U46,48)
1	74LS139	Dual one of four decoder (U32)
1	74LS160	4 bit counter (U5)
1	74LS175	Quad D Latch (U23)
1	74LS240	Octal Inverting Bus Driver (U45)
3	74LS244	Octal Bus Driver (U39,40,43)
2	74LS373	Octal Transparent Latch (U28,30)
3	74LS374	Octal D Latch (U37,42,44)
1	81LS95/97	Octal Buffer (U29)
1	81LS96/98	Octal Inverting Buffer (U36)
1	25LS2521	Octal Comparator (U41)
1	8085AH-1	6 MHz CPU (U18)
1	8259A	Interrupt Controller (U26)
1	2716	Type EPROM w/MPX software (U27)
2	7805	5 Volt Positive Voltage Regulators (U1,2)
8	2147	Type RAM chips (U10-17) 4K version
or		
8	2167	Type RAM chips (U10-17) 16K version
OTHER MISC. ELECTRICAL COMPONENTS		
2	SIP Resistor Packs (R1,2)	
2	4.7K ohm resistor (R3,4)	
1	10K ohm resistor (R5)	
4	10V tantalum capacitors (C1-4)	
41	Bypass Capacitors (all unmarked)	
1	Crystal 12 MHz (X1)	
2	8 position DIP switch (S1,2)	



COMPONENT LAYOUT

IF YOU NEED ASSISTANCE ALWAYS CONTACT
YOUR **COMPUPRO DEALER FIRST**

CUSTOMER SERVICE INFORMATION

Our paramount concern is that you be satisfied with any Godbout CompuPro product. If this product fails to operate properly, it may be returned to us for service; see warranty information below.

If you need further information feel free to write us at:

Box 2355, Oakland Airport, CA 94614-0355

LIMITED WARRANTY INFORMATION

Godbout Electronics will repair or replace, at our option, any parts found to be defective in either materials or workmanship for a period of 1 year from date of invoice. Defective parts *MUST* be returned for replacement.

If a defective part causes a Godbout Electronics product to operate improperly during the 1 year warranty period, we will service it free (original owner only) if delivered and shipped at owner's expense to and from Godbout Electronics. If improper operation is due to an error or errors on the part of the purchaser, there may be a repair charge. Purchaser will be notified if this charge exceeds $50.00.

We are not responsible for damage caused by the use of solder intended for purposes other than electronic equipment construction, failure to follow printed instructions, misuse or abuse, unauthorized modifications, use of our products in applications other than those intended by Godbout Electronics, theft, fire, or accidents.

Return to purchaser of a fully functioning unit meeting all advertised specifications in effect as of date of purchase is considered to be complete fulfillment of all warranty obligations assumed by Godbout Electronics. This warranty covers only products marketed by Godbout Electronics and does not cover other equipment used in conjunction with said products. We are not responsible for incidental or consequential damages.

Prices and specifications are subject to change without notice, owing to the volatile nature and pricing structure of the electronics industry.

BULLDOG COMPUTER
IBM PC - XT - AT COMPUPRO
1334 Chapel Street
New Haven, CT 06511
(203) 777-1476 or -7763

COMPUPRO division GODBOUT ELECTRONICS - BOX 2355 OAKLAND AIRPORT, CA 94614