intel®

APPLICATION NOTE

AP-59

September 1979

Using the 8259A Programmable Interrupt Controller

Robin Jigour
Microcomputer Applications

121500-001

Using the 8259A Programmable Interrupt Controller

Contents

INTRODUCTION

The Intel 8259A is a Programmable Interrupt Controller (PIC) designed for use in real-time interrupt driven microcomputer systems. The 8259A manages eight levels of interrupts and has built-in features for expansion up to 64 levels with additional 8259A's. Its versatile design allows it to be used within MCS-80, MCS-85, MCS-86, and MCS-88 microcomputer systems. Being fully programmable, the 8259A provides a wide variety of modes and commands to tailor 8259A interrupt processing for the specific needs of the user. These modes and commands control a number of interrupt oriented functions such as interrupt priority selection and masking of interrupts. The 8259A programming may be dynamically changed by the software at any time, thus allowing complete interrupt control throughout program execution.

The 8259A is an enhanced, fully compatible revision of its predecessor, the 8259. This means the 8259A can use all hardware and software originally designed for the 8259 without any changes. Furthermore, it provides additional modes that increase its flexibility in MCS-80 and MCS-85 systems and allow it to work in MCS-86 and MCS-88 systems. These modes are:

- MCS-86/88 Mode
- Automatic End of Interrupt Mode
- Level Triggered Mode
- Special Fully Nested Mode
- Buffered Mode

Each of these are covered in depth further in this application note.

This application note was written to explain completely how to use the 8259A within MCS-80, MCS-85, MCS-86, and MCS-88 microcomputer systems. It is divided into five sections. The first section, "Concepts", explains the concepts of interrupts and presents an overview of how the 8259A works with each microcomputer system mentioned above. The second section, "Functional Block Diagram", describes the internal functions of the 8259A in block diagram form and provides a detailed functional description of each device pin. "Operation of the 8259A", the third section, explains in depth the operation and use of each of the 8259A modes and commands. For clarity of explanation, this section doesn't make reference to the actual programming of the 8259A. Instead, all programming is covered in the fourth section, "Programming the 8259A". This section explains how to program the 8259A with the modes and commands mentioned in the previous section. These two sections are referenced in Appendix A. The fifth and final section "Application Examples", shows the 8259A in three typical applications. These applications are fully explained with reference to both hardware and software.

The reader should note that some of the terminology used throughout this application note may differ slightly from existing data sheets. This is done to better clarify and explain the operation and programming of the 8259A.

1. CONCEPTS

In microcomputer systems there is usually a need for the processor to communicate with various Input/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.

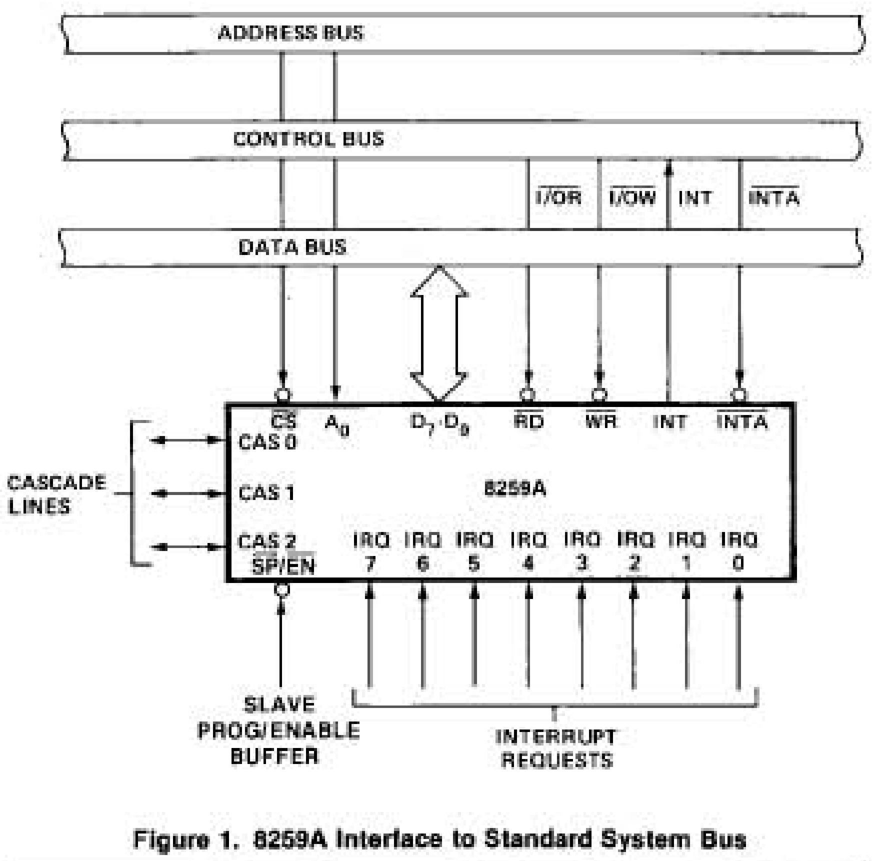


Figure 1. 8259A Interface to Standard System Bus

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 $\overline{\text{INTA}}$ (Interrupt Acknowledge) pulse via the 8228 System Controller Bus Driver. This $\overline{\text{INTA}}$ 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 $\overline{\text{INTA}}$ pulses from the 8080A via the 8228. Upon receiving the first $\overline{\text{INTA}}$ 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 $\overline{\text{INTA}}$ 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 $\overline{\text{INTA}}$ pulse and the upper 8 bits (MSB) during the third $\overline{\text{INTA}}$ 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.

1.2 MCS-85™—8259A OVERVIEW

An MCS-85—8259A configuration processes interrupts in much the same format as an MCS-80—8259A configuration. When an interrupt occurs, a sequence of three $\overline{\text{INTA}}$ pulses causes the 8259A to release onto the data bus a CALL instruction and an interrupt-vector address for the corresponding service routine. Other events that occur during the 8080A interrupt machine cycle, such as disabling interrupts and not incrementing the program counter, also occur in the 8085A interrupt acknowledge machine cycle. Additionally, the instructions for saving registers, enabling or disabling of interrupts, and returning from service routines are literally the same.

The 8085A, however, has a different interrupt hardware scheme as shown in Figure 3. For one, the 8085A supplies its own $\overline{\text{INTA}}$ output pin rather than using an addi-

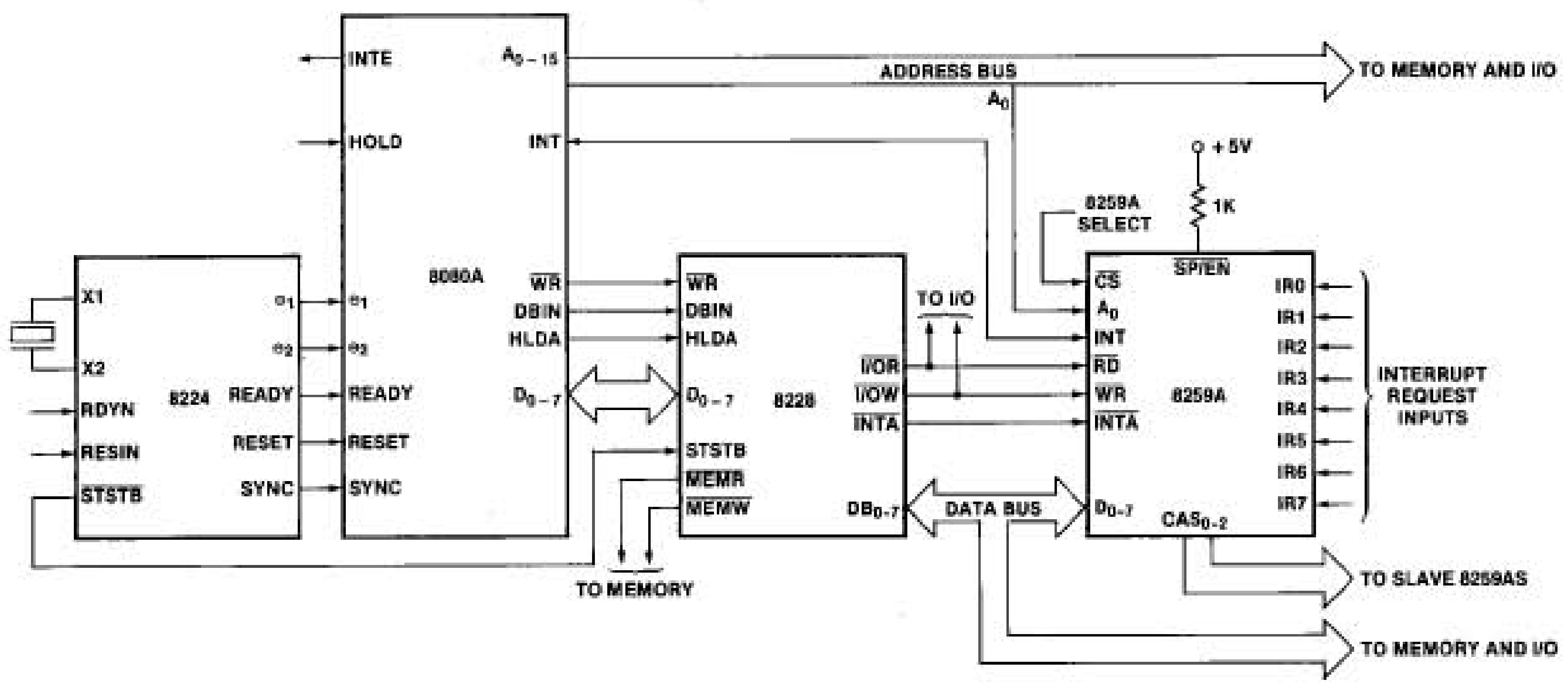


Figure 2. MCS-80 8259A Basic Configuration Example

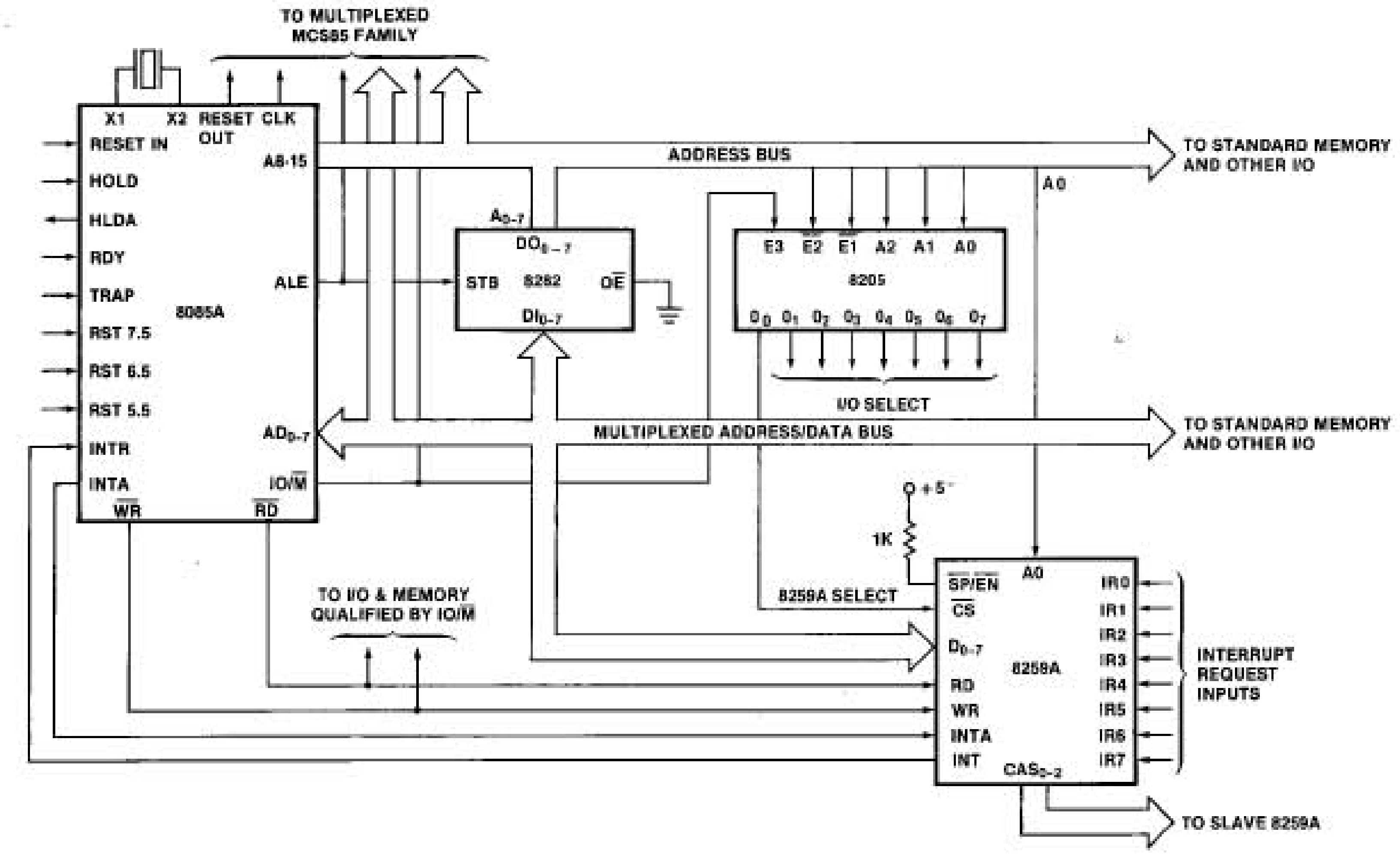


Figure 3. MCS-85™ 8259A Basic Configuration Example

tional chip, as the 8080A uses the 8228 System Controller Bus Driver. Another hardware difference is the 8085A has five hardware interrupt pins: INTR, RST 7.5, RST 6.5, RST 5.5, and TRAP. The INTR (Interrupt Request) pin is the equivalent to the 8080A's INT pin. The RST (Restart) pins and TRAP pin are all restart interrupts which vector program execution to an individual dedicated address when asserted. The important factor associating these interrupts is their relative priority, as shown below:

TRAP	Highest Priority
RST 7.5	
RST 6.5	
RST 5.5	
INTR	Lowest Priority

The INTR pin has lowest priority among the other 8085A hardware interrupts. Thus, precautions to prevent interrupting 8259A service routines may be necessary. This, of course, depends on how the 8085A interrupts are being used in a particular application. Such precautions can be implemented, however, by masking the RST pins using the SIM instruction. The TRAP pin on the other hand is non-maskable; all interrupt pins but TRAP can be controlled by the EI (Enable Interrupt) and DI (Disable Interrupt) instructions.

For a complete description of the 8085A interrupt structure, refer to the MCS-85 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.

Figure 4 shows an MCS-86 MAX Mode system interfacing with an 8259A on the local bus. This MCS-86—8259A configuration is also representative of an MCS-88—8259A configuration except for the data bus which is 16 bits for 8086 and 8 bits for 8088. In the MCS-86 system the 8259A must be on the lower 8 bits of the data bus. Note that the 8259A could also be interfaced on the system bus.

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

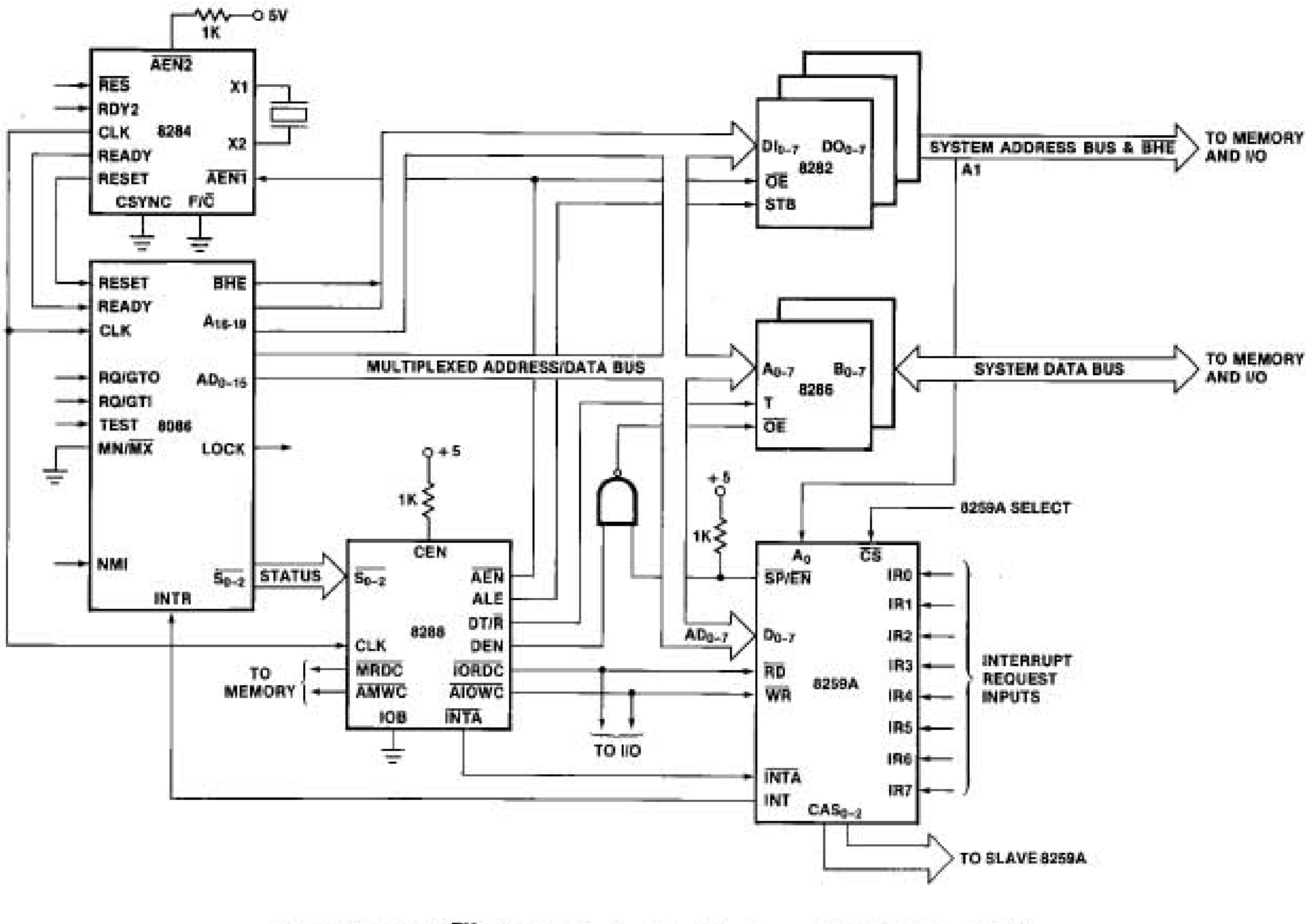


Figure 4. MSC-86™ 8259A Basic Configuration Example (8086 in Max. Mode)

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.

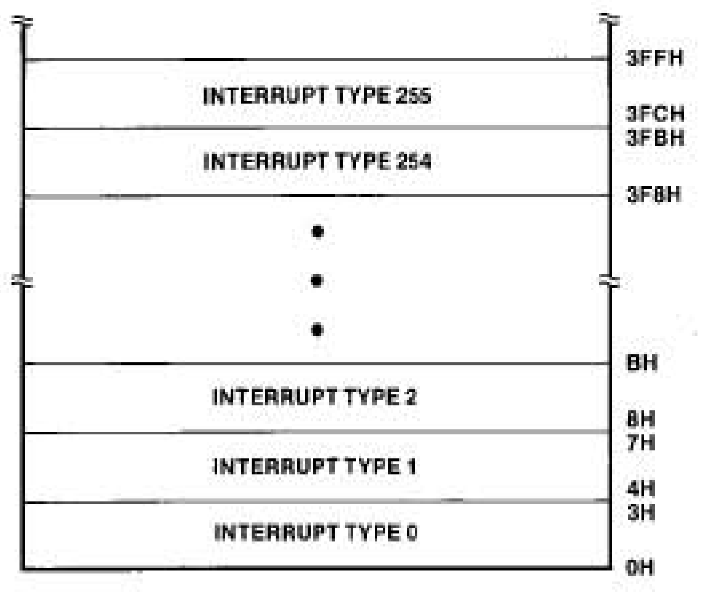


Figure 5. 8086/8088 Interrupt Vector Table

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 4×80H, 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.

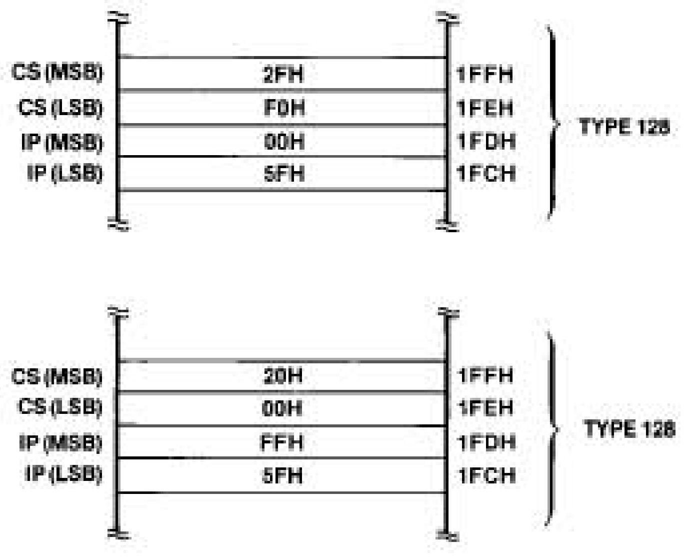


Figure 6. Two Examples of 8086/8088 Interrupt Type-128 Vectoring to Location 2FF5FH

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.

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.

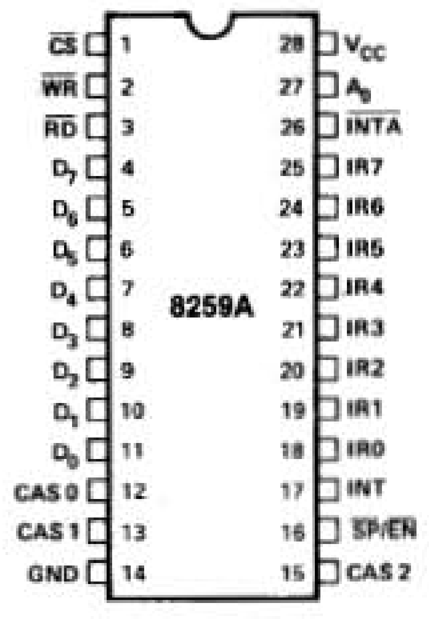
Figure 8 shows conceptually how the Interrupt Request (IR) input handles an interrupt request and how the various interrupt registers interact. The figure represents one of eight "daisy-chained" priority cells, one for each IR input.

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.

PIN CONFIGURATION



PIN NAMES

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A_0	COMMAND SELECT ADDRESS
$\overline{CS}$	CHIP SELECT
CAS1-CAS0	CASCADE LINES
$\overline{SP}/\overline{EN}$	SLAVE PROGRAM/ENABLE BUFFER
INT	INTERRUPT OUTPUT
$\overline{INTA}$	INTERRUPT ACKNOWLEDGE INPUT
IR0-IR7	INTERRUPT REQUEST INPUTS

BLOCK DIAGRAM

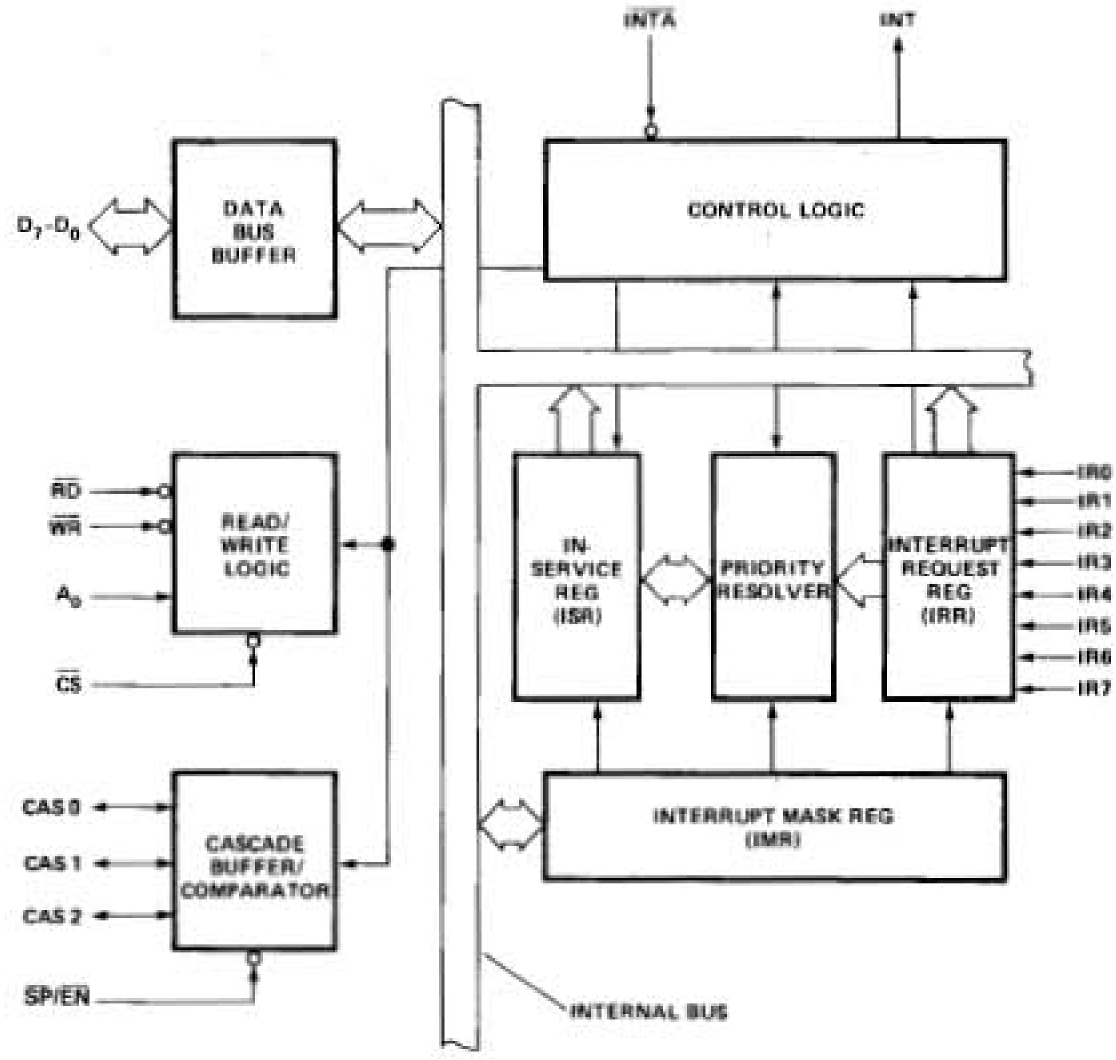
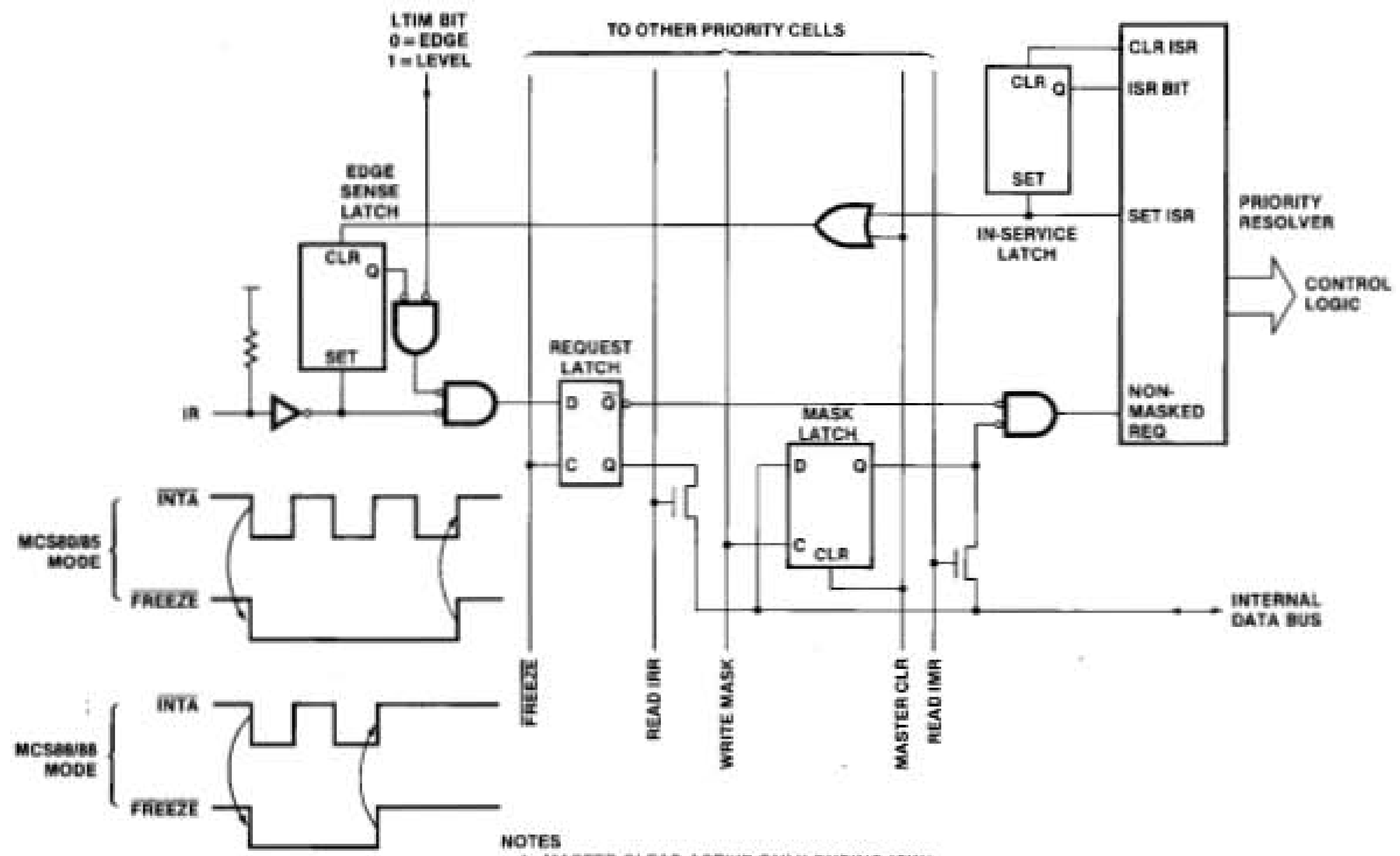


Figure 7. 8259A Block Diagram and Pin Configuration



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

Name	Pin #	I/O	Function
$\overline{CS}$	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 $\overline{INTA}$ 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 $\overline{INTA}$ 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 $\overline{INTA}$ 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 $\overline{INTA}$ pulse. Contents of the third interrupt-vector byte is shown in Figure 9C.

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

A. FIRST INTERRUPT VECTOR BYTE, MCS80/85 MODE

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

B. SECOND INTERRUPT VECTOR BYTE, MCS80/85 MODE

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

C. THIRD INTERRUPT VECTOR BYTE, MCS80/85 MODE

Figure 9. 9A–C. Interrupt-Vector Bytes for 8259A, MCS 80/85 Mode

MCS-86/88™ Mode

When programmed in the MCS-86/88 mode, the 8259A should only be used within an MCS-86 or MCS-88 system. In this mode, the 8086/8088 will handle interrupts in the format described earlier in the "8259A—8086/8088 Overview".

Upon interrupt in the MCS-86/88 mode, the 8259A will output a single interrupt-vector byte to the data bus. This is in response to only two $\overline{\text{INTA}}$ 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.

IR	D7	D6	D5	D4	D3	D2	D1	D0
7	T7	T6	T5	T4	T3	1	1	1
6	T7	T6	T5	T4	T3	1	1	0
5	T7	T6	T5	T4	T3	1	0	1
4	T7	T6	T5	T4	T3	1	0	0
3	T7	T6	T5	T4	T3	0	1	1
2	T7	T6	T5	T4	T3	0	1	0
1	T7	T6	T5	T4	T3	0	0	1
0	T7	T6	T5	T4	T3	0	0	0

Figure 10. Interrupt Vector Byte, MCS 86/88™ Mode

3.2 INTERRUPT PRIORITIES

A variety of modes and commands are available for controlling interrupt priorities of the 8259A. All of them are programmable, that is, they may be changed dynamically under software control. With these modes and commands, many possibilities are conceivable, giving the user enough versatility for almost any interrupt controlled application.

Fully Nested Mode

The fully nested mode of operation is a general purpose priority mode. This mode supports a multilevel-interrupt structure in which priority order of all eight IR inputs are arranged from highest to lowest.

Unless otherwise programmed, the fully nested mode is entered by default upon initialization. At this time, IR0 is assigned the highest priority through IR7 the lowest. The fully nested mode, however, is not confined to this IR structure alone. Once past initialization, other IR inputs can be assigned highest priority also, keeping the multilevel-interrupt structure of the fully nested mode. Figure 11A–C shows some variations of the priority structures in the fully nested mode.

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	7	6	5	4	3	2	1	0

A

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	4	3	2	1	0	7	6	5

B

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	1	0	7	6	5	4	3	2

C

Figure 11. A–C. Some Variations of Priority Structure in the Fully Nested Mode

Further explanation of the fully nested mode, in this section, is linked with information of general 8259A interrupt operations. This is done to ease explanation to the user in both areas.

In general, when an interrupt is acknowledged, the highest priority request is determined from the IRR (Interrupt Request Register). The interrupt vector is then placed on the data bus. In addition, the corresponding bit in the ISR (In-Service Register) is set to designate the routine in service. This ISR bit remains set until an EOI (End-Of-Interrupt) command is issued to the 8259A. EOI's will be explained in greater detail shortly.

In the fully nested mode, while an ISR bit is set, all further requests of the same or lower priority are inhibited from generating an interrupt to the microprocessor. A higher priority request, though, can generate an interrupt, thus vectoring program execution to its service routine. Interrupts are only acknowledged, however, if the microprocessor has previously executed an "Enable Interrupts" instruction. This is because the interrupt request pin on the microprocessor gets disabled automatically after acknowledgement of any interrupt. The assembly language instructions used to enable interrupts are "EI" for 8080A/8085A and "STI" for 8086/8088. Interrupts can be disabled by using the instruction "DI" for 8080A/ 8085A and "CLI" for 8086/8088. When a routine is completed a "return" instruction is executed, "RET" for 8080A/8085A and "IRET" for 8086/8088.

Figure 12 illustrates the correct usage of interrupt related instructions and the interaction of interrupt levels in the fully nested mode.

Assuming the IR priority assignment for the example in Figure 12 is IR0 the highest through IR7 the lowest, the sequence is as follows. During the main program, IR3 makes a request. Since interrupts are enabled, the microprocessor is vectored to the IR3 service routine. During the IR3 routine, IR1 asserts a request. Since IR1 has higher priority than IR3, an interrupt is generated. However, it is not acknowledged because the microprocessor disabled interrupts in response to the IR3 interrupt. The IR1 interrupt is not acknowledged until the "Enable Interrupts" instruction is executed. Thus the IR3 routine has a "protected" section of code over which no interrupts (except non-maskable) are allowed. The IR1 routine has no such "protected" section since an "Enable Interrupts" instruction is the first one in its service routine. Note that in this example the IR1 request must stay high until it is acknowledged. This is covered in more depth in the "Interrupt Triggering" section.

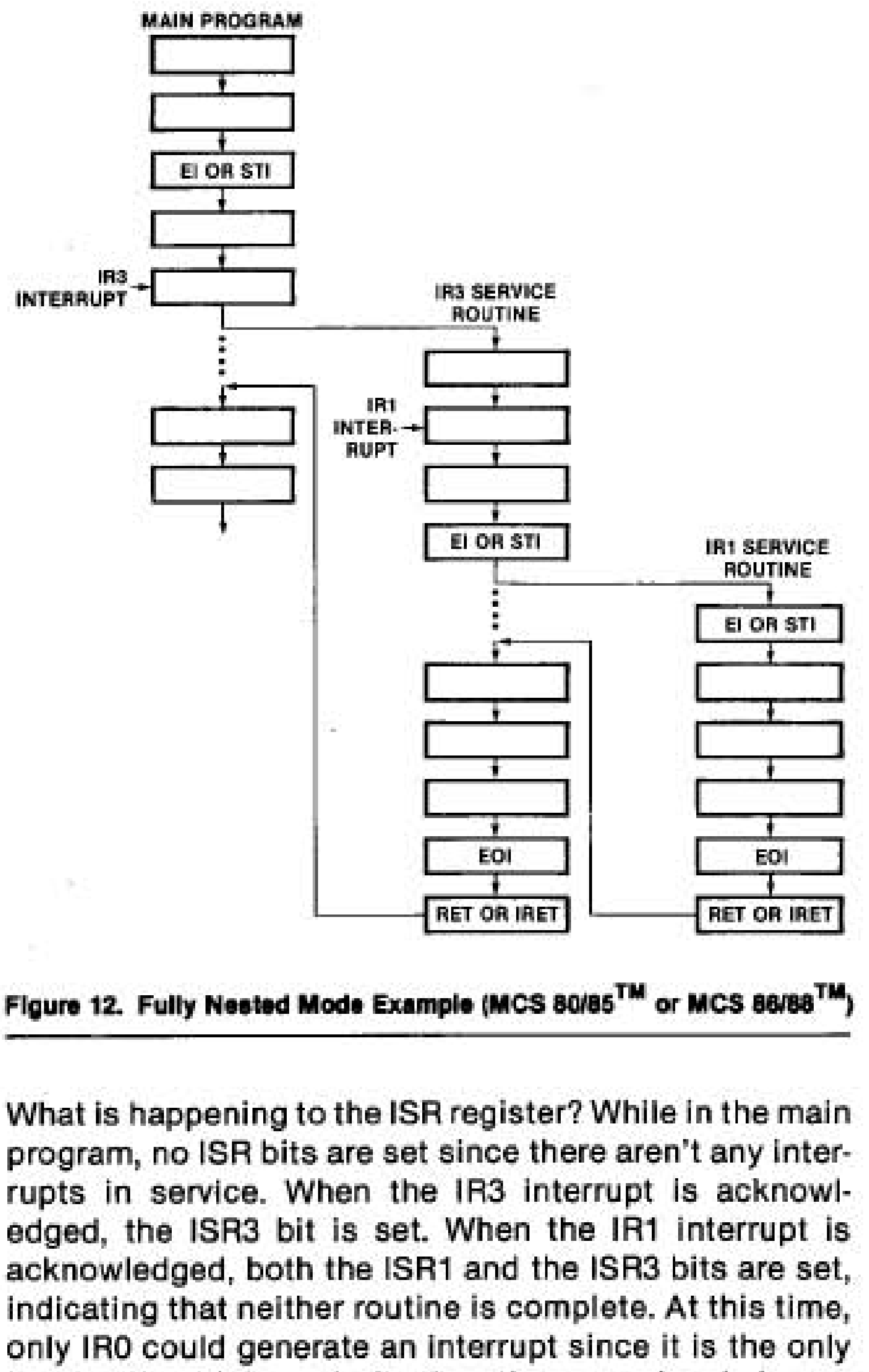


Figure 12. Fully Nested Mode Example (MCS 80/85™ or MCS 86/88™)

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.

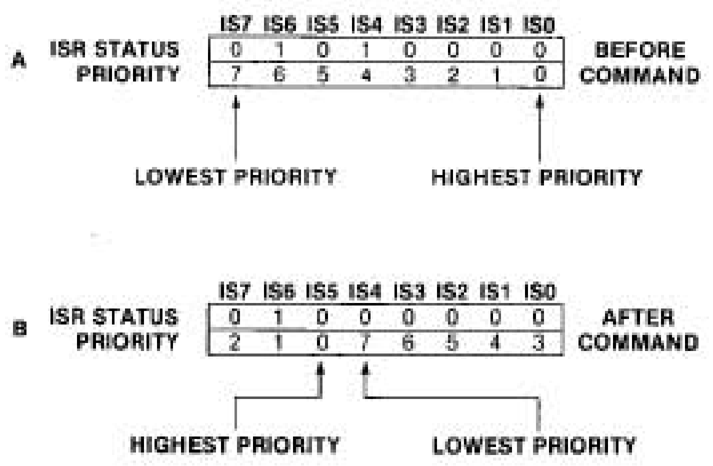


Figure 13. A–B. Rotate on Non-specific EOI Command Example

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.

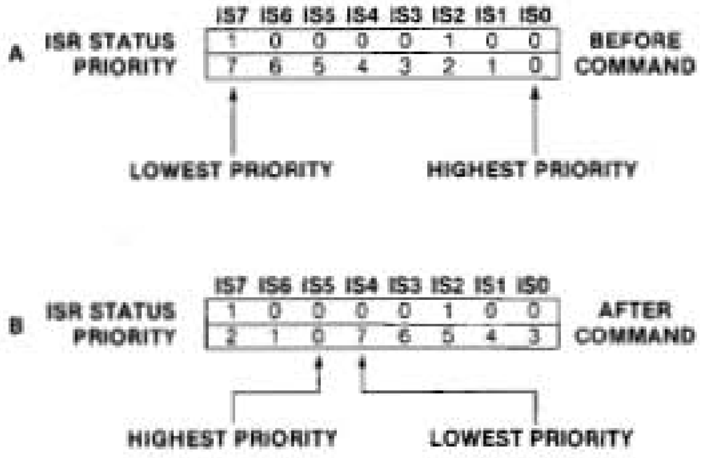


Figure 14. A–B. Set Priority Command Example

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.

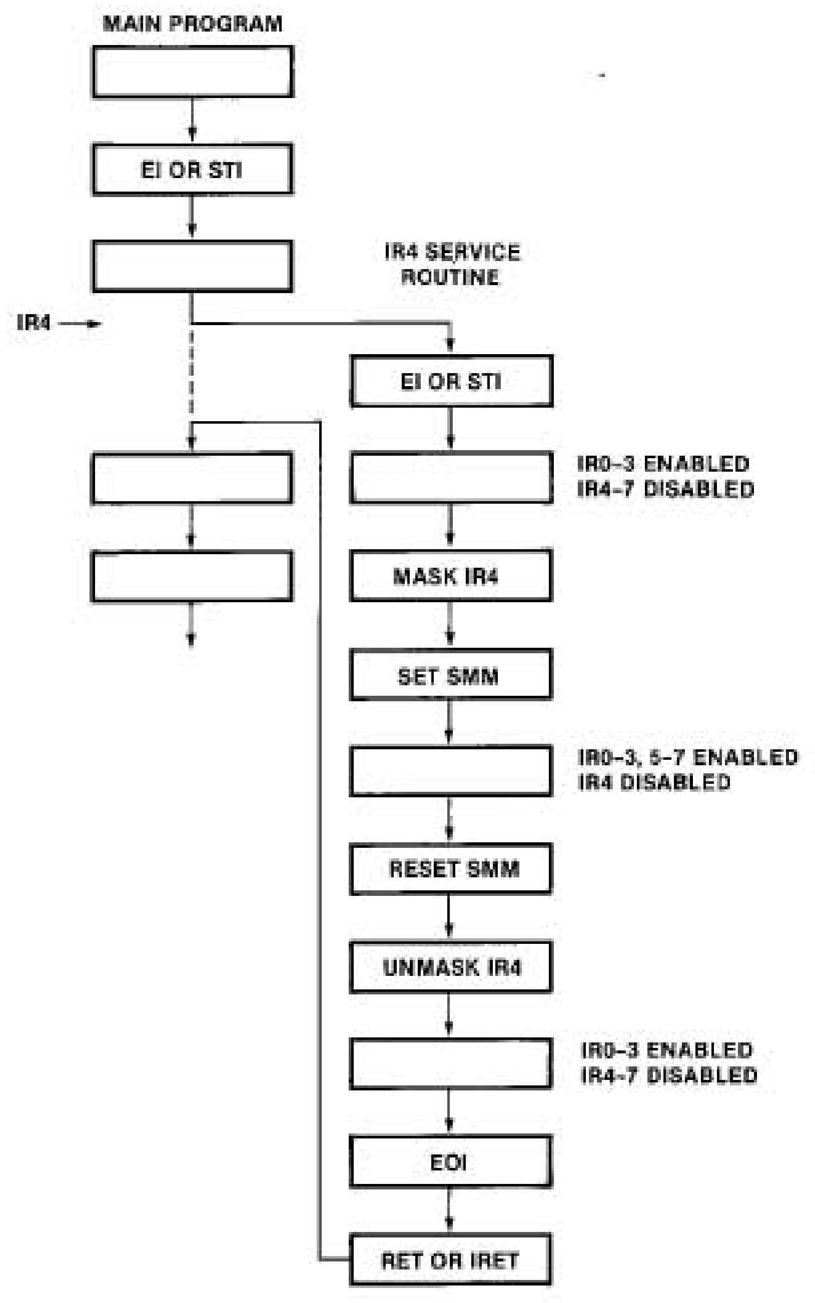


Figure 15. Special Mask Mode Example (MCS 80/85™ or MCS 86/88™)

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 ac-

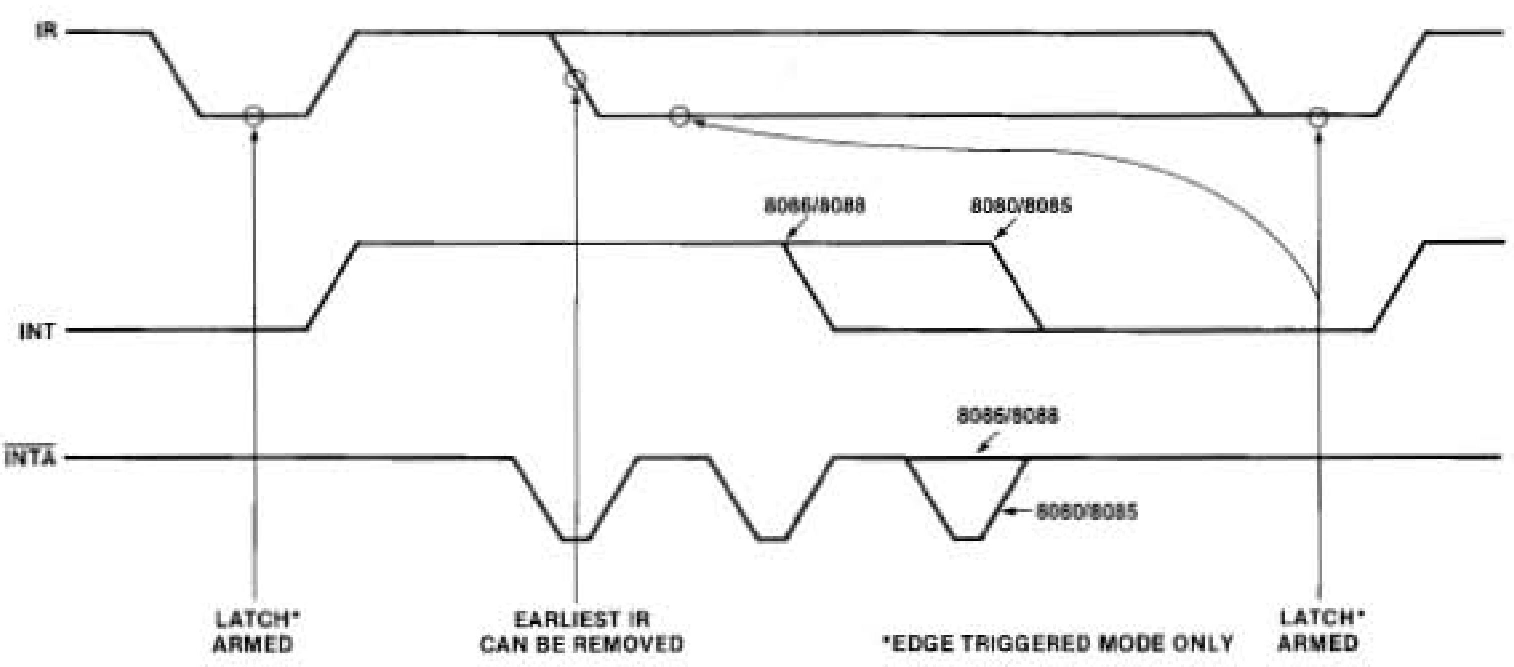


Figure 16. IR Triggering Timing Requirements

tual requesting device has to be determined by the software in the service routine.

Caution should be taken when using the automatic EOI mode and the level triggered mode together. Since in the automatic EOI mode an EOI is automatically performed at the end of the interrupt acknowledge sequence, if the processor enables interrupts while an IR input is still high, an interrupt will occur immediately. To avoid this situation interrupts should be kept disabled until the end of the service routine or until the IR input returns low.

Edge Triggered Mode

When in the edge triggered mode, the 8259A will only recognize interrupts if generated by an inactive (low) to active (high) transition on an IR input. The edge triggered mode incorporates an edge lockout method of operation. This means that after the rising edge of an interrupt request and the acknowledgement of the request, the positive level of the IR input won't generate further interrupts on this level. The user needn't worry about quickly removing the request after acknowledgement in fear of generating further interrupts as might be the case in the level triggered mode. Before another interrupt can be generated the IR input must return to the inactive state.

Referring back to Figure 16, the timing requirements for interrupt triggering is shown. Like the level triggered mode, in the edge triggered mode the request on the IR input must remain active until after the falling edge of the first $\overline{\text{INTA}}$ 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 $\overline{RD}$ pulse to the 8259A (an INput instruction), the contents of the desired register can be acquired. There is no need to issue a read register command every time the IRR or ISR is to be read. Once a read register command is received by the 8259A, it "remembers" which register has been selected. Thus, all that is necessary to read the contents of the same register more than once is the $\overline{RD}$ pulse and the correct addressing (A0 = 0, explained in "Programming the 8259A"). Upon initialization, the selection of registers defaults to the IRR. Some caution should be taken when using the read register command in a system that supports several levels of interrupts. If the higher priority routine causes an interrupt between the read register command and the actual input of the register contents, there's no guarantee that the same register will be selected when it returns. Thus it is best in such cases to disable interrupts during the operation.

Reading the contents of the IMR is different than reading the IRR or ISR. A read register command is not necessary when reading the IMR. This is because the IMR can be addressed directly for both reading and writing. Thus all that the 8259A requires for reading the IMR is a $\overline{RD}$ pulse and the correct addressing (A0 = 1, explained in "Programming the 8259A").

Poll Command

As mentioned towards the beginning of this application note, there are two methods of servicing peripherals: status polling and interrupt servicing. For most applications the interrupt service method is best. This is because it requires the least amount of CPU time, thus increasing system throughput. However, for certain applications, the status poll method may be desirable.

For this reason, the 8259A supports polling operations with the poll command. As opposed to the conventional method of polling, the poll command offers improved device servicing and increased throughput. Rather than having the processor poll each peripheral in order to find the actual device requiring service, the processor polls the 8259A. This allows the use of all the previously mentioned priority modes and commands. Additionally, both polled and interrupt methods can be used within the same program.

To use the poll command the processor must first have its interrupt request pin disabled. Once the poll command is issued, the 8259A will treat the next ($\overline{CS}$ qualified) $\overline{RD}$ pulse issued to it (an INput instruction) as an interrupt 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.

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".

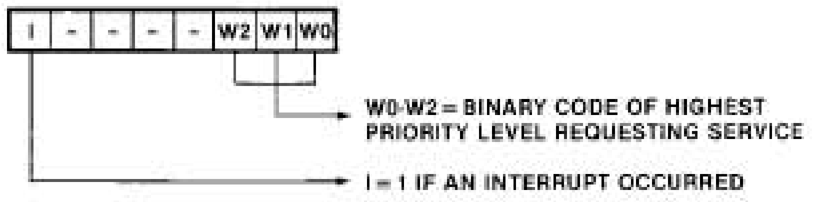

I
-
-
-
-
W2
W1
W0
W0-W2 = BINARY CODE OF HIGHEST PRIORITY LEVEL REQUESTING SERVICE
I = 1 IF AN INTERRUPT OCCURRED

Figure 17. Poll Word

3.5 INTERRUPT CASCADING

As mentioned earlier, more than one 8259A can be used to expand the priority interrupt scheme to up to 64 levels without additional hardware. This method for expanded interrupt capability is called "cascading". The 8259A supports cascading operations with the cascade mode. Additionally, the special fully nested mode and the buffered mode are available for increased flexibility when cascading 8259A's in certain applications.

Cascade Mode

When programmed in the cascade mode, basic operation consists of one 8259A acting as a master to the others which are serving as slaves. Figure 18 shows a system containing a master and two slaves, providing a total of 22 interrupt levels.

A specific hardware set-up is required to establish operation in the cascade mode. With Figure 18 as a reference, note that the master is designated by a high on the $\overline{SP}/\overline{EN}$ 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).

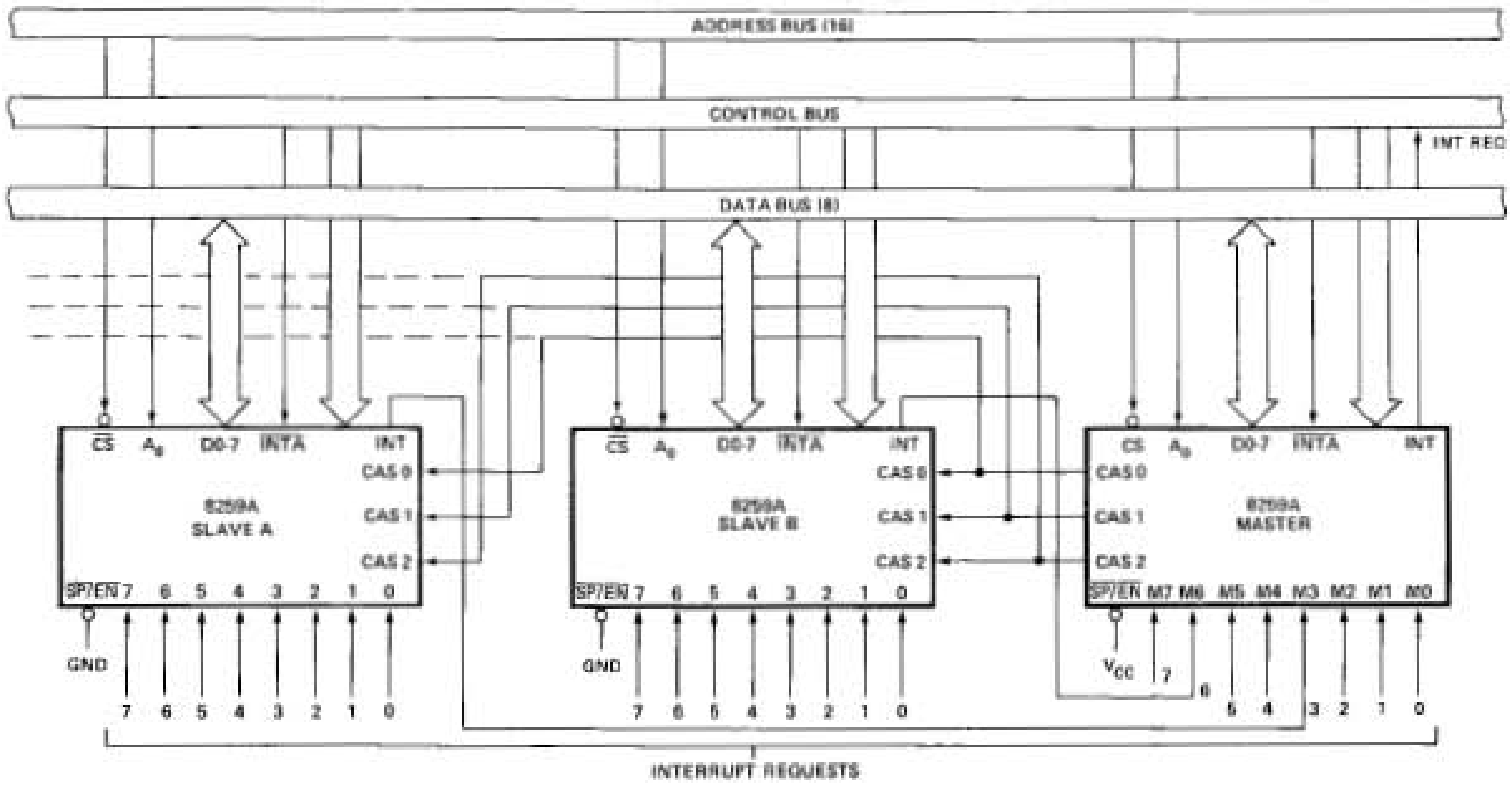


Figure 18. Cascaded 8259A'S 22 Interrupt Levels

Besides hardware set-up requirements, all 8259A's must be software programmed to work in the cascade mode. Programming the cascade mode is done during the initialization of each 8259A. The 8259A that is selected as master must receive specification during its initialization as to which of its IR inputs are connected to a slave's INT pin. Each slave 8259A, on the other hand, must be designated during its initialization with an ID (0 through 7) corresponding to which of the master's IR inputs its INT pin is connected to. This is all necessary so the CAS0-2 pins of the masters will be able to address each individual slave. Note that as in normal operation, each 8259A must also be initialized to give its IR 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 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.

Buffered Mode

The buffered mode is useful in large systems where buffering is required on the data bus. Although not limited to only 8259A cascading, it's most pertinent in this use. In the buffered mode, whenever the 8259A's data bus output is enabled, its $\overline{\text{SP}}/\overline{\text{EN}}$ pin will go low. This signal can be used to enable data transfer through a buffer transceiver in the required direction.

Figure 19 shows a conceptual diagram of three 8259A's in cascade, each slave is controlling an individual 8286 8-bit bidirectional bus driver by means of the buffered mode. Note the pull-up on the $\overline{\text{SP}}/\overline{\text{EN}}$. It is used to enable data transfer to the 8259A for its initial programming. When data transfer is to go from the 8259A to the processor, $\overline{\text{SP}}/\overline{\text{EN}}$ will go low; otherwise, it will be high.

A question should arise, however, from the fact that the $\overline{\text{SP}}/\overline{\text{EN}}$ pin is used to designate a master from a slave; how can it be used for both master-slave selection and buffer control? The answer to this is the provision for software programmable master-slave selection when in the buffer mode. The buffered mode is selected during each 8259A's initialization. At the same time, the user can assign each individual 8259A as a master or slave (see "Programming the 8259A").

4. PROGRAMMING THE 8259A

Programming the 8259A is accomplished by using two types of command words: Initialization Command Words (ICWs) and Operational Command Words (OCWs). All the modes and commands explained in the previous section, "Operation of the 8259A", are programmable using the ICWs and OCWs (see Appendix A for cross reference). The ICWs are issued from the processor in a sequential format and are used to set-up the 8259A in an initial state of operation. The OCWs are issued as needed to vary and control 8259A operation.

Both ICWs and OCWs are sent by the processor to the 8259A via the data bus (8259A $\overline{\text{CS}}=0$, $\overline{\text{WR}}=0$). The 8259A distinguishes between the different ICWs and OCWs by the state of its A0 pin (controlled by processor addressing), the sequence they're issued in (ICWs only), and some dedicated bits among the ICWs and OCWs. Those bits which are dedicated are indicated so by fixed values (0 or 1) in the corresponding ICW or OCW programming formats which are covered shortly. Note, when issuing either ICWs or OCWs, the interrupt request pin of the processor should be disabled.

4.1 INITIALIZATION COMMAND WORDS (ICWs)

Before normal operation can begin, each 8259A in a system must be initialized by a sequence of two to four programming bytes called ICWs (Initialization Command Words). The ICWs are used to set-up the necessary conditions and modes for proper 8259A operation.

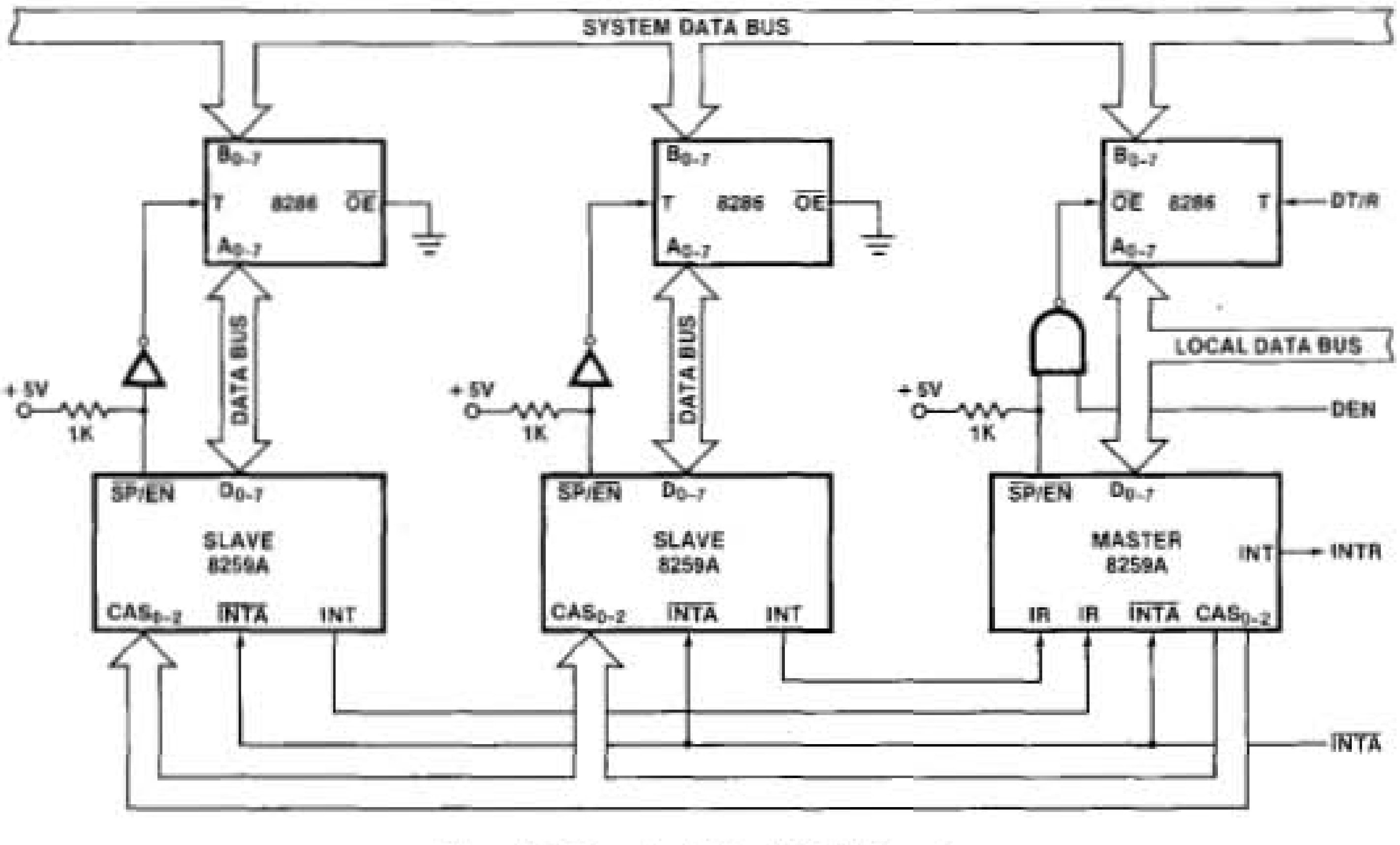


Figure 19. Cascade-Buffered Mode Example

Figure 20 shows the initialization flow of the 8259A. Both ICW1 and ICW2 must be issued for any form of 8259A operation. However, ICW3 and ICW4 are used only if designated so in ICW1. Determining the necessity and use of each ICW is covered shortly in individual groupings. Note that, once 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 remaining ICWs as designated in ICW1.

B. The ISR (In-Service Register) and IMR (Interrupt Mask Register) are both cleared.

C. The special mask mode is reset.

D. The rotate in automatic EOI mode flip-flop is cleared.

E. The IRR (Interrupt Request Register) is selected for the read register command.

F. If the IC4 bit equals 0 in ICW1, all functions in ICW4 are cleared; 8080/8085 mode is selected by default.

G. The fully nested mode is entered with an initial priority assignment of IR0 highest through IR7 lowest.

H. The edge sense latch of each IR priority cell is cleared, thus requiring a low to high transition to generate an interrupt (edge triggered mode effected only).

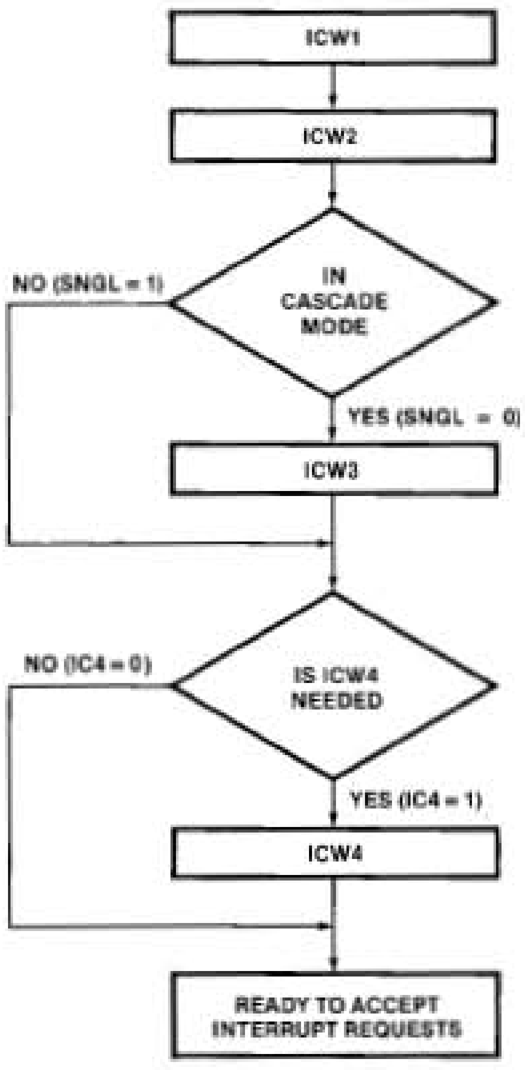
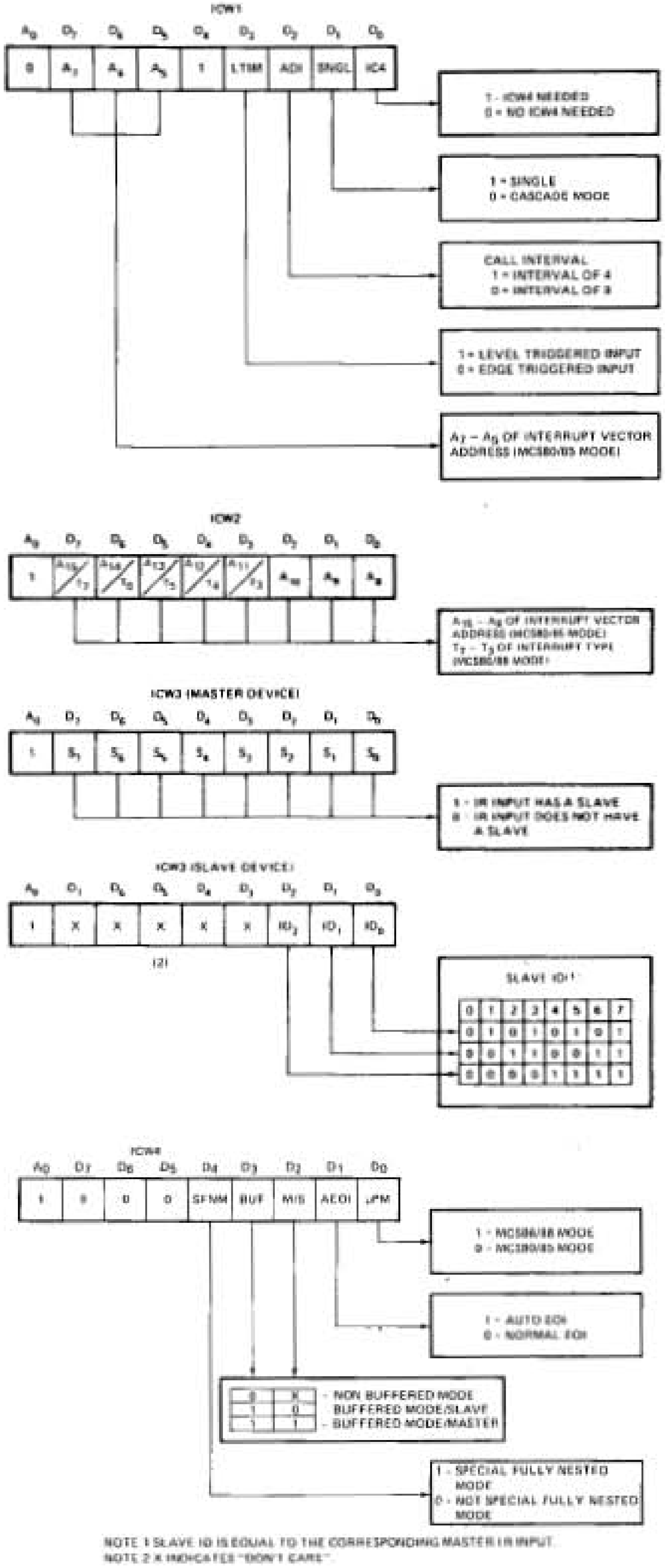


Figure 20. Initialization Flow

The ICW programming format, Figure 21, shows bit designation and a short definition of each ICW. With the ICW format as reference, the functions of each ICW will now be explained individually.


ICW1
A0 D7 D6 D5 D4 D3 D2 D1 D0
0 A7 A6 A5 1 LTIM ADI SNGL IC4
1 - ICW4 NEEDED
0 = NO ICW4 NEEDED
1 = SINGLE
0 = CASCADE MODE
CALL INTERVAL
1 = INTERVAL OF 4
0 = INTERVAL OF 8
1 = LEVEL TRIGGERED INPUT
0 = EDGE TRIGGERED INPUT
A7 – A5 OF INTERRUPT VECTOR ADDRESS (MCS80/85 MODE)
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 (MCS86/88 MODE)
ICW3 (MASTER DEVICE)
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 S7 S6 S5 S4 S3 S2 S1 S0
1 - IR INPUT HAS A SLAVE
0 - IR INPUT DOES NOT HAVE A SLAVE
ICW3 (SLAVE DEVICE)
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 X X X X X ID2 ID1 ID0
(2)
SLAVE ID(1)
0 1 2 3 4 5 6 7
0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1
ICW4
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 0 0 0 SFNM BUF M/S AEOI µPM
1 - MCS86/88 MODE
0 - MCS80/85 MODE
1 - AUTO EOI
0 - NORMAL EOI
0 X - NON BUFFERED MODE
1 0 - BUFFERED MODE/SLAVE
1 1 - BUFFERED MODE/MASTER
1 - SPECIAL FULLY NESTED MODE
0 - NOT SPECIAL FULLY NESTED MODE
NOTE 1 SLAVE ID IS EQUAL TO THE CORRESPONDING MASTER IR INPUT.
NOTE 2 X INDICATES "DON'T CARE".

SOME OF THE TERMINOLOGY USED MAY DIFFER SLIGHTLY FROM EXISTING 8259A DATA SHEETS. THIS IS DONE TO BETTER CLARIFY AND EXPLAIN THE PROGRAMMING OF THE 8259A, THE OPERATIONAL RESULTS REMAIN THE SAME.

Figure 21. Initialization Command Words (ICWS) Programming Format

ICW1 and ICW2

Issuing ICW1 and ICW2 is the minimum amount of programming needed for any type of 8259A operation. The majority of bits within these two ICWs are used to designate the interrupt vector starting address. The remaining bits serve various purposes. Description of the ICW1 and ICW2 bits is as follows:

IC4: The IC4 bit is used to designate to the 8259A whether or not ICW4 will be issued. If any of the ICW4 operations are to be used, ICW4 must equal 1. If they aren't used, then ICW4 needn't be issued and IC4 can equal 0. Note that if IC4 = 0, the 8259A will assume operation in the MCS-80/85 mode.

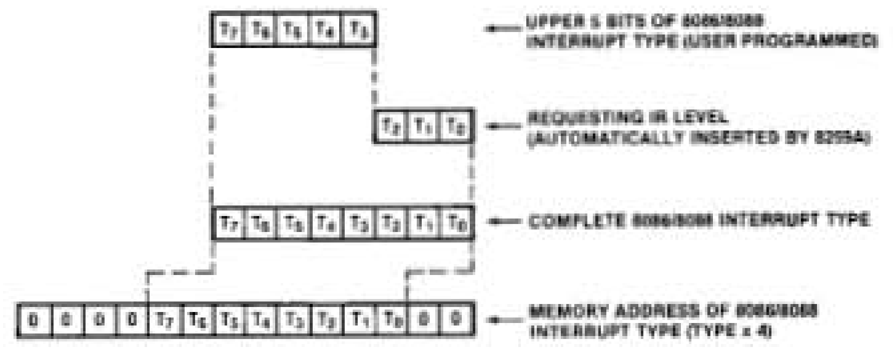
SNGL: The SNGL bit is used to designate whether or not the 8259A is to be used alone or in the cascade mode. If the cascade mode is desired, SNGL must equal 0. In doing this, the 8259A will accept ICW3 for further cascade mode programming. If the 8259A is to be used as the single 8259A within a system, the SNGL bit must equal 1; ICW3 won't be accepted.

ADI: The ADI bit is used to specify the address interval for the MCS-80/85 mode. If a 4-byte address interval is to be used, ADI must equal 1. For an 8-byte address interval, ADI must equal 0. The state of ADI is ignored when the 8259A is in the MCS-86/88 mode.

LTIM: The LTIM bit is used to select between the two IR input triggering modes. If LTIM = 1, the level triggered mode is selected. If LTIM = 0, the edge triggered mode is selected.

A5–A15: The A5–A15 bits are used to select the interrupt vector address when in the MCS-80/85 mode. There are two programming formats that can be used to do this. Which one is implemented depends upon the selected address interval (ADI). If ADI is set for the 4-byte interval, then the 8259A will automatically insert A0–A4 (A0, A1 = 0 and A2, A3, A4 = IR0–7). Thus A5–A15 must be user selected by programming the A5–A15 bits with the desired address. If ADI is set for the 8-byte interval, then A0–A5 are automatically inserted (A0, A1, A2 = 0 and A3, A4, A5 = IR0–7). This leaves A6–A15 to be selected by programming the A6–A15 bits with the desired address. The state of bit 5 is ignored in the latter format.

T3–T7: The T3–T7 bits are used to select the interrupt type when the MCS-86/88 mode is used. The programming of T3–T7 selects the upper 5 bits. The lower 3 bits are automatically inserted, corresponding to the IR level causing the interrupt. The state of bits A5–A10 will be ignored when in the MCS-86/88 mode. Establishing the actual memory address of the interrupt is shown in Figure 22.



T_7 T_6 T_5 T_4 T_3
UPPER 5 BITS OF 8086/8088 INTERRUPT TYPE (USER PROGRAMMED)
T_2 T_1 T_0
REQUESTING IR LEVEL (AUTOMATICALLY INSERTED BY 8259A)
T_7 T_6 T_5 T_4 T_3 T_2 T_1 T_0
COMPLETE 8086/8088 INTERRUPT TYPE
0 0 0 0 T_7 T_6 T_5 T_4 T_3 T_2 T_1 T_0 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

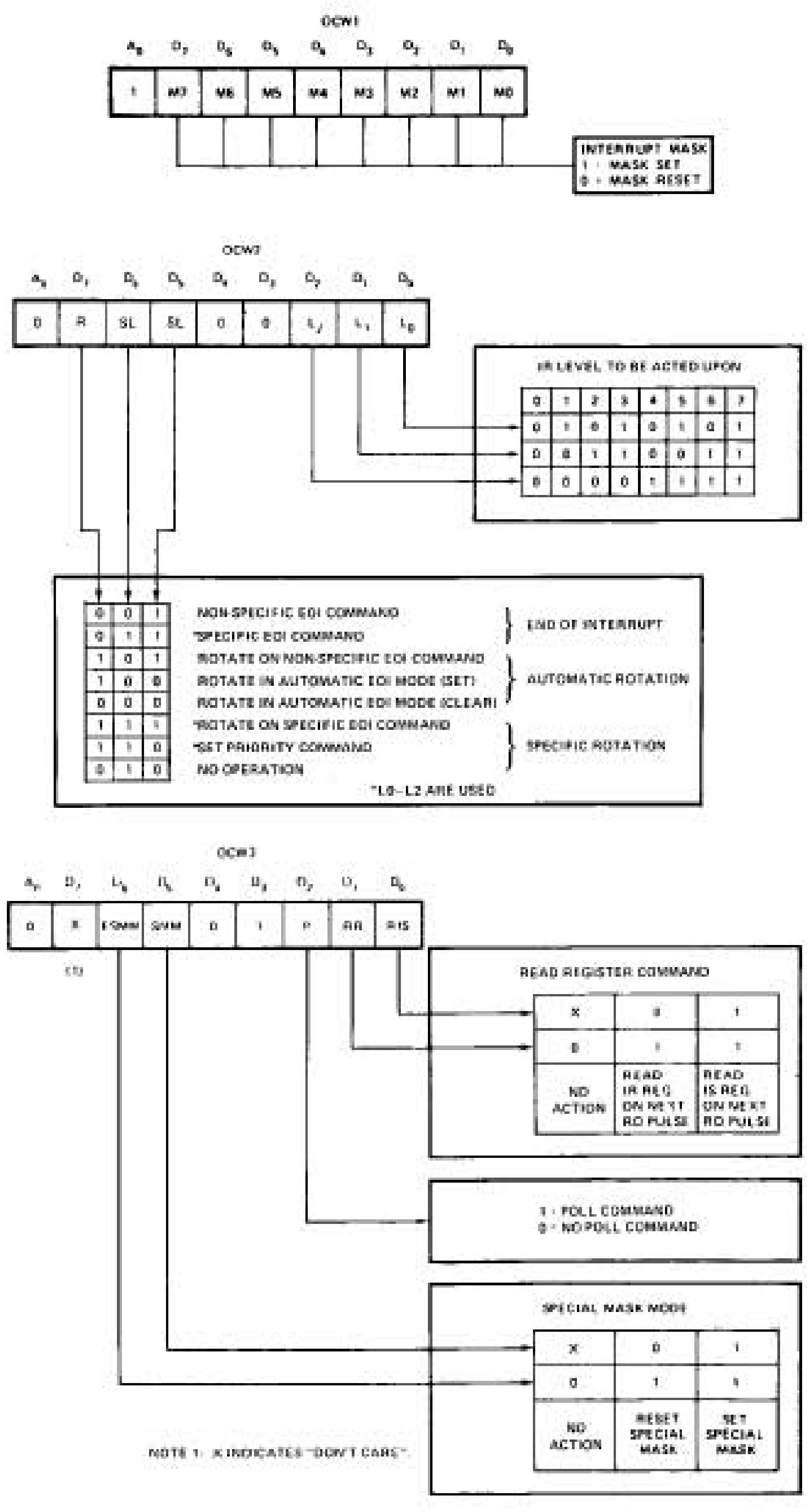
OCW1 is used solely for 8259A masking operations. It provides a direct link to the IMR (Interrupt Mask Register). The processor can write to or read from the IMR via OCW1. The OCW1 bit definition is as follows:

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.


OCW1
A_0 D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
1 M7 M6 M5 M4 M3 M2 M1 M0
INTERRUPT MASK
1 = MASK SET
0 = MASK RESET
OCW2
0 R SL EOI 0 0 L_2 L_1 L_0
IR LEVEL TO BE ACTED UPON
NON-SPECIFIC EOI COMMAND
*SPECIFIC EOI COMMAND
ROTATE ON NON-SPECIFIC EOI COMMAND
ROTATE IN AUTOMATIC EOI MODE (SET)
ROTATE IN AUTOMATIC EOI MODE (CLEAR)
*ROTATE ON SPECIFIC EOI COMMAND
*SET PRIORITY COMMAND
NO OPERATION
END OF INTERRUPT
AUTOMATIC ROTATION
SPECIFIC ROTATION
*L0–L2 ARE USED
OCW3
0 X ESMM SMM 0 1 P RR RIS
(1)
READ REGISTER COMMAND
NO ACTION
READ IR REG ON NEXT RD PULSE
READ IS REG ON NEXT RD PULSE
1 = POLL COMMAND
0 = NO POLL COMMAND
SPECIAL MASK MODE
NO ACTION
RESET SPECIAL MASK
SET SPECIAL MASK
NOTE 1: X INDICATES "DON'T CARE".

SOME OF THE TERMINOLOGY USED MAY DIFFER SLIGHTLY FROM EXISTING 8259A DATA SHEETS. THIS IS DONE TO BETTER CLARIFY AND EXPLAIN THE PROGRAMMING OF THE 8259A. THE OPERATIONAL RESULTS REMAIN THE SAME.

Figure 23. Operational Command Words (OCWs) Programming Format

EOI: The EOI bit is used for all end of interrupt commands (not automatic end of interrupt mode). If set to a 1, a form of an end of interrupt command will be executed depending on the state of the SL and R bits. If EOI is 0, an end of interrupt command won't be executed.

SL: The SL bit is used to select a specific level for a given operation. If SL is set to a 1, the L0–L2 bits are enabled. The operation selected by the EOI and R bits will be executed on the specified interrupt level. If SL is 0, the L0–L2 bits are disabled.

R: The R bit is used to control all 8259A rotation operations. If the R bit is set to a 1, a form of priority rotation will be executed depending on the state of SL and EOI bits. If R is 0, rotation won't be executed.

OCW3

OCW3 is used to issue various modes and commands to the 8259A. There are two main categories of operation associated with OCW3: interrupt status and interrupt masking. Bit definition of OCW3 is as follows:

RIS:	The RIS bit is used to select the ISR or IRR for the read register command. If RIS is set to 1, ISR is selected. If RIS is 0, IRR is selected. The state of the RIS is only honored if the RR bit is a 1.
RR:	The RR bit is used to execute the read register command. If RR is set to a 1, the read register command is issued and the state of RIS determines the register to be read. If RR is 0, the read register command isn't issued.
P:	The P bit is used to issue the poll command. If P is set to a 1, the poll command is issued. If it is 0, the poll command isn't issued. The poll command will override a read register command if set simultaneously.
SMM:	The SMM bit is used to set the special mask mode. If SMM is set to a 1, the special mask mode is selected. If it is 0, it is not selected. The state of the SMM bit is only honored if it is enabled by the ESMM bit.
ESMM:	The ESMM bit is used to enable or disable the effect of the SMM bit. If ESMM is set to a 1, SMM is enabled. If ESMM is 0, SMM is disabled. This bit is useful to prevent interference of mode and command selections in OCW3.

5. APPLICATION EXAMPLES

In this section, the 8259A is shown in three different application examples. The first is an actual design implementation supporting an 8080A microprocessor system, "Power Fail/Auto Start with Battery Back-Up RAM". The second is a conceptual example of incorporating more than 64 interrupt levels in an 8080A or 8085A system, "78 Level Interrupt System". The third application is a conceptual design using an 8086 system, "Timer Controlled Interrupts". Although specific microprocessor systems are used in each example, these applications can be applied to either MCS-80, MCS-85, MCS-86, or MCS-88 systems, providing the necessary hardware and software changes are made. Overall, these applications should serve as a useful guide, illustrating the various procedures in using the 8259A.

5.1 POWER FAIL/AUTO-START WITH BATTERY BACK-UP RAM

The first application illustrates the 8259A used in an 8080A system, supporting a battery back-up scheme for the RAM (Random Access Memory) in a microcomputer system. Such a scheme is important in numerical and process control applications. The entire microcomputer system could be supported by a battery back-up scheme, however, due to the large amount of current usually required and the fact that most machinery is not supported by an auxiliary power source, only the state of calculations and variables usually need to be saved. In the event of a loss of power, if these items are not already stored in RAM, they can be transferred there and saved using a simple battery back-up system.

The vehicle used in this application is the Intel® SBC-80/20 Single Board Computer. An 8259A is used in the SBC-80/20 along with control lines helpful in implementing the power-down and automatic restart sequence used in a battery back-up system. The SBC-80/20 also contains user-selectable jumpers which allow the on-board RAM to be powered by a supply separate from the supply used for the non-RAM components. Also, the output of an undedicated latch is available to be connected to the IR inputs of the 8259A (the latch is cleared via an output port). In addition, an undedicated, buffered input line is provided, along with an input to the RAM decoder that will protect memory when asserted.

The additional circuitry to be described was constructed on an SBC-905 prototyping board. An SBC-635 power supply was used to power the non-RAM section of the SBC-80/20 while an external DC supply was used to simulate the back-up battery supplying power to the RAM. The SBC-635 was used since it provides an open collector ACLO output which indicates that the AC input line voltage is below 103/206 VAC (RMS).

The following is an example of a power-down and restart sequence that introduces the various power fail signals.

1. An AC power failure occurs and the ACLO goes high (ACLO is pulled up by the battery supply). This indicates that DC power will be reliable for at most 7.5 ms. The power fail circutry generates a Power Fail Interrupt ($\overline{\text{PFI}}$) signal. This signal sets the $\overline{\text{PFI}}$ latch, which is connected to the IR0 input of the 8259A, and sets the Power Fail Sense (PFS) latch. The state of this latch will indicate to the processor, upon reset, whether it is coming up from a power failure (warm start) or if it is coming up initially (cold start).
2. The processor is interrupted by the 8259A when the PFI latch is set. This pushes the pre-power-down program counter onto the stack and calls the service routine for the IR0 input. The IR0 service routine saves the processor status and any other needed variables. The routine should end with a HALT instruction to minimize bus transitions.
3. After a predetermined length of time (5 ms in this example) the power fail circuitry generates a Memory Protect ($\overline{\text{MPRO}}$) signal. All processing for the power failure (including the interrupt response delays) must be completed within this 5 ms window. The $\overline{\text{MPRO}}$ signal ensures that spurious transitions on the system control bus caused by power going down do not alter the contents of the RAM.
4. DC power goes down.
5. AC power returns. The power-on reset circuitry on the SBC-80/20 generates a system RESET.
6. The processor reads the state of the $\overline{\text{PFS}}$ line to determine the appropriate start-up sequence. The PFS latch is cleared, the MPRO signal is removed, and the PFI latch driving IR0 is cleared by the Power Fail Sense Reset ($\overline{\text{PFSR}}$) signal. The system then continues from the pre-power-down location for a warm start by restoring the processor status and popping the pre-power-down program counter off the stack.

Figure 24 illustrates this timing.

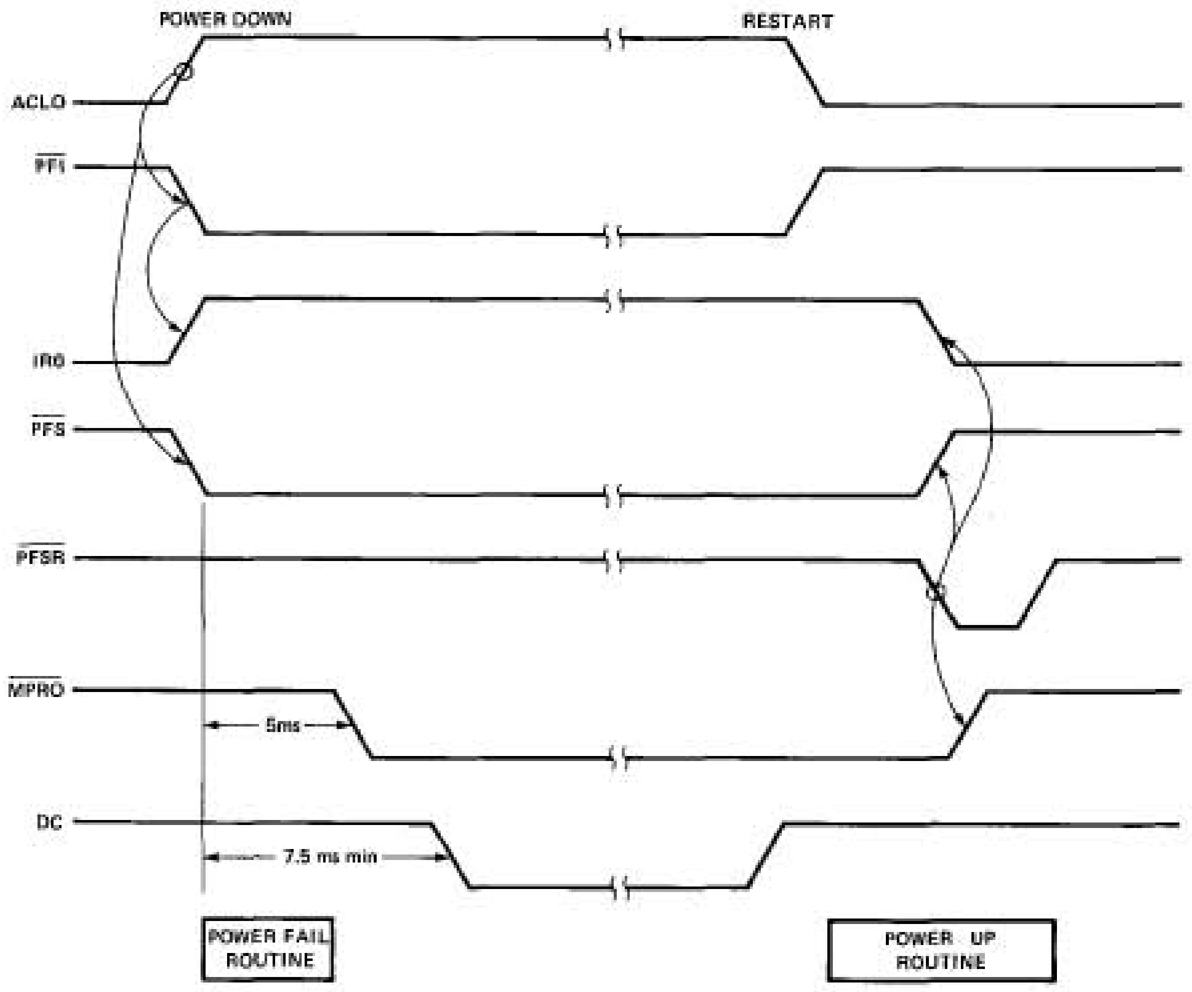


Figure 24. Power Down Restart Timing

Figure 25 shows the block diagram for the system. Notice that the RAM, the RAM decoder, and the power-down circuitry are powered by the battery supply.

The schematic of the power-down circuitry and the SBC-80/20 interface is shown in Figure 26. The design is very straightforward and uses CMOS logic to minimize the battery current requirements. The cold start switch is necessary to ensure that during a cold start, the $\overline{PFS}$ line is indicating "cold start" sense ($\overline{PFS}$ high). Thus, for a cold start, the cold start switch is depressed during power on. After that, no further action is needed. Notice that the $\overline{PFI}$ signal sets the on-board PFI latch. The output of this latch drives the 8259A IR0 input. This latch is cleared during the restart routine by executing an OUTput D4H instruction. The state of the $\overline{PFS}$ line may be read on the least significant data bus line (DB0) by executing an INput D4H instruction. An 8255 port (8255 #1, port C, bit 0) is used to control the $\overline{PFSR}$ line.

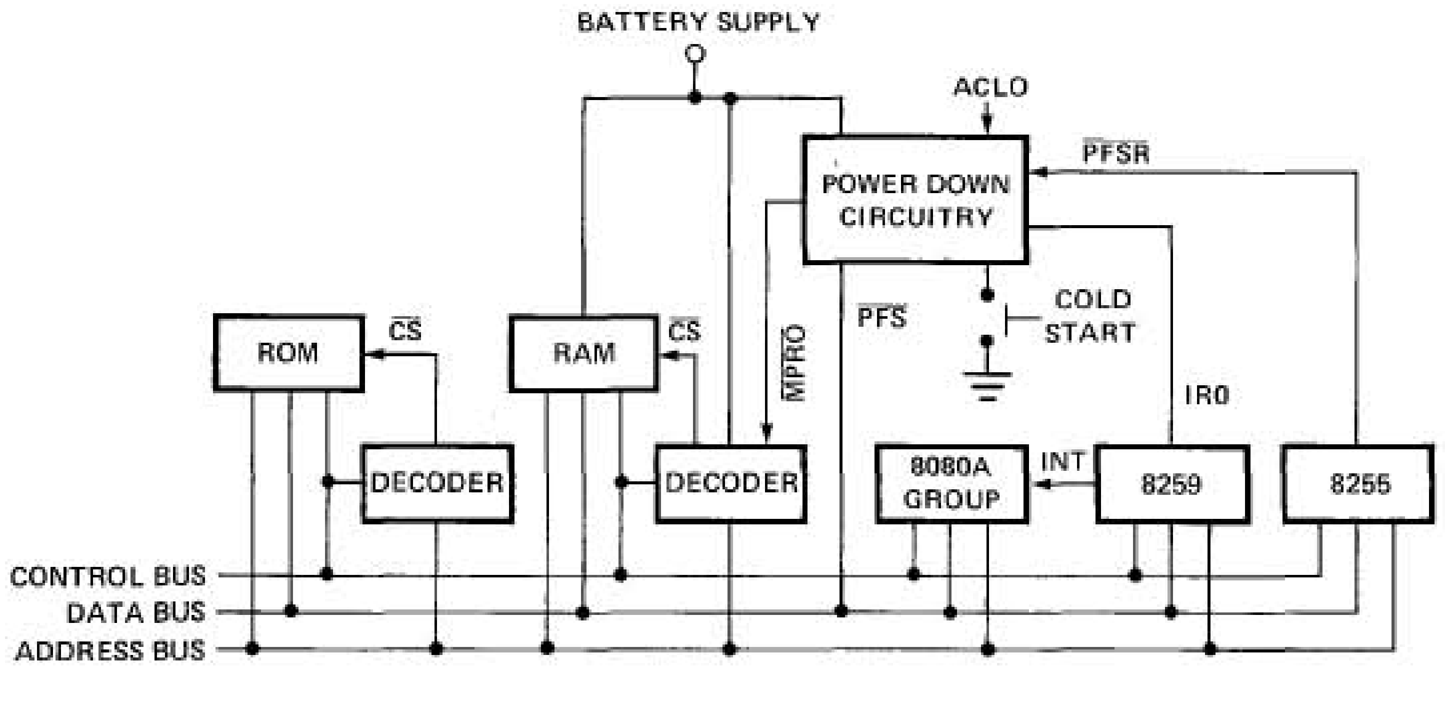


Figure 25. Block Diagram of SBC 80/20 with Power Down Circuit

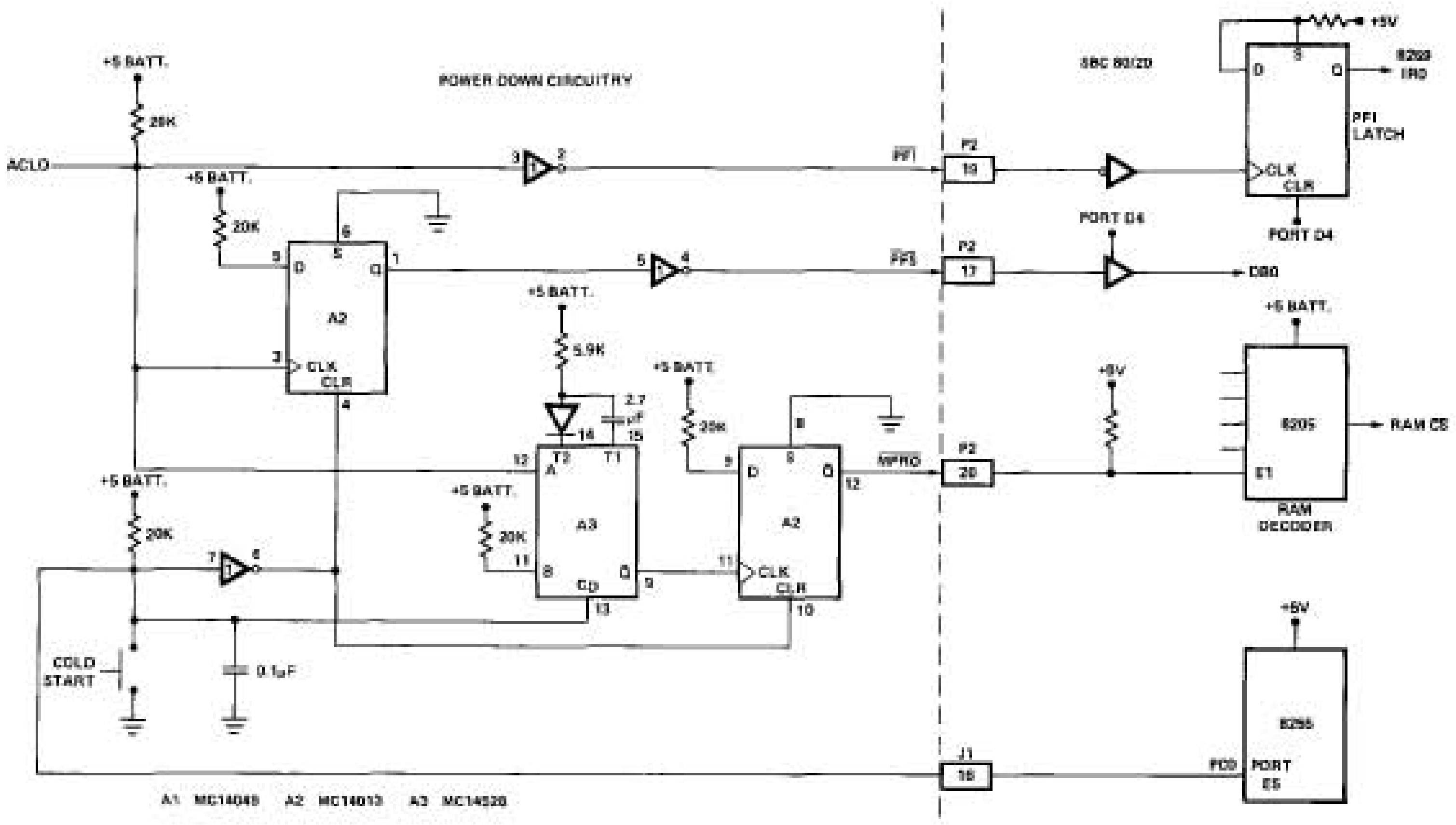


Figure 26. Power Down Circuit - SBC 80/20 Interface

The fully nested mode for the 8259A is used in its initial state to ensure the IR0 always has the highest priority. The remaining IR inputs can be used for any other purpose in the system. The only constraint is that the service routines must enable interrupts as early as possible. Obviously, this is to ensure that the power-down interrupt does not have to wait for service. If a rotating priority scheme is desired, another 8259A could be added as a slave and be programmed to operate in a rotating mode. The master would remain in the initial state of the fully nested mode so that the IR0 still remains the highest priority input.

The software to support the power-down circuitry is shown in Figure 27. The flow for each label will be discussed.

After any system reset, the processor starts execution at location 0000H (START). The $\overline{\text{PFS}}$ status is read and execution is transferred to CSTART if $\overline{\text{PFS}}$ indicates a cold start (i.e., someone is depressing the cold start switch) or WSTART if a warm start is indicated ($\overline{\text{PFS}}$ LOW). CSTART is the start of the user's program. The Stack Pointers (SP) and device initialization were included just to remind the reader that these must occur. The first EI instruction must appear after the 8259A has received its initialization sequence. The 8259A (and other devices) are initialized in the INIT subroutine.

When a power failure occurs, execution is vectored by the 8259A to REGSAV by way of the jump table at JSTART. The pre-power-down program counter is placed on the stack. REGSAV saves the processor registers and flags in the usual manner by pushing them onto the stack. Other items, such as output port status, programmable peripheral states, etc., are pushed onto the stack at this time. The Stack Pointer (SP) could be pushed onto the stack by way of the register pair HL but the top of the stack can exist anywhere in memory and there is no way then of knowing where that is when in the power-up routine. Thus, the SP is saved at a dedicated location in RAM. It isn't really necessary to send an EOI command to the 8259A in REGSAV since power will be removed from the 8259A, but one is included for completeness. The final instruction before actually losing power is a HALT. This minimizes somewhat spurious transitions on the various busses and lets the processor die gracefully.

On reset, when a warm start is detected, execution is transferred to WSTART. WSTART activates $\overline{\text{PFSR}}$ by way of the 8255 (all outputs go low then the 8255 is initialized). In the power-down circuitry, $\overline{\text{PFSR}}$ clears the PFS latch and removes the $\overline{\text{MPRO}}$ signal which then allows access to the RAM. WSTART also clears the PFI latch which arms the 8259A IR0 input. Then the 8259A is re-initialized along with any other devices. The SP is retrieved from RAM and the processor registers and flags are restored by popping them off the stack. Interrupts are then enabled. Now the power-down program counter is on top of the stack, so executing a RETurn instruction transfers the processor to exactly where it left off before the power failure.

Aside from illustrating the usefulness of the 8259A (and the SBC-80/20) in implementing a power failure protected microcomputer system, this application should also point out a way of preserving the processor status when using interrupts.

```
LOC  OBJ         SEQ         SOURCE STATEMENT

                   0 ;
                   1 ;
                   2 ;POWER DOWN AND RESTART FOR THE SBC 80/20
                   3 ;
                   4 ;
                   5 ;SYSTEM EQUATES
00DA               6 PT59A    EQU    0DAH           ;8259 PORT WITH A0=0
00DB               7 PT59B    EQU    0DBH           ;8259 PORT WITH A0=1
00E7               8 PPI1CT   EQU    0E7H           ;8255 #1 CONTROL PORT
00E6               9 PPI1C    EQU    0E6H           ;8255 #1 PORT C
3F00              10 SPSAVE   EQU    3F00H          ;SP STORAGE IN RAM
0001              11 JPT      EQU    01H            ;MSB OF 8259 JUMP TABLE
                  12 ;
                  13 ;
                  14 ;STARTING POINT AFTER SYSTEM RESET
                  15 ;
                  16 ;
0000              17          ORG    0H
0000 DBD4         18 START:   IN     0D4H           ;READ PFS/ STATUS
0002 1F           19          RAR                   ;PFS/ ON D0, PUT IN CARRY
0003 DA2001       20          JC     CSTART         ;PFS/=1, THEN COLD START
                  21 ;
                  22 ;
                  23 ;RESTART LOCATION. PFS/=0, THEN WARM START
                  24 ;
                  25 ;
0006 3E80         26 WSTART:  MVI    A,80H          ;SET 8255 #1 TO OUTPUT MODE
0008 D3E7         27          OUT    PPI1CT         ;8255 CONTROL PORT. PFSR/ GOES LOW
                  28          ;
                  29          ;OUTPUT COMMAND MAKES PFSR/ GO LOW WHICH REMOVES MPRO/ AND
                  30          ;CLEARS PFS LATCH
                  31          ;
000A 3E01         32          MVI    A,01H          ;RETURN PFSR/ HIGH
000C D3E6         33          OUT    PPI1C          ;8255 #1 PORT C
000E D3D4         34          OUT    0D4H           ;RESET PFI LATCH
0010 CD1D00       35          CALL   INIT           ;GO INITIALIZE EVERYTHING
0013 2A003F       36          LHLD   SPSAVE         ;RETRIEVE SP FROM RAM
0016 F9           37          SPHL                  ;PUT BACK INTO SP
0017 C1           38          POP    B              ;RESTORE BC
0018 D1           39          POP    D              ;RESTORE DE
0019 E1           40          POP    H              ;RESTORE HL
001A F1           41          POP    PSW            ;RESTORE A PLUS FLAGS
001B FB           42          EI                    ;ENABLE INTERRUPTS
001C C9           43          RET                   ;PC-POWER-DOWN PC ON TOP OF STACK
                  44                                ;RETURN TO IT
                  45 ;
                  46 ;
                  47 ;INITIALIZATION ROUTINE. AT LEAST DO 8259 BUT OTHERS CAN BE ADDED
                  48 ;
                  49 ;
001D 3E16         50 INIT:    MVI    A,16H          ;F=1,S=1,A7-A5=0  ICW1
001F D3DA         51          OUT    PT59A          ;8259 PORT WITH A0=0
0021 3E01         52          MVI    A,JPT          ;MSB OF JUMP TABLE ICW2
0023 D3DB         53          OUT    PT59B          ;8259 PORT WITH A0=1
                  54
                  55          ;ADD ANY OTHER INITIALIZATIONS HERE
                  56          ;
0025 C9           57          RET                   ;RETURN
                 -58 ;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                  59 ;
                  60 ;POWER DOWN ROUTINE TO SAVE REGISTERS AND STATUS
                  61 ;
                  62 ;
0026 F5           63 PWDOWN:  PUSH   PSW            ;SAVE A PLUS FLAGS
0027 E5           64          PUSH   H              ;SAVE HL
0028 D5           65          PUSH   D              ;SAVE DE
0029 C5           66          PUSH   B              ;SAVE BC
002A 210000       67          LXI    H,0000H        ;GET SET TO GET SP
002D 39           68          DAD    SP             ;SP NOW IN HL
002E 22003F       69          SHLD   SPSAVE         ;SAVE SP IN RAM
                  70          ;
                  71          ;EOI NOT REALLY NEEDED BUT INCLUDED FOR COMPLETENESS
                  72          ;
0031 3E20         73          MVI    A,20H          ;NON-SPECIFIC EOI
0033 D3DA         74          OUT    PT59A          ;8259 PORT WITH A0=0
0035 76           75          HLT                   ;HALT - GO DOWN GRACEFULLY
                  76 ;
                  77 ;
                  78 ;8259 JUMP TABLE. ONLY IR0 IS USED, OTHERS DIRECTED TO RAM
                  79 ;
                  80
0100              81          ORG    100H
0100 C32600       82 JTSTRT:  JMP    PWDOWN         ;IR0
0103 00           83          NOP
0104 C3103F       84          JMP    3F10H          ;IR1
0107 00           85          NOP
0108 C3203F       86          JMP    3F20H          ;IR2
010B 00           87          NOP
010C C3303F       88          JMP    3F30H          ;IR3
010F 00           89          NOP
0110 C3403F       90          JMP    3F40H          ;IR4
0113 00           91          NOP
0114 C3503F       92          JMP    3F50H          ;IR5
0117 00           93          NOP
0118 C3603F       94          JMP    3F60H          ;IR6
011B 00           95          NOP
011C C3703F       96          JMP    3F70H          ;IR7
011F 00           97          NOP
                  98 ;
                  99 ;
                 100 ;COLD START LOCATION. USER'S PROGRAM ENTERS HERE
                 101 ;
                 102 ;
0120 31803F      103 CSTART:  LXI    SP,3F80H       ;INITIALIZE SP
0123 CD1D00      104          CALL   INIT           ;INITIALIZE EVERYTHING ELSE
0126 D3D4        105          OUT    0D4H           ;RESET PFI LATCH
0128 FB          106          EI                    ;ENABLE INTERRUPTS
                 107          ;
                 108          ;USER PROGRAM STARTS HERE
                 109          ;
                 110          END                   ;DONE
```

Figure 27. Power Down and Restart Software

5.2 78 LEVEL INTERRUPT SYSTEM

The second application illustrates an interrupt structure with greater than 64 levels for an 8080A or 8085A system. In the cascade mode, the 8259A supports up to 64 levels with direct vectoring to the service routine. Extending the structure to greater than 64 levels requires polling, using the poll command. A 78 level interrupt structure is used as an illustration; however, the principles apply to systems with up to 512 levels.

To implement the 78 level structure, 3 tiers of 8259A's are used. Nine 8259A's are cascaded in the master-slave scheme, giving 64 levels at tier 2. Two additional 8259A's are connected, by way of the INT outputs, to two of the 64 inputs. The 16 inputs at tier 3, combined with the 62 remaining tier 2 inputs, give 78 total levels. The fully nested structure is preserved over all levels, although direct vectoring is supplied for only the tier 2 inputs. Software is required to vector any tier 3 requests. Figure 28 shows the tiered structure used in this example. Notice that the tier 3 8259A's are connected to the bottom level slave (SA7). The master-slaves are interconnected as shown in "Interrupt Cascading", while the tier 3 8259A's are connected as "masters"; that is, the $\overline{\text{SP}}$/EN pins are pulled high and the CAS pins are left unconnected. Since these 8259A's are only going to be used with the poll command, no $\overline{\text{INTA}}$ is required, therefore the $\overline{\text{INTA}}$ pins are pulled high.

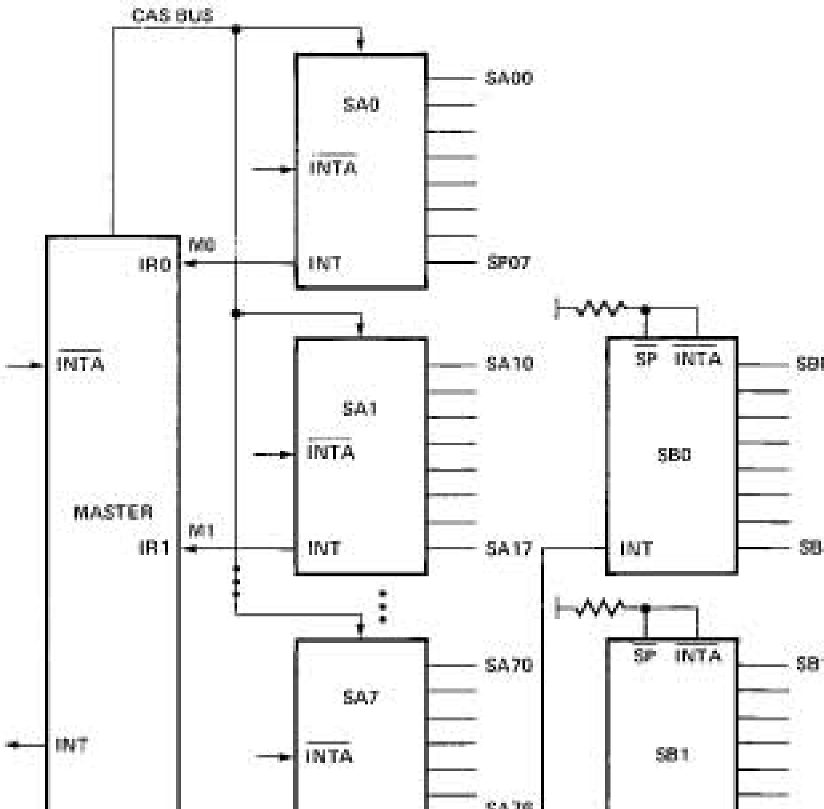


Figure 28. 78 Level Interrupt Structure

The concept used to implement the 78 levels is to directly vector to all tier 2 input service routines. If a tier 2 input contains a tier 3 8259A, the service routine for that input will poll the tier 3 8259A and branch to the tier 3 input service routine based on the poll word read after the poll command. Figure 29 shows how the jump table is organized assuming a starting location of 1000H and contiguous tables for all the tier 2 8259A's. Note that "SA35" denotes the IR5 input of the slave connected to the master IR3 input. Also note that for the normal tier 2 inputs, the jump table vectors the processor directly to the service routine for that input, while for the tier 2 inputs with 8259A's connected to their IR inputs, the processor is vectored to a service routine (i.e., SB0) which will poll to determine the actual tier 3 input requesting service. The polling routine utilizes the jump table starting at 1200H to vector the processor to the correct tier 3 service routine.

Each 8259A must receive an initialization sequence regardless of the mode. Since the tier 1 and 2 8259A's are in cascade and the special fully nested mode is used (covered shortly), all ICWs are required. The tier 3 8259A's don't require ICW3 or ICW4 since only polling will be used on them and they are connected as masters not in the cascade mode. The initialization sequence for each tier is shown in Figure 30. Notice that the master is initialized with a "dummy" jump table starting at 00H since all vectoring is done by the slaves. The tier 3 devices also receive "dummy" tables since only polling is used on tier 3.

As explained in "Interrupt Cascading", to preserve a truly fully nested mode within a slave, the master 8259A should be programmed in the special fully nested mode. This allows the master to acknowledge all interrupts at and above the level in service disregarding only those of lower priority. The special fully nested mode is programmed in the master only, so it only affects the immediate slaves (tier 2 not tier 3). To implement a fully nested structure among tier 3 slaves some special housekeeping software is required in all the tier-2-with-tier-3-slave routines. The software should simply save the state of the tier 2 IMR, mask all the lower tier 2 interrupts, then issue a specific EOI, resetting the ISR of the tier 2 interrupt level. On completion of the routine the IMR is restored.

Figure 31 shows an example flow and program for any tier 2 service routine without a tier 3 8259A. Figure 32 shows an example flow and program for any tier 2 service routine with a tier 3 8259A. Notice the reading of the ISR in both examples; this is done to determine whether or not to issue an EOI command to the master (refer to the section on "Special Fully Nested Mode" for further details).

```
LOCATION    8259    CODE            COMMENTS

 1000 H     SA0     JMP   SA00      ; SA00 SERVICE ROUTINE
   .
   .
 101C H             JMP   SA07      ; SA07 SERVICE ROUTINE
 1020 H     SA1     JMP   SA10      ; SA10 SERVICE ROUTINE
   .
   .
 103C H             JMP   SA17      ; SA17 SERVICE ROUTINE
   .         .           .
   .         .           .          ; SA20–SA67 SERVICE ROUTINES
   .         .           .
 10E0 H     SA7     JMP   SA70      ; SA70 SERVICE ROUTINE
   .
   .
 10F8 H             JMP   SB0       ; SB0 POLL ROUTINE
 10FC H             JMP   SB1       ; SB1 POLL ROUTINE
 1200 H     SB0     JMP   SB00      ; SB00 SERVICE ROUTINE
   .
   .
 121C H             JMP   SB07      ; SB07 SERVICE ROUTINE
 1220 H     SB1     JMP   SB10      ; SB10 SERVICE ROUTINE
   .
   .
 123C H             JMP   SB17      ; SB17 SERVICE ROUTINE
```

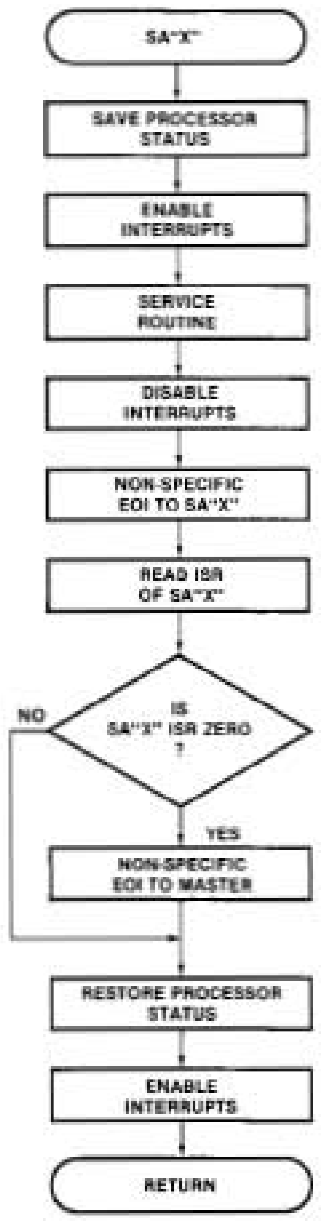
Figure 29. Jump Table Organization

```
; INITIALIZATION SEQUENCE FOR 78 LEVEL INTERRUPT STRUCTURE
;
; INITIALIZE MASTER
;
 MINT:    MVI     A,15H      ; ICWI, LTM=0, ADI=1, S=0, IC4=1
          OUT     MPTA       ; MASTER PORT A0=0
          MVI     A,00H      ; ICW2, DUMMY ADDRESS
          OUT     MPTB       ; MASTER PORT A0=1
          MVI     A,0FFH     ; ICW3, S7-S0=1
          OUT     MPTB       ; MASTER PORT A0=1
          MVI     A,I0H      ; ICW4, SFNM=1
          OUT     MPTB       ; MASTER PORT A0=1
;
; INITIALIZE SA SLAVES - X DENOTES SLAVE ID (SEE KEY)
;
 SAXINT:  MVI     A,α        ; SEE KEY FOR ICW1, LTM=0, ADI=1, S=0, IC4=1
          OUT     SAXPTA     ; SA"X" PORT A0=0
          MVI     A, 10H     ; ICW2, ADDRESS MSB
          OUT     SAXPTB     ; SA"X" PORT A0=1
          MVI     A0XH       ; ICW3, SA ID
          OUT     SAXPTB     ; SA"X" PORT A0=1
          MVI     A10H       ; ICW4, SFNM=1
          OUT     SAXPTB     ; SA"X" PORT A0=1
;
; REPEAT ABOVE FOR EACH SA SLAVE
;
; INITIALIZE SB SLAVES - X DENOTES 0 or 1 (DO SB0, REPEAT FOR SB1)
;
 SBXINT   MVI     A,I6H      ; ICW1, LTM=0, ADI=1, S=1, IC4=0
          OUT     SBXPTA     ; SB"X" PORT A0=0
          MVI     A,00H      ; ICW2, DUMMY ADDRESS
          OUT     SBXPTB     ; SB"X" PORT A0=1
```

SA INITIALIZATION KEY

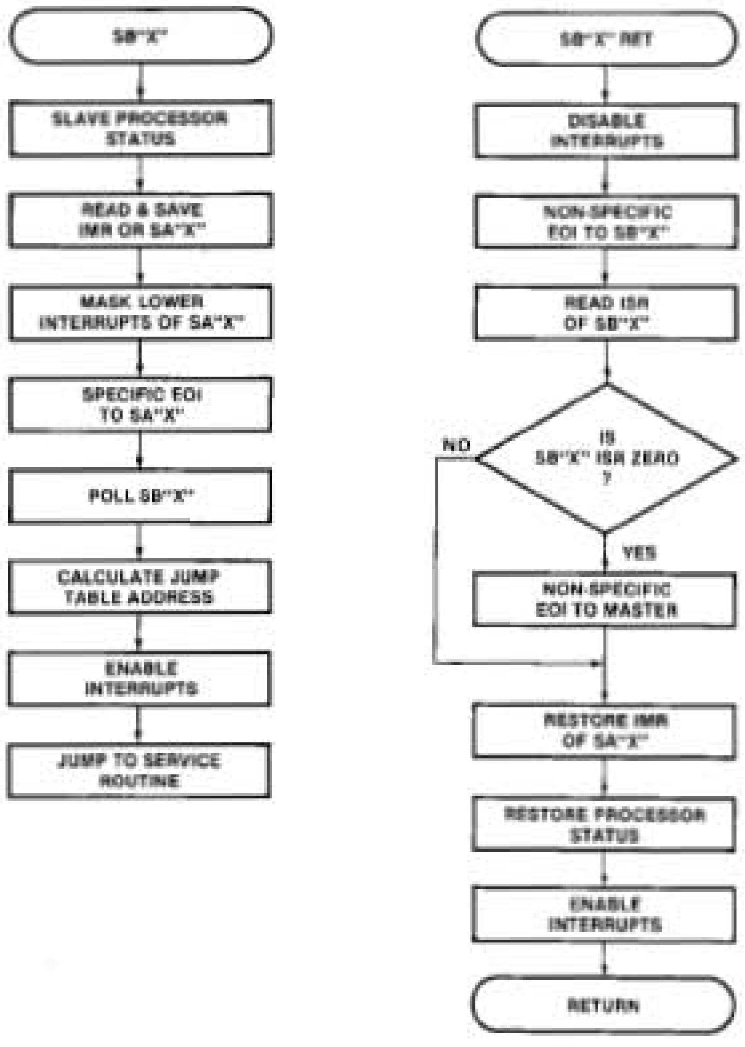
SA"X"	α (ICW1)	JUMP TABLE START (H)
0	15	1000
1	35	1020
2	55	1040
3	75	1060
4	95	1080
5	B5	10A0
6	D5	10C0
7	F5	10E0

Figure 30. Initialization Sequence for 78 Level Interrupt Structure



```
; SA"X" ROUTINE - GENERAL INTERRUPT SERVICE ROUTINE
; FOR TIER 2 INTERRUPTS WITHOUT TIER 3 8259A
;
  SAX:     PUSH D                  ; SAVE DE
           PUSH B                  ; SAVE BC
           PUSH H                  ; SAVE HL
           PUSH PSW                ; SAVE A, FLAGS
           EI                      ; ENABLE INTERRUPTS
;
; SERVICE ROUTINE GOES HERE
;
           DI                      ; DISABLE INTERRUPTS
           MVI     20H             ; OCW2, NON-SPECIFIC EOI
           OUT     SAXPTA          ; SA"X" PORT A0 = 0
           MUI     A,0BH           ; OCW3, READ REGISTER, ISR
           OUT     SAXPTA          ; SA"X" PORT A0 = 0
           IN      SAXPTA          ; SA"X" PORT A0 = 0, SA"X" ISR
           ANI     0FFH            ; TEST FOR ZERO
           JZN     SAXRSR          ; IF NOT ZERO, RESTORE STATUS
           MVI     A,0BH           ; OCW2, NON-SPECIFIC EOI
           OUT     MASPTA          ; MASTER PORT A0 = 0
  SAXRSR:  POP     PSW             ; RESTORE A, FLAGS
           POP     H               ; RESTORE HL
           POP     B               ; RESTORE BC
           POP     D               ; RESTORE DE
           EI                      ; ENABLE INTERRUPTS
           RET                     ; RETURN
```

Figure 31. Example Service Routine for Tier 2 Interrupt (SA"X") without Tier 3 8259A (SB"X")



```
; SB"X" ROUTINE - SERVICE ROUTINE FOR TIER 2
; INTERRUPTS WITH TIER 3 8259AS
  SBX:     PUSH D                  ; SAVE DE
           PUSH B                  ; SAVE BC
           PUSH H                  ; SAVE HL
           PUSH PSW                ; SAVE A, FLAGS
           IN      SAXPTB          ; READ SA"X" IMR
           MOV     D,A             ; SAVE
           MVI     A,XXH           ; MASK SA"X" LOWER IR
           OUT     SAXPTB          ; SA"X" PORT A0 = 1
           MVI     A,6XH           ; OCW2 SPECIFIC EOI SA"X"
           OUT     SAXPTA          ; SA"X" PORT A0 = 1
           LXI     H,1200H         ; JUMP TABLE START
           MVI     B,00H           ; CLEAR B
           MVI     A,0CH           ; OCW3, POLL COMMAND
           OUT     SBXPTA          ; SB"X" PORT A0 = 0
           IN      SBXPTA          ; GET POLL WORD
           ANI     07H             ; LIMIT TO 3 BITS
           ADD     A               ; GET TABLE OFFSET
           ADD     A               ;
           MOV     C,A             ; OFFSET TO C
           DAD     B               ; HL HAS TABLE ADDRESS
           EI                      ; ENABLE INTERRUPTS
;
; SB"X"RET ROUTINE - FOR EOI AND MASK RESTORE
; AFTER SB"X" ROUTINE
;
  SBXRET   DI                      ; DISABLE INTERRUPTS
           MVI     A,20H           ; OCW2, NON SPECIFIC EOI
           OUT     SBXPTA          ; SA"X" PORT A0 = 0
           MVI     A,0BH           ; OCW3, READ REGISTER ISR
           OUT     SAXPTA          ; SA"X" PORT A0 = 0
           IN      SBXPTA          ; SA"X" PORT A0 = 0, ISR
           ANI     0FFH            ; TEST FOR ZERO
           JNZ     SBXRSR          ; IF ≠ 0 RESTORE IMR
           MVI     A,20H           ; OCW2, NON SPECIFIC EOI
           OUT     MASPTA          ; MASTER PORT A0 = 0
  SBXRSR:  MOV     A,D             ; RESTORE SA"X" IMR
           OUT     SAXPTB          ; SA"X" PORT A0 = 1
           POP     PSW             ; RESTORE A, FLAGS
           POP     H               ; RESTORE HL
           POP     B               ; RESTORE BC
           POP     D               ; RESTORE BC
           EI                      ; RESTORE DE
           RET                     ; RETURN
```

Figure 32. Example Service Routine for Tier 2 Interrupt (SA"X") with Tier 3 8259A (SB"X")

5.3 TIMER CONTROLLED INTERRUPTS

In a large number of controller type microprocessor designs, certain timing requirements must be implemented throughout program execution. Such time dependent applications include control of keyboards, displays, CRTs, printers, and various facets of industrial control. These examples, however, are just a few of many designs which require device servicing at specific rates or generation of time delays. Trying to maintain these timing requirements by processor control alone can be costly in throughput and software complexity. So, what can be done to alleviate this problem? The answer, use the 8259A Programmable Interrupt Controller and external timing to interrupt the processor for time dependent device servicing.

This application example uses the 8259A for timer controlled interrupts in an 8086 system. External timing is done by two 8253 Programmable Interval Timers. Figure 33 shows a block diagram of the timer controlled interrupt circuitry which was built on the breadboard area of an SDK-86 (system design kit). Besides the 8259A and the 8253's, the necessary I/O decoding is also shown. The timer controlled interrupt circuitry interfaces with the SDK-86 which serves as the vehicle of operation for this design.

A short overview of how this application operates is as follows. The 8253's are programmed to generate interrupt requests at specific rates to a number of the 8259A IR inputs. The 8259A processes these requests by interrupting the 8086 and vectoring program execution to the appropriate service routine. In this example, the routines use the SDK-86 display panel to display the number of the interrupt level being serviced. These routines are merely for demonstration purposes to show the necessary procedures to establish the user's own routines in a timer controlled interrupt scheme.

Let's go over the operation starting with the actual interrupt timing generation which is done by two 8253 Programmable Interval Timers (8253 #1 and 8253 #2). Each 8253 provides three individual 16-bit counters (counters 0–2) which are software programmable by the processor. Each counter has a clock input (CLK), gate input (GATE), and an output (OUT). The output signal is based on divisions of the clock input signal. Just how or when the output occurs is determined by one of the 8253's six programmable modes, a programmable 16-bit count, and the state of the gate input.

Figure 34 shows the 8253 timing configuration used for generating interrupts to the 8259A. The SDK-86's PCLK (peripheral clock) signal provides a 400 ns period clock to CLK0 of 8253 #1. Counter 0 is used in mode 3 (square wave rate generator), and acts as a prescaler to provide the clock inputs of the other counters with a 10 ms period square wave. This 10 ms clock period made it easy to calculate exact timings for the other counters. Counter 2 of the 8253 #1 is used in mode 2 (rate generator), it is programmed to output a 10 ms pulse for every 200 pulses it receives (every 2 sec). The output of counter 2 causes an interrupt on IR1 of the 8259A. All the 8253 #2 counters are used in mode 5 (hardware triggered strobe) in which the gate input initiates counter operations. In this case the output of 8253 #1 counter 2 controls the gate of each 8253 #2 counter. When one of the 8253 #2 counters receive the 8253 #1 counter 2 output pulse on its gate, it will output a pulse (10 ms in duration) after a certain preprogrammed number of clock pulses have occurred. The programmed number of clock pulses for the 8253 #2 counters is as follows: 50 pulses (0.5 sec) for counter 0, 100 pulses (1 sec) for counter 1, and 150 pulses (1.5 sec) for counter 2. The outputs of these counters cause interrupt requests on IR2 through IR4 of the 8259A. Counter 1 of 8253 #1 is used in mode 0 (interrupt on terminal count). Unlike the other modes used which initialize operation automatically or by gate triggering, mode 0 allows software controlled counter initialization. When counter 1 of 8253 #1 is set during program execution, it will count 25 clocks (250 ms) and then pull its output high, causing an interrupt request on IR0 of the 8259A. Figure 35 shows the timing generated by the 8253's which cause interrupt request on the 8259A IR inputs.

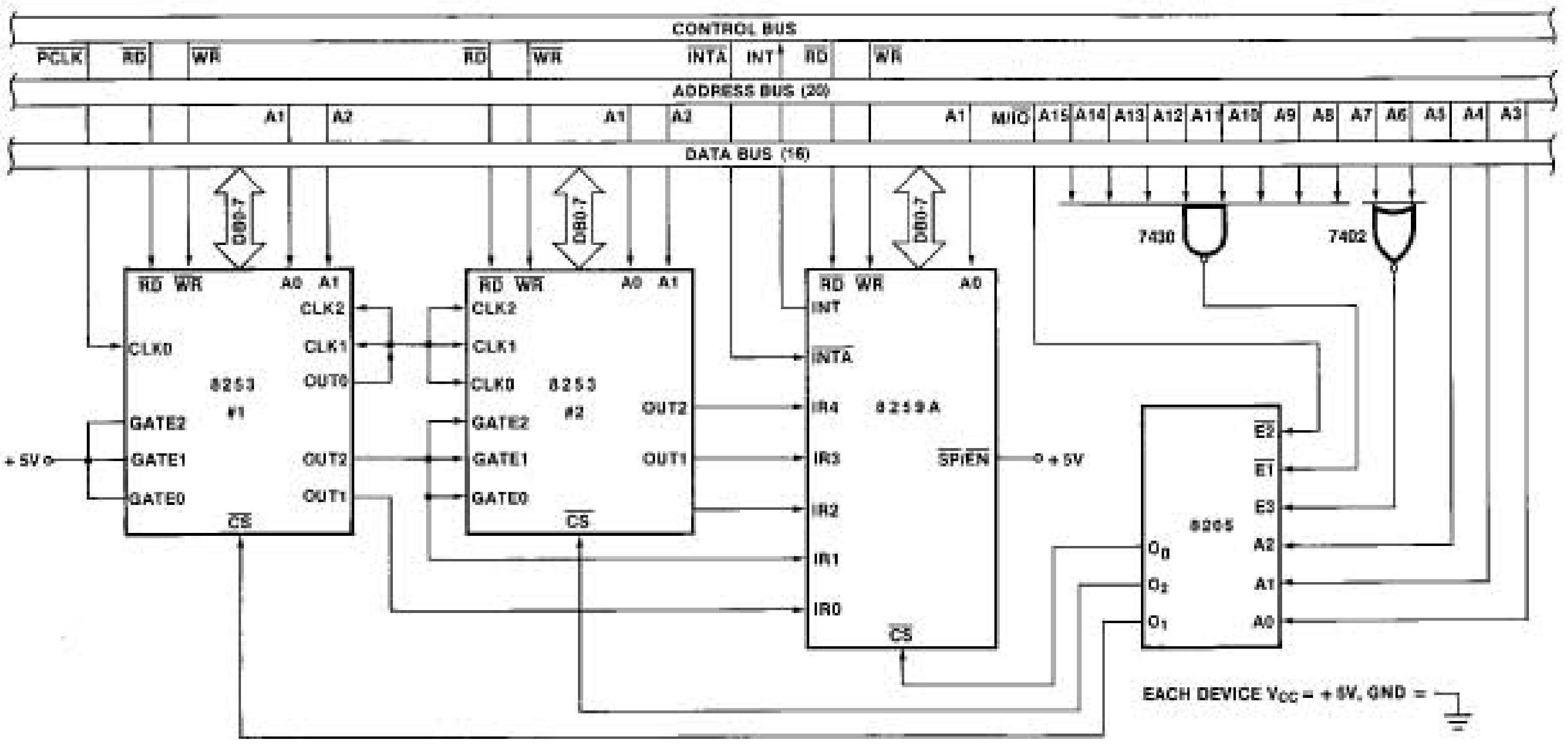


Figure 33. Timer Controlled Interrupt Circuit on SDK 86 Breadboard Area

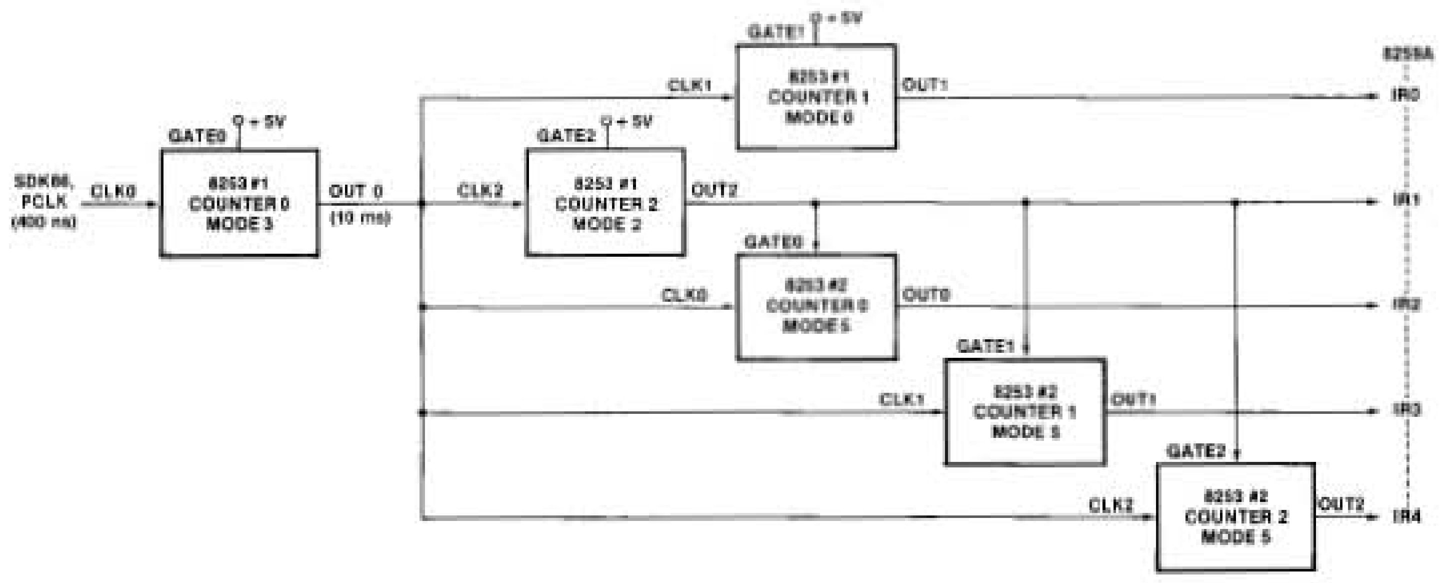


Figure 34. 8253 Timing Configuration for Timer Controlled Interrupts

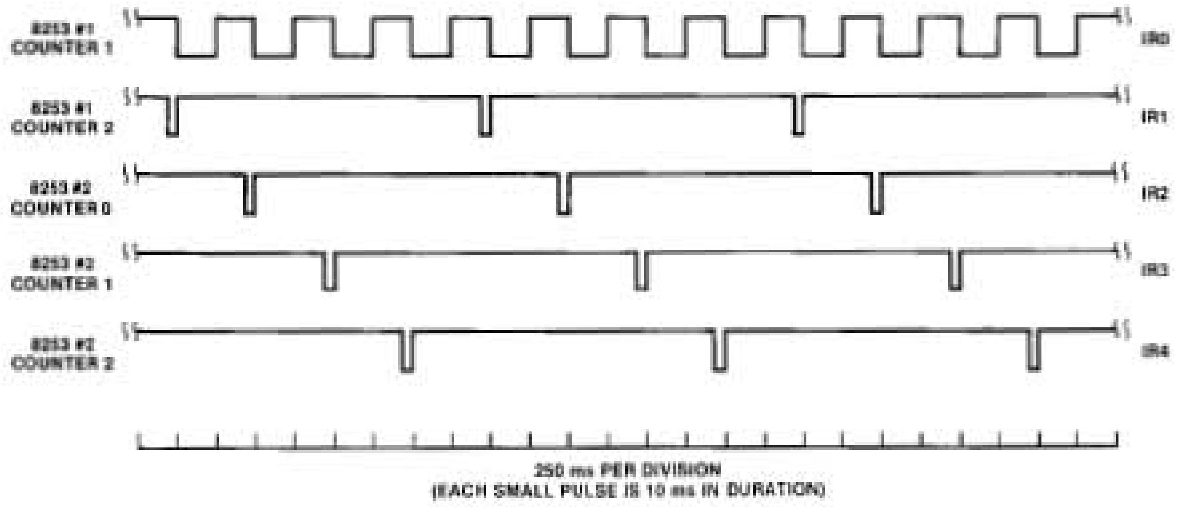


Figure 35. 8259A IR Input Signal From 8253S

There are basically two methods of timing generation that can be used in a timer controlled interrupt structure: dependent timing and independent timing. Dependent timing uses a single timing occurrence as a reference to base other timing occurrences on. On the other hand, independent timing has no mutual reference between occurrences. Industrial controller type applications are more apt to use dependent timing, whereas independent timing is prone to individual device control.

Although this application uses primarily dependent timing, independent timing is also incorporated as an example. The use of dependent timing can be seen back in Figure 34, where timing for IR2 through IR4 uses the IR1 pulse as reference. Each one of the 8253 #2 counters will generate an interrupt request a specific amount of times after the IR1 interrupt request occurs. When using the dependent method, as in this case, the IR2 through IR4 requests must occur before the next IR1 request. Independent timing is used to control the IR0 interrupt request. Note that its timing isn't controlled by any of the other IR requests. In this timer controlled interrupt configuration the dependent timing is initially set to be self running and the independent timing is software initialized. However, both methods can work either way by using the various 8253 modes to generate the same interrupt timing.

The 8259A processes the interrupts generated by the 8253's according to how it is programmed. In this application it is programmed to operate in the edge triggered mode, MCS-86/88 mode, and automatic EOI mode. In the edge triggered mode an interrupt request on an 8259A IR input becomes active on the rising edge. With this in mind, Figure 35 shows that IR0 will generate an interrupt every half second and IR1 through IR4 will each generate an interrupt every 2 seconds spaced apart at half second intervals. Interrupt vectoring in the MCS-86/88 mode is programmed so IR0, when activated, will select interrupt type 72. This means IR1 will select interrupt type 73, IR2 interrupt type 74, and so on through IR4. Since IR5 through IR7 aren't used, they are masked off. This prevents the possibility of any accidental interrupts and rids the necessity to tie the unused IR inputs to a steady level. Figure 36 shows the 8259A IR levels (IR0–IR4) with their corresponding interrupt type in the 8086 interrupt-vector table. Type 77 in the table is selected by a software "INT" instruction during program execution. Each type is programmed with the necessary code segment and instruction pointer values for vectoring to the appropriate service routine. Since the 8259A is programmed in the automatic EOI Mode, it doesn't require an EOI command to designate the completion of the service routine.

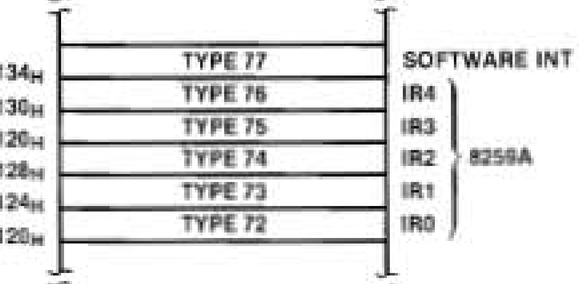


Figure 36. Interrupt "Type" Designation

As mentioned earlier, the interrupt service routines in this application are used merely to demonstrate the timer controlled interrupt scheme, not to implement a particular design. Thus a service routine simply displays the number of its interrupting level on the SDK-86 display panel. The display panel is controlled by the 8279 Keyboard and Display Controller. It is initialized to display "Ir" in its two left-most digits during the entire display sequence. When an interrupt from IR1 through IR4 occurs the corresponding routine will display its IR number via the 8279. During each IR1 through IR4 service routine a software "INT77" instruction is executed. This instruction vectors program execution to the service routine designated by type 77, which sets the 8253 counter controlling IR0 so it will cause an interrupt in 250 ms. When the IR0 interrupt occurs its routine will turn off the digit displayed by the IR1 through IR4 routines. Thus each IR level (IR1-IR4) will be displayed for 250 ms followed by a 250 ms off time caused by IR0. Figure 37 shows the entire display sequence of the timer controlled interrupt application.

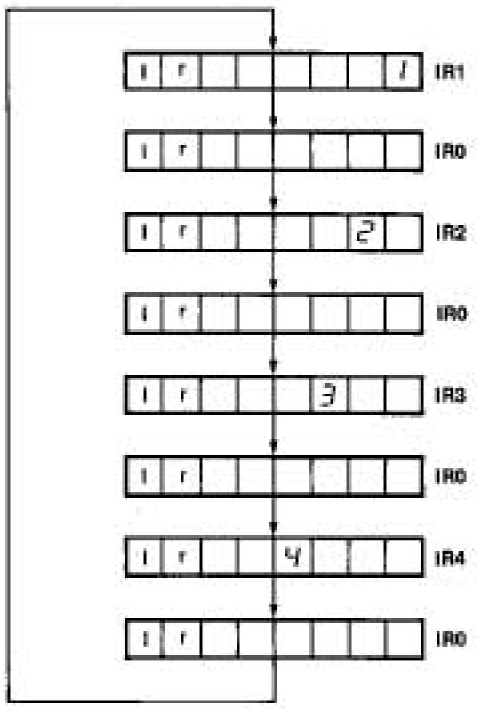
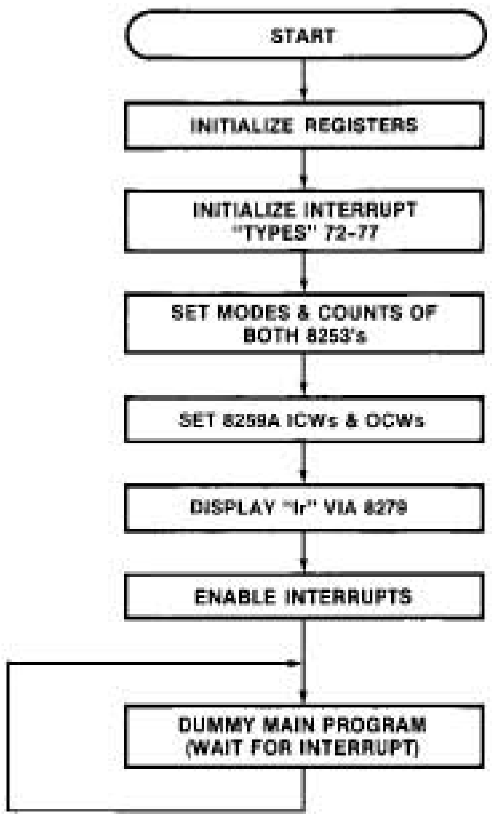


Figure 37. SDK Display Sequence for Timer Controlled Interrupts Program (Each Display Block Shown is 250 msec in Duration)

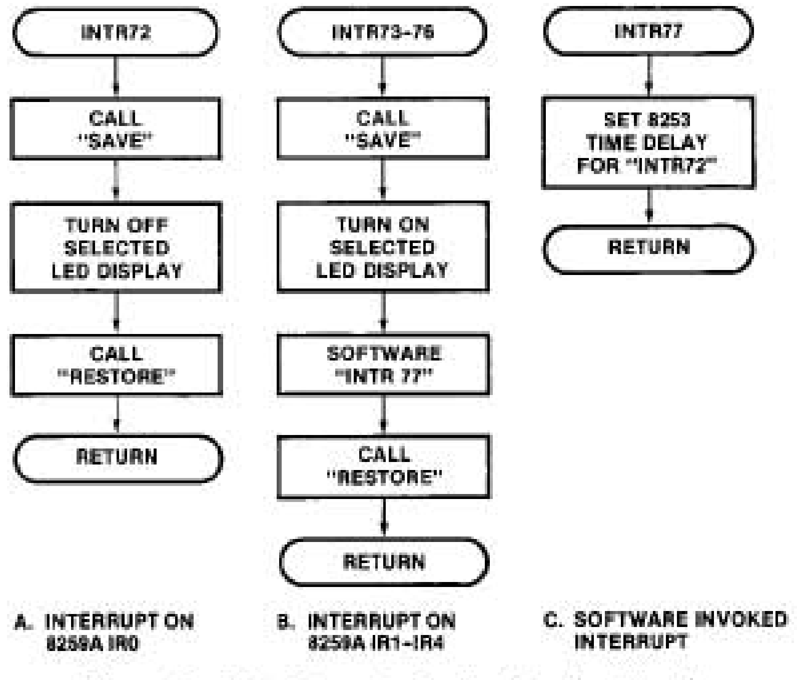
Now that we've covered the operation, let's move on to the program flow and structure of the timer controlled interrupt program. The program flow is made up of an initialization section and six interrupt service routines. The initialization program flow is shown in Figure 38. It starts by initializing some of the 8086's registers for program operation; this includes the extra segment, data segment, stack segment, and stack pointer. Next, by using the extra segement as reference, interrupt types 72 through 77 are set to vector interrupts to the appropriate routines. This is done by moving the code segment and instruction pointer values of each service routine into the corresponding type location. The 8253 counters are then programmed with the proper mode and count to provide the interrupt timing mentioned earlier. All counters with the exception of the 8253 #1, counter 1 are fully initialized at this point and will start counting. Counter 1 of 8253 #1 starts counting when its counter is loaded during the "INTR77" service routine, which will be covered shortly. Next, the 8259A is issued ICW1, ICW2, ICW4, and OCW1. The ICWs program the 8259A for the edge triggered mode, automatic EOI mode, and the proper interrupt vectoring (IR0, type 72). OCW1 is used to mask off the unused IR inputs (IR5-IR7). The 8279 is then set to display "IR" on its two left-most digits. After that the 8086 enables interrupts and a "dummy" main program is executed to wait for interrupt requests.



START
INITIALIZE REGISTERS
INITIALIZE INTERRUPT "TYPES" 72-77
SET MODES & COUNTS OF BOTH 8253's
SET 8259A ICWs & OCWs
DISPLAY "Ir" VIA 8279
ENABLE INTERRUPTS
DUMMY MAIN PROGRAM (WAIT FOR INTERRUPT)

Figure 38. Initialization Program Flow for Timer Controlled Interrupts

There are six different interrupt service routines used in the program. Five of these routines, "INTR72" through "INTR76", are vectored to via the 8259A. Figure 39A-C shows the program flow for all six service routines. Note that "INTR73" through "INTR76" (IR1-IR4) basically use the same flow. These four similar routines display the number of its interrupting IR level on the SDK-86 display panel. The "INTR77" routine is vectored to by software during each of the previously mentioned routines and sets up interrupt timing to cause the "INTR72" (IR0) routine to be executed. The "INTR72" routine turns off the number on the SDK-86 display panel.



INTR72
CALL "SAVE"
TURN OFF SELECTED LED DISPLAY
CALL "RESTORE"
RETURN
INTR73-76
CALL "SAVE"
TURN ON SELECTED LED DISPLAY
SOFTWARE "INTR 77"
CALL "RESTORE"
RETURN
INTR77
SET 8253 TIME DELAY FOR "INTR72"
RETURN
A. INTERRUPT ON 8259A IR0
B. INTERRUPT ON 8259A IR1-IR4
C. SOFTWARE INVOKED INTERRUPT

Figure 39. A-C. Interrupts Service Routine Flow for Timer Controlled Interrupts.

To best explain how these service routines work, let's assume an interrupt occurred on IR1 of the 8259A. The associated service routine for IR1 is "INTR73". Entering "INTR73", the first thing done is saving the pre-interrupt program status. This isn't really necessary in this program since a "dummy" main program is being executed; however, it is done as an example to show the operation. Rather than having code for saving the registers in each separate routine, a mutual call routine, "SAVE", is used. This routine will save the register status by pushing it on the stack. The next portion of "INTR73" will display the number of its IR level, "1", in the first digit of the SDK-86 display panel. After that, a software INT instruction is executed to vector program execution to the "INTR77" service routine. The "INTR77" service routine simply sets the 8253 #1 counter 1 to cause an interrupt on IR0 in 250 ms and then returns to "INTR73". Once back in "INTR73", the pre-interrupt status is restored by a call routine, "RESTORE". It does the opposite of "SAVE", returning the register status by popping it off the stack. The "INTR73" routine then returns to the "dummy " main program. The flow for the "INTR74" through "INTR76" routines are the same except for the digit location and the IR level displayed.

After 250 ms have elapsed, counter 1 of 8253 #1 makes an interrupt request on IR0 of the 8259A. This causes the "INTR72" service routine to be executed. Since this routine interrupts the main program, it also uses the "SAVE" routine to save pre-interrupt program status. It then turns off the digit displaying the IR level. In the case of the "INTR73" routine, the "1" is blanked out. The pre-interrupt status is then restored using the "RESTORE" routine and program execution returns to the "dummy" main program.

The complete program for the timer controlled interrupts application is shown in Appendix B. The program was executed in SDK-86 RAM starting at location 0500H (code segment = 0050, instruction pointer = 0).

CONCLUSION

This application note has explained the 8259A in detail and gives three applications illustrating the use of some of the numerous programmable features available. It should be evident from these discussions that the 8259A is an extremely flexible and easily programmable member of the Intel® MCS-80, MCS-85, MCS-86, and MCS-88 families.

APPENDIX A

This table is provided merely for reference information between the "Operation of the 8259A" and "Programming the 8259A" sections of this application note. It shouldn't be used as a programming reference guide (see "Programming the 8259A").

Operational Description	Command Words	Bits
MCS-80/85™ Mode	ICW1, ICW4*	IC4, μPM*
Address Interval for MCS-80/85 Mode	ICW1	ADI
Interrupt Vector Address for MCS-80/85 Mode	ICW1, ICW2	A5–A15
MCS-86/88 Mode	ICW1, ICW4	IC4, μPM
Interrupt Vector Byte for MCS-86/88 Mode	ICW2	T3–T7
Fully Nested Mode	OCW–Default	—
Non-Specific EOI Command	OCW2	EOI
Specific EOI Command	OCW2	SEOI, EOI, LO–L2
Automatic EOI Mode	ICW1, ICW4	IC4, AEOI
Rotate On Non-Specific EOI Command	OCW2	EOI
Rotate In Automatic EOI Mode	OCW2	R, SEOI, EOI
Set Priority Command	OCW2	L0–L2
Rotate on Specific EOI Command	OCW2	R, SEOI, EOI
Interrupt Mask Register	OCW1	M0–M7
Special Mask Mode	OCW3	ESMM–SMM
Level Triggered Mode	ICW1	LTIM
Edge Triggered Mode	ICW1	LTIM
Read Register Command, IRR	OCW3	ERIS, RIS
Read Register Command, ISR	OCW3	ERIS, RIS
Read IMR	OCW1	M0–M7
Poll Command	OCW3	P
Cascade Mode	ICW1, ICW3	SNGL, S0-7, ID0-2
Special Fully Nested Mode	ICW1, ICW4	IC4, SFNM
Buffered Mode	ICW1, ICW4	IC4, BUF, M/S

*Only needed if ICW4 is used for purposes other than μP mode set.

```
MCS-86 ASSEMBLER    TCI59A                                                                              PAGE    1


ISIS-II MCS-86 ASSEMBLER V1.0 ASSEMBLY OF MODULE TCI59A
OBJECT MODULE PLACED IN :F1:TCI59A.OBJ
ASSEMBLER INVOKED BY: :F1:ASM86 :F1:TCI59A.SRC

LOC  OBJ                  LINE   SOURCE

                             1   ;******************* TIMER CONTROLLED INTERRUPTS ******************
                             2   ;
                             3   ;
                             4   ;
                             5   ;              EXTRA SEGMENT DECLARATIONS
                             6   ;
----                         7   EXTRA SEGMENT
                             8   ;
0120                         9          ORG     120H
0120 0401                   10   TP72IP DW      INTR72          ;TYPE 72 INSTRUCTION POINTER
0122 ????                   11   TP72CS DW      ?               ;TYPE 72 CODE SEGMENT
0124 1801                   12   TP73IP DW      INTR73          ;TYPE 73 INSTRUCTION POINTER
0126 ????                   13   TP73CS DW      ?               ;TYPE 73 CODE SEGMENT
0128 3001                   14   TP74IP DW      INTR74          ;TYPE 74 INSTRUCTION POINTER
012A ????                   15   TP74CS DW      ?               ;TYPE 74 CODE SEGMENT
012C 4801                   16   TP75IP DW      INTR75          ;TYPE 75 INSTRUCTION POINTER
012E ????                   17   TP75CS DW      ?               ;TYPE 75 CODE SEGMENT
0130 6001                   18   TP76IP DW      INTR76          ;TYPE 76 INSTRUCTION POINTER
0132 ????                   19   TP76CS DW      ?               ;TYPE 76 CODE SEGMENT
0134 7801                   20   TP77IP DW      INTR77          ;TYPE 77 INSTRUCTION POINTER
0136 ????                   21   TP77CS DW      ?               ;TYPE 77 CODE SEGMENT
                            22   ;
----                        23   EXTRA ENDS
                            24   ;
                            25   ;              DATA SEGMENT DECLARATIONS
                            26   ;
----                        27   DATA   SEGMENT
                            28   ;
0000 ????                   29   STACK1 DW      ?                       ;VARIABLE TO SAVE CALL ADDRESS
0002 ????                   30   AXTEMP DW      ?                       ;VARIABLE TO SAVE AX REGISTER
0004 ??                     31   DIGIT  DB      ?                       ;VARIABLE TO SAVE SELECTED DIGIT
                            32   ;
----                        33   DATA   ENDS
                            34   ;
                            35   ;              CODE SEGMENT DECLARATION
                            36   ;
----                        37   CODE   SEGMENT
                            38   ;
                            39   ASSUME ES:EXTRA,DS:DATA,CS:CODE
                            40   ;
                            41   ;              INITIALIZE REGISTERS
                            42   ;
0000 B80000                 43   START: MOV     AX,0H                   ;EXTRA SEGMENT AT 0H
0003 8EC0                   44          MOV     ES,AX
0005 B87000                 45          MOV     AX,70H                  ;DATA SEGMENT AT 700H
0008 8ED8                   46          MOV     DS,AX
000A B87800                 47          MOV     AX,78H                  ;STACK SEGMENT AT 780H
000D 8ED0                   48          MOV     SS,AX
000F BC8000                 49          MOV     SP,80H                  ;STACK POINTER AT 80H (STACK=800H)
```

```
MCS-86 ASSEMBLER    TCI59A                                                              PAGE    2

LOC  OBJ              LINE   SOURCE

                        50   ;
                        51   ;              LOAD INTERRUPT VECTOR TABLE
                        52   ;
0012 B80401             53   TYPES:  MOV    AX,OFFSET (INTR72)      ;LOAD TYPE 72
0015 26A32001           54           MOV    TP72IP,AX
0019 268C0E2201         55           MOV    TP72CS,CS
001E B81801             56           MOV    AX,OFFSET (INTR73)      ;LOAD TYPE 73
0021 26A32401           57           MOV    TP73IP,AX
0025 268C0E2601         58           MOV    TP73CS,CS
002A B83001             59           MOV    AX,OFFSET (INTR74)      ;LOAD TYPE 74
002D 26A32801           60           MOV    TP74IP,AX
0031 268C0E2A01         61           MOV    TP74CS,CS
0036 B84801             62           MOV    AX,OFFSET (INTR75)      ;LOAD TYPE 75
0039 26A32C01           63           MOV    TP75IP,AX
003D 268C0E2E01         64           MOV    TP75CS,CS
0042 B86001             65           MOV    AX,OFFSET (INTR76)      ;LOAD TYPE 76
0045 26A33001           66           MOV    TP76IP,AX
0049 268C0E3201         67           MOV    TP76CS,CS
004E B87801             68           MOV    AX,OFFSET (INTR77)      ;LOAD TYPE 77
0051 26A33401           69           MOV    TP77IP,AX
0055 268C0E3601         70           MOV    TP77CS,CS
                        71   ;
                        72   ;              8253 INITIALIZATION
                        73   ;
005A BA0EFF             74   SET531: MOV    DX,0FF0EH               ;8253 #1 CONTROL WORD
005D B036               75           MOV    AL,36H                  ;COUNTER 0, MODE 3, BINARY
005F EE                 76           OUT    DX,AL
0060 B071               77           MOV    AL,71H                  ;COUNTER 1, MODE 0, BCD
0062 EE                 78           OUT    DX,AL
0063 B0B5               79           MOV    AL,0B5H                 ;COUNTER 2, MODE 2, BCD
0065 EE                 80           OUT    DX,AL
0066 BA08FF             81           MOV    DX,0FF08H               ;LOAD COUNTER 0 (10MS)
0069 B0A8               82           MOV    AL,0A8H                 ;LSB
006B EE                 83           OUT    DX,AL
006C B061               84           MOV    AL,61H                  ;MSB
006E EE                 85           OUT    DX,AL
006F BA0CFF             86           MOV    DX,0FF0CH               ;LOAD COUNTER 2 (2SEC)
0072 B000               87           MOV    AL,00H                  ;LSB
0074 EE                 88           OUT    DX,AL
0075 B002               89           MOV    AL,02H                  ;MSB
0077 EE                 90           OUT    DX,AL
0078 BA16FF             91   SET532: MOV    DX,0FF16H               ;8253 #2 CONTROL WORD
007B B03B               92           MOV    AL,3BH                  ;COUNTER 0, MODE 5,BCD
007D EE                 93           OUT    DX,AL
007E B07B               94           MOV    AL,7BH                  ;COUNTER 1, MODE 5, BCD
0080 EE                 95           OUT    DX,AL
0081 B0BB               96           MOV    AL,0BBH                 ;COUNTER 2, MODE 5, BCD
0083 EE                 97           OUT    DX,AL
0084 BA10FF             98           MOV    DX,0FF10H               ;LOAD COUNTER 0 (.5SEC)
0087 B050               99           MOV    AL,50H                  ;LSB
0089 EE                100           OUT    DX,AL
008A B000              101           MOV    AL,00H                  ;MSB
008C EE                102           OUT    DX,AL
008D BA12FF            103           MOV    DX,0FF12H               ;LOAD COUNTER 1 (1SEC)
0090 B000              104           MOV    AL,00H                  ;LSB
```

```
MCS-86 ASSEMBLER     TCI59A                                                                    PAGE    3

LOC  OBJ               LINE   SOURCE

0092 EE                 105           OUT      DX,AL
0093 B001               106           MOV      AL,01H                  ;MSB
0095 EE                 107           OUT      DX,AL
0096 BA14FF             108           MOV      DX,0FF14H               ;LOAD COUNTER 2 (1.5SEC)
0099 B050               109           MOV      AL,50H                  ;LSB
009B EE                 110           OUT      DX,AL
009C B001               111           MOV      AL,01H                  ;MSB
009E EE                 112           OUT      DX,AL
                        113   ;
                        114   ;                8259A INITIALIZATION
                        115   ;
009F BA00FF             116   SET59A: MOV      DX,0FF00H               ;8259A A0=0
00A2 B013               117           MOV      AL,13H                  ;ICW1-LTIM=0,S=1,IC4=1
00A4 EE                 118           OUT      DX,AL
00A5 BA02FF             119           MOV      DX,0FF02H               ;8259A A0=1
00A8 B048               120           MOV      AL, 48H                 ;ICW2-INTERRUPT TYPE 72 (120H)
00AA EE                 121           OUT      DX,AL
00AB B003               122           MOV      AL, 03H                 ;ICW4-SFNM=0,BUF=0,AEOI=1,MPM=
00AD EE                 123           OUT      DX,AL
00AE B0E0               124           MOV      AL,0E0H                 ;OCW1-MASK IR5,6,7 (NOT USED)
00B0 EE                 125           OUT      DX,AL
                        126   ;
                        127   ;                8279 INITIALIZATION
                        128   ;
00B1 BAEAFF             129   SET79:  MOV      DX,0FFEAH               ;8279 COMMAND WORDS AND STATUS
00B4 B0D0               130           MOV      AL,0D0H                 ;CLEAR DISPLAY
00B6 EE                 131           OUT      DX,AL
00B7 EC                 132   WAIT79: IN       AL,DX                   ;READ STATUS
00B8 D0C0               133           ROL      AL,1                    ;"DU" BIT TO CARRY
00BA 72FB               134           JB       WAIT79                  ;JUMP IF DISPLAY IS UNAVAILABLE
00BC B087               135           MOV      AL, 87H                 ;DIGIT 8
00BE EE                 136           OUT      DX,AL
00BF BAE8FF             137           MOV      DX,0FFE8H               ;8279 DATA WORD
00C2 B006               138           MOV      AL,06H                  ;CHARACTER "1"
00C4 EE                 139           OUT      DX,AL
00C5 BAEAFF             140           MOV      DX,0FFEAH               ;8279 COMMAND WORD
00C8 B086               141           MOV      AL,86H                  ;DIGIT 7
00CA EE                 142           OUT      DX,AL
00CB BAE8FF             143           MOV      DX,0FFE8H               ;8279 DATA WORD
00CE B050               144           MOV      AL,50H                  ;CHARACTER "R"
00D0 EE                 145           OUT      DX,AL
00D1 FB                 146           STI                              ;ENABLE INTERRUPTS
                        147   ;
                        148   ;
                        149   ;                DUMMY PROGRAM
                        150   ;
00D2 EBFE               151   DUMMY:  JMP      DUMMY                   ;WAIT FOR INTERRUPT
                        152   ;
                        153   ;
00D4 A30200             154   SAVE:   MOV      AXTEMP,AX               ;SAVE AX
00D7 58                 155           POP      AX                      ;POP CALL RETURN ADDRESS
00D8 A30000             156           MOV      STACK1,AX               ;SAVE CALL RETURN ADDRESS
00DB A10200             157           MOV      AX,AXTEMP               ;RESTORE AX
00DE 50                 158           PUSH     AX                      ;SAVE PROCESSOR STATUS
00DF 53                 159           PUSH     BX
```

```
MCS-86 ASSEMBLER    TCI59A                                                                    PAGE   4

LOC  OBJ                LINE   SOURCE

00E0 51                  160           PUSH   CX
00E1 52                  161           PUSH   DX
00E2 55                  162           PUSH   BP
00E3 56                  163           PUSH   SI
00E4 57                  164           PUSH   DI
00E5 1E                  165           PUSH   DS
00E6 06                  166           PUSH   ES
00E7 A10000              167           MOV    AX,STACK1             ;RESTORE CALL RETURN ADDRESS
00EA 50                  168           PUSH   AX                    ;PUSH CALL RETURN ADDRESS
00EB C3                  169           RET
                         170   ;
00EC 58                  171   RESTOR: POP    AX                    ;POP CALL RETURN ADDRESS
00ED A30000              172           MOV    STACK1,AX             ;SAVE CALL RETURN ADDRESS
00F0 07                  173           POP    ES                    ;RESTORE PROCESSOR STATUS
00F1 1F                  174           POP    DS
00F2 5F                  175           POP    DI
00F3 5E                  176           POP    SI
00F4 5D                  177           POP    BP
00F5 5A                  178           POP    DX
00F6 59                  179           POP    CX
00F7 5B                  180           POP    BX
00F8 58                  181           POP    AX
00F9 A30200              182           MOV    AXTEMP,AX             ;SAVE AX
00FC A10000              183           MOV    AX,STACK1             ;RESTORE CALL RETURN ADDRESS
00FF 50                  184           PUSH   AX                    ;PUSH CALL RETURN ADDRESS
0100 A10200              185           MOV    AX,AXTEMP             ;RESTORE AX
0103 C3                  186           RET
                         187   ;
                         188   ;
                         189   ;              INTERRUPT 72, CLEAR DISPLAY,IR0 8259A
                         190   ;
0104 E8CDFF              191   INTR72: CALL   SAVE                  ;ROUTINE TO SAVE PROCESSOR STATUS
0107 BAEAFF              192           MOV    DX,0FFEAH             ;8279 COMMAND WORD
010A A00400              193           MOV    AL,DIGIT              ;SELECTED LED DIGIT
010D EE                  194           OUT    DX,AL
010E BAE8FF              195           MOV    DX,0FFE8H             ;8279 DATA
0111 B000                196           MOV    AL,00H                ;BLANK OUT DIGIT
0113 EE                  197           OUT    DX,AL
0114 E8D5FF              198           CALL   RESTOR                ;ROUTINE TO RESTORE PROCESSOR STATUS
0117 CF                  199           IRET                         ;RETURN FROM INTERRUPT
                         200   ;
                         201   ;
                         202   ;              INTERRUPT 73, IR1 8259A
                         203   ;
0118 E8B9FF              204   INTR73: CALL   SAVE                  ;ROUTINE TO SAVE PROCESSOR STATUS
011B BAEAFF              205           MOV    DX,0FFEAH             ;8279 COMMAND WORD
011E B080                206           MOV    AL,80H                ;LED DISPLAY DIGIT 1
0120 A20400              207           MOV    DIGIT,AL
0123 EE                  208           OUT    DX,AL
0124 BAE8FF              209           MOV    DX,0FFE8H             ;8279 DATA
0127 B006                210           MOV    AL,06H                ;CHARACTER "1"
0129 EE                  211           OUT    DX,AL
012A CD4D                212           INT    77                    ;TIMER DELAY FOR LED ON TIME
012C E8BDFF              213           CALL   RESTOR                ;ROUTINE TO RESTORE PROCESSOR STATUS
012F CF                  214           IRET                         ;RETURN FROM INTERRUPT
```

```
MCS-86 ASSEMBLER    TCI59A                                                                   PAGE    5

LOC  OBJ              LINE   SOURCE

                       215   ;
                       216   ;
                       217   ;                  INTERRUPT 74, IR2 8259A
                       218   ;
0130 E8A1FF            219   INTR74: CALL      SAVE                     ;ROUTINE TO SAVE PROCESSOR STATUS
0133 BAEAFF            220           MOV       DX,0FFEAH                ;8279 COMMAND WORD
0136 B081              221           MOV       AL,81H                   ;LED DISPLAY DIGIT 2
0138 A20400            222           MOV       DIGIT,AL
013B EE                223           OUT       DX,AL
013C BAE8FF            224           MOV       DX,0FFE8H                ;8279 DATA
013F B05B              225           MOV       AL,5BH                   ;CHARACTER "2"
0141 EE                226           OUT       DX,AL
0142 CD4D              227           INT       77                       ;TIMER DELAY FOR LED ON TIME
0144 E8A5FF            228           CALL      RESTOR                   ;ROUTINE TO RESTORE PROCESSOR STATUS
0147 CF                229           IRET                               ;RETURN FROM INTERRUPT
                       230   ;
                       231   ;
                       232   ;                  INTERRUPT 75, IR3 8259A
                       233   ;
0148 E889FF            234   INTR75: CALL      SAVE                     ;ROUTINE TO SAVE PROCESSOR STATUS
014B BAEAFF            235           MOV       DX,0FFEAH                ;8279 COMMAND WORD
014E B082              236           MOV       AL,82H                   ;LED DISPLAY DIGIT 3
0150 A20400            237           MOV       DIGIT,AL
0153 EE                238           OUT       DX,AL
0154 BAE8FF            239           MOV       DX,0FFE8H                ;8279 DATA
0157 B04F              240           MOV       AL,4FH                   ;CHARACTER "3"
0159 EE                241           OUT       DX,AL
015A CD4D              242           INT       77                       ;TIMER DELAY FOR LED ON TIME
015C E88DFF            243           CALL      RESTOR                   ;ROUTINE TO RESTORE PROCESSOR STATUS
015F CF                244           IRET                               ;RETURN FROM INTERRUPT
                       245   ;
                       246   ;
                       247   ;                  INTERRUPT 76, IR4 8259A
                       248   ;
0160 E871FF            249   INTR76: CALL      SAVE                     ;ROUTINE TO SAVE PROCESSOR STATUS
0163 BAEAFF            250           MOV       DX,0FFEAH                ;8279 COMMAND WORD
0166 B083              251           MOV       AL,83H                   ;LED DISPLAY DIGIT 4
0168 A20400            252           MOV       DIGIT,AL
016B EE                253           OUT       DX,AL
016C BAE8FF            254           MOV       DX,0FFE8H                ;8279 DATA
016F B066              255           MOV       AL,66H                   ;CHARACTER "4"
0171 EE                256           OUT       DX,AL
0172 CD4D              257           INT       77                       ;TIMER DELAY FOR LED ON TIME
0174 E875FF            258           CALL      RESTOR                   ;ROUTINE TO RESTORE PROCESSOR STATUS
0177 CF                259           IRET                               ;RETURN FROM INTERRUPT
                       260   ;
                       261   ;
                       262   ;                  INTERRUPT 77, TIMER DELAY, SOFTWARE CONTROLLED
                       263   ;
0178 BA0AFF            264   INTR77: MOV       DX,0FF0AH                ;LOAD COUNTER 1 8253 #1 (250 MSEC)
017B B025              265           MOV       AL,25H                   ;LSB
017D EE                266           OUT       DX,AL
017E B000              267           MOV       AL,00H                   ;MSB
0180 EE                268           OUT       DX,AL
0181 CF                269           IRET                               ;RETURN FROM INTERRUPT
```

```
MCS-86 ASSEMBLER    TCI59A                                                    PAGE    6


LOC  OBJ                  LINE   SOURCE

                           270   ;
                           271   ;
----                       272   CODE    ENDS;
                           273   ;
                           274   ;
 0000                      275           END      START

SYMBOL TABLE LISTING
------ ----- -------


NAME     TYPE     VALUE  ATTRIBUTES

??SEG .  SEGMENT         SIZE=0000H PARA PUBLIC
AXTEMP.  V WORD   0002H  DATA
CODE. .  SEGMENT         SIZE=0182H PARA
DATA. .  SEGMENT         SIZE=0005H PARA
DIGIT .  V BYTE   0004H  DATA
DUMMY .  L NEAR   00D2H  CODE
EXTRA .  SEGMENT         SIZE=0138H PARA
INTR72.  L NEAR   0104H  CODE
INTR73.  L NEAR   0118H  CODE
INTR74.  L NEAR   0130H  CODE
INTR75.  L NEAR   0148H  CODE
INTR76.  L NEAR   0160H  CODE
INTR77.  L NEAR   0178H  CODE
RESTOR.  L NEAR   00ECH  CODE
SAVE. .  L NEAR   0004H  CODE
SET531.  L NEAR   005AH  CODE
SET532.  L NEAR   0078H  CODE
SET59A.  L NEAR   009FH  CODE
SET79 .  L NEAR   00B1H  CODE
STACK1.  V WORD   0000H  DATA
START .  L NEAR   0000H  CODE
TP72CS.  V WORD   0122H  EXTRA
TP72IP.  V WORD   0120H  EXTRA
TP73CS.  V WORD   0126H  EXTRA
TP73IP.  V WORD   0124H  EXTRA
TP74CS.  V WORD   012AH  EXTRA
TP74IP.  V WORD   0128H  EXTRA
TP75CS.  V WORD   012EH  EXTRA
TP75IP.  V WORD   012CH  EXTRA
TP76CS.  V WORD   0132H  EXTRA
TP76IP.  V WORD   0130H  EXTRA
TP77CS.  V WORD   0136H  EXTRA
TP77IP.  V WORD   0134H  EXTRA
TYPES .  L NEAR   0012H  CODE
WAIT79.  L NEAR   00B7H  CODE


ASSEMBLY COMPLETE, NO ERRORS FOUND
```