intel®

APPLICATION NOTE

AP-31

Using The 8259 Programmable Interrupt Controller

John Beaston
Microcomputer Applications

$1.00

Related Documents

"Intel8080 Microcomputer Systems User's Manual"

"SBC 80/20 Single Board Computer Hardware Reference Manual"

"Intel 8080 Microcomputer Peripherals User's Manual"

"Intel Data Catalog"

The material in this Application Note is for informational purposes only and is subject to change without notice. Intel Corporation has made an effort to verify that the material in this document is correct. However, Intel Corporation does not assume any responsibility for errors that may appear in this document.

The following are trademarks of Intel Corporation and may be used only to describe Intel Products:

ICE
INSITE
INTEL
INTELLEC
LIBRARY MANAGER
MCS
MEGACHASSIS
MICROAMP
PROMPT
UPI

INTEL CORPORATION ASSUMES NO RESPONSIBILITY FOR THE USE OF ANY CIRCUITRY OTHER THAN CIRCUITRY EMBODIED IN AN INTEL PRODUCT. NO OTHER CIRCUIT PATENT LICENSES ARE IMPLIED.

Using The 8259 Programmable Interrupt Controller

Contents

INTRODUCTION

The Intel® 8259 is a Programmable Interrupt Controller (PIC) designed for use in real-time, interrupt-driven microcomputer systems. The 8259 manages eight levels of interrupts and has built-in features allowing expandability up to 64 levels with the addition of other 8259s. A selection of programmable priority modes is available to reconfigure how the 8259 processes interrupt requests. Individual interrupt inputs may also be masked under software control. These modes and masks may be dynamically changed by the software at any time during program execution. This means that the complete interrupt structure can be defined as required, based on the total system environment. The 8259 is part of the MCS-80/85 Microcomputer Family and as such, it interfaces to the 8080/8085 system with a minimum of external hardware.

This application note explains the 8259 as a component and shows its use in two typical applications. These applications are an interrupt controlled power-faillauto-start scheme for a microcomputer system with battery back-up RAM, and a >64 level interrupt-driven system. The battery back-up system will be described in detail and the conceptual software for the >64 level interrupt-driven system will be presented.

The first section of this application note introduces the concept of interrupts and reviews how interrupts are handled by the Intel' 8080A Microprocessor. It is fairly tutorial in nature, and may be skipped by the morc knowledgeable reader. The second section describes the 8259 from a functional standpoint with explanation of the block diagram. Each device pin is explained in detail. The third section defines the various operating modes along with the specific software required. Short initialization and setup routines are given to illustrate the programming concepts. The fourth, and final, section describes the applications mentioned earlier.

CONCEPTS

In microcomputer systems, there is usually a need for the processor to communicate with various Input/Output 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-wisc. Overall, the Polled method of I/O servicing can have a detrimental effect on system throughput, thus limiting the tasks that could be performed by the processor.

A morc 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. In effect, the device would asynchronously signal the processor when it required service. The processor would finish its current instruction and then jump to the service routine for the device requesting service. Once the service routine is complete, the processor would resume exactly where it left off in the main program.

This method of I/O servicing is called Interrupt. The status line of the peripheral is replaced by an "interrupt request" line. Asserting this line signals the processor that service is needed. Using interrupts, 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 approach, system throughput would increase, allowing more tasks to be handled by the processor.

There arc two basic methods of implementing the Interrupt approach: polled interrupts and vectored interrupts. Conceptually, in the polled interrupt method, the peripherals' "interrupt request" lines are combinatorially OR'd into one line that interrupts the processor if any peripheral required service. The processor then polls each peripheral to determine the requesting device. In this scheme, the priority of tlie device is determined by its position in the polling sequence. Once the requesting device is found, the processor branches to the corresponding service routine. In contrast, vectorcd interrupts are those in which the requesting device supplies information which allows the processor to directly call the appropriate service routine. This method usually requires more hardware than the polled method. However. it allows much faster response to an interrupt since the polling time is eliminated. In simple vectored interrupt systems, all devices have the same priority. This is sometimes a limitation since the speed of the vectored method may be needed, while the prioritization of the polled method is also required; a flexible interrupt structure would have both.

In order to implement a truly flexible priority-vectorcd interrupt structure, a Programmable Interrupt Controller (PIC), such as the 8259, may be used. The 8259 functions as the overall manager of the interrupt-driven system and can implement both the polled and vectored interrupt structures. In the vectored structure it accepts interrupt requests from the peripherals, determines which of the incoming requests is the highest priority, ascertains whether the highest priority incoming request has higher priority than the interrupt level currently being serviced (if any) and then issues an interrupt to the processor based on the determination. Since each peripheral usually has a unique service routine associated with it, the PIC, after interrupting the processor, provides a "vectored" CALL instruction to point the processor directly to the servicc routine required by the interrupting device. In the polled structure, the same request priority determination is made, however software polls the 8259 rather than the peripherals. When polled, the 8259 returns a data word indicating the highest priority peripheral requesting service. The software then uses this data word to branch to the appropriate service routine.

A variety of priority modes is a desirable feature of a PIC. Many options are conceivable; however, let's describe a few which are available with the 8259 and will be mentioned later.

Fully *Nested* – Each input is assigned a priority. Interrupt Request input IR7 receives the lowest priority while IRO receives the highest. A higher priority request will interrupt a lower priority service routine, but not vice versa. The lower priority service routine will be resumed upon completion of the higher priority routine. This is essentially a "general purpose" mode.

Rotating ***Priority*** – Like in the Fully Nested mode, each input is assigned a priority. However, when an interrupt occurs and the appropriatc service routine is executed, the priorities are rotated so that the most recently serviced input has the lowest priority. Thus, if there are N inputs, a serviced peripheral will have to wait, in the worst case, until the other N–1 peripherals are serviced before receiving service again. This mode prevents "hogging" of the processor by a single peripheral and gives each input an equal chance at the processor.

Specific *Priority* – This mode is similiar to the Rotating mode. The only difference is that the software can select the bottom priority input without an interrupt having to have occurred. Thus, the priority assignments may be changed at any time depending on the needs of the main program or the service routine.

In the 8259, these modes are programmable; that is, they may be changed dynamically under software control. Additionally, each mode may be modified by the use of interrupt masks. These masks allow individual inputs to be masked off; i.e., not be able to cause an interrupt regardless of its priority. Each mask is under software control.

Before we discuss how the 8259 handles interrupts, let's digress slightly to review how the 8080 itself handles interrupt requests.

8080 INTERRUPTS

A peripheral device can initiate an interrupt to the 8080 by simply pulling the 8080's Interrupt pin

(INT) high. The INT line is asynchronous, therefore an interrupt request may be asserted at any time. The 8080 can, however, enable and disablc interrupts under software control by use of the Enable Interrupt (EI) and Disable Interrupt (DI) instructions. These instructions either set (EI) or reset (DI) an internal interrupt enable flip-flop. The output of this flip-flop is made available on the INTE (Interrupt Enabled) pin. Interrupts are disabled (INTE low) upon resetting the 8080.

At the end of each instruction cycle, thc 8080 examines the state of the INT pin and the INTE flip-flop. If interrupts are enabled and an interrupt request is being made (both pins high), the 8080 enters an INTERRUPT machine cycle. During the INTERRUPT cycle, the 8080 resets the interrupt enable flip-flop (INTE goes low disabling response to further interrupts) and issues an Interrupt Acknowledge ($\overline{\text{INTA}}$), by way of the System Controller 8228, to tell the interrupting device that it has the 8080's attention and may remove the INT assertion. In addition, the Program Counter (PC) is not incremented as it normally would be in normal machine cycles. This ensures that the 8080 can return to the pre-interrupt program location if the PC is saved. At this point, the 8080 expects the interrupting device to place an instruction on the data bus. The 8080 is, in effect, saying "Okay, now you have my attention. You are granted one wish. What will it be?" Any instruction may be used, but there are only two logical choices: a RESTART (RST) or a CALL. The reason one of these two should be used is that both put the program counter on the stack, allowing it to be restored after the interrupt service routine is complete.

When a CALL instruction is placed on the data bus in response to the Interrupt Acknowledge ($\overline{\text{INTA}}$), the 8080 saves the program counter by pushing it onto the stack and then issues two additional $\overline{\text{INTA}}$s by way of the 8228. In response, the interrupting device is expected to return two bytes which are the starting address of its service routine. The lower 8 bits of the address (LSB) are released at the first INTA and the higher 8-bits (MSB) are released at the second $\overline{\text{INTA}}$. Execution then starts at this destination address. Using a CALL instruction in response to an interrupt is an extremely powerful tool in I/O servicing. However, a significant amount of hardware is usually required in order to ensure that the correct sequence of data is placed on the data bus. For systems not having a large number of peripherals, a special CALL instruction is provided in the 8080 instruction set.

The RESTART (RST) instructions are actually special one-byte calls which have the destination address embedded within the 8-bit opcode. Executing an RST causes execution to be transferred (vectored) to one of eight fixed memory locations, see Figure 1. Any of these addresses may bc used to store the first instructions of an interrupt service routine. In simple systems, the desired RST instruction can be generated by a simple 8-bit buffer external to the interrupting device. Since the RST instructions are calls, the old program counter contents are placed on the stack.

RST	HEX OP CODE	DESTINATION ADDRESS
RST 0	C7	00 H
RST 1	CF	08 H
RST 2	D7	10 H
RST 3	DF	18 H
RST 4	E7	20 H
RST 5	EF	28 H
RST 6	F7	30 H
RST 7	FF	38 H

Figure 1. RST Instruction Format

Return to the main program from an interrupt service routine is identical for both the CALL and the RST instructions. Assuming an equal number of pushes and pops from the stack during the service routine, the pre-interrupt program counter is on top of thc stack at the end of the routine. Executing a RETURN (RET) instruction pops the top of the stack into the program counter, causing the main program to take up whcrc it left off before receiving the interrupt. It is the service routine's responsibility to save and restore thc processor registers and status as appropriate. Remcmber that interrupts are disabled after an Interrupt Acknowledge so an EI instruction must be executed in the service routine in order for the 8080 to respond to further interrupt requests.

8259—8080 OVERVIEW

Figure 2 shows the 8259—8080 system bus interface. It is recommended that an 8228 (or 8238)

System Controller and Bus Driver be used in conjunction with the 8080 when an 8259 is used to manage interrupts. This combination ensures that the 3 required INTA pulses occur in response to an interrupt. Using the 8212 I/O Port as an 8080 status latch does not provide the necessary INTA sequence.

The normal sequence of events that occur when an interrupt request is asserted is as follows:

1. One or more Interrupt Request lines (IR0–IR7) is raised high signaling the 8259 that peripheral service is being requested.
2. The 8259 accepts the requests, resolves the priorities, and sends an INT to the 8080.
3. The 8080 suspends the program flow at the end of the current instruction (INTE must be high), and issues an INTA by way of the 8228.
4. Upon receiving the INTA, the 8259 places a CALL instruction onto the data bus.
5. This CALL causes the 8080 to issue two additional INTAs by way of the 8228.
6. These additional INTAs allow the 8259 to release the address for the service routine of the interrupting peripheral onto the bus.
7. This completes the 3-byte CALL. Execution is vectored to the peripheral's service routine.

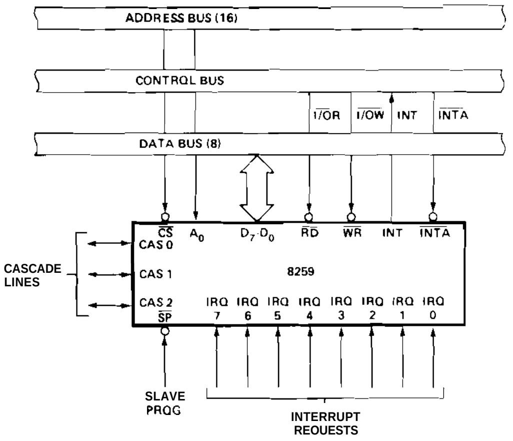
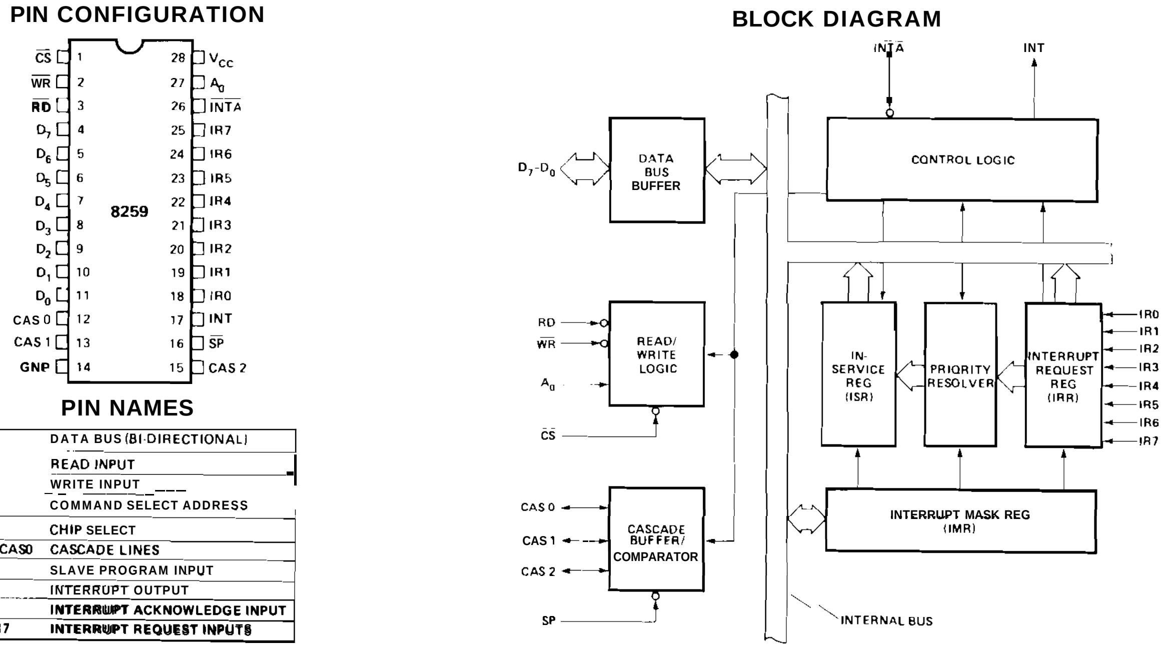


Figure 2. 8259 Interface to 8080 Standard System Bus

8259 BLOCK DIAGRAM

A block diagram of the 8259 is shown in Figure 3. As can be seen from the figure, the 8259 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 go quickly over the individual blocks directly related to interrupt handling; the IRR, ISR, IMR. PR, and the Control logic. Then. by way of a conceptual diagram, we show how these various blocks interact. The remaining functional blocks are then discussed.



D7-D0	DATA BUS (BI-DIRECTIONAL)
RD	READ INPUT
WR	WRITE INPUT
A0	COMMAND SELECT ADDRESS
CS	CHIP SELECT
CAS1-CAS0	CASCADE LINES
SP	SLAVE PROGRAM INPUT
INT	INTERRUPT OUTPUT
INTA	INTERRUPT ACKNOWLEDGE INPUT
IR0-IR7	INTERRUPT REQUEST INPUTS

Figure 3. Block Diagram and Pin Configuration

Basically, interrupt requests are handled by three "cascaded" registers. The Interrupt Request Register (IRR) is used to store all the interrupt levels requesting service; the In-Service Register (ISR) stores all the levels which are being serviced; and the Interrupt Mask Register (IMR) stores the bits of the interrupt lines to be masked. The Priority Resolver (PR) looks at the IRR, ISR, and IMR and determines whether an INT should be issued by the Control logic to the 8080.

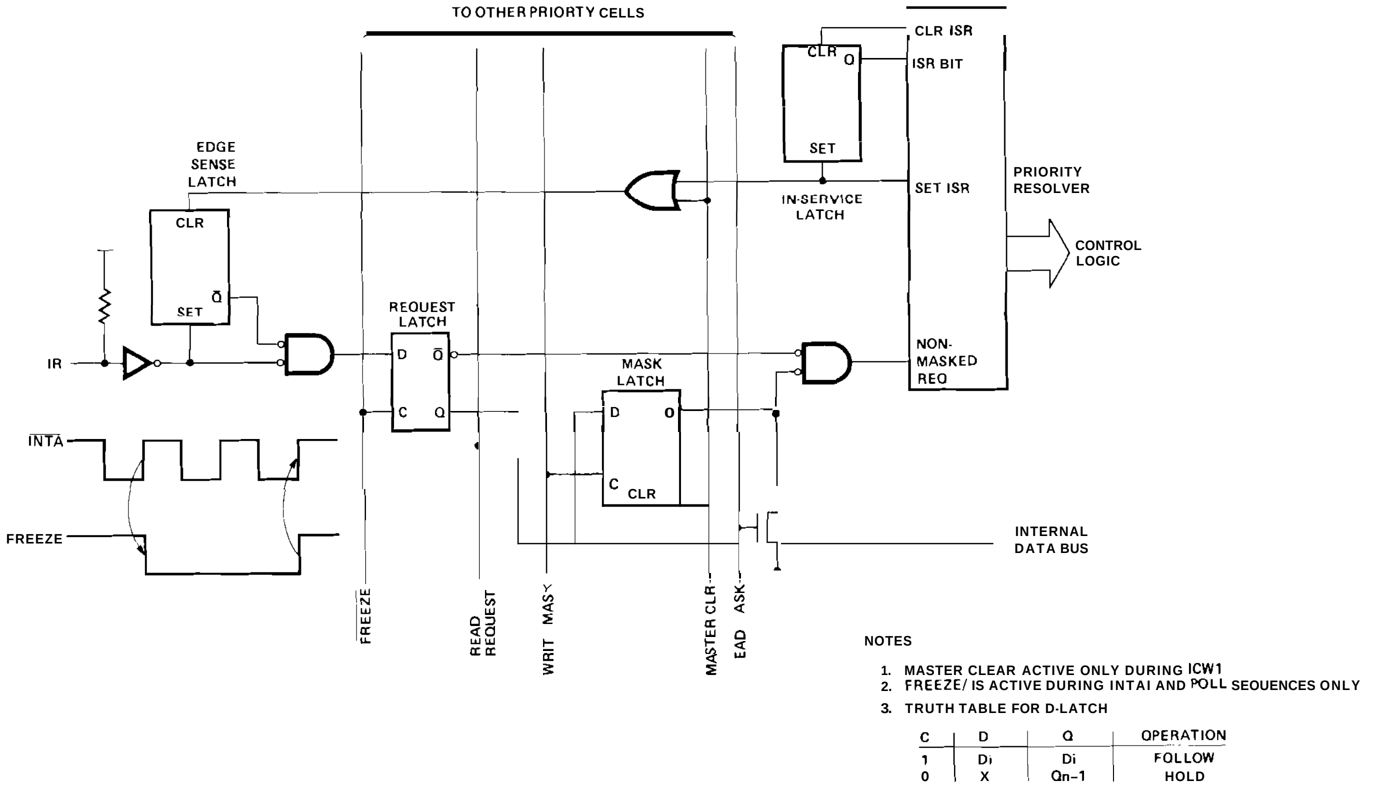
Figure 4 shows conceptually how the Interrupt Request (IR) input is handled and how the various registers interact. The figure represents one of eight "daisy-chained" priority cells; one for each IR input. The input circuitry is rather novel so it is discussed first.

INPUT CIRCUIT

There are two classical ways of sensing an active interrupt request: a level sensitive or an edge sensitive input. A level sensitive input requires the request input go to the active state and remain active until that interrupt is acknowledged. This structure is quite common and allows WIRE-OR'ed interrupt requests (the actual interrupting device must be determined via software as mentioned before). But (watch out!) the request must be removed shortly after acknowledgement or another, unwanted, interrupt could be generated.

The edge sensitive input requires only an inactive to active transition of the request input. This transition is saved in a flip-flop, so the active level need be maintained only long enough to serve as a clock pulse to the flip-flop. The level may remain active an arbitrarily long time without danger of generating an unwanted interrupt. It must ultimately return inactive before another active transition can be sensed. This structure is handy for handling interrupts from transient events, however it prevents WIRE-OR'ing since this connection does not provide the transitions needed. Be careful of edge inputs; noise on the request line could generate an erroneous interrupt.

The 8259 uses an edge lockout input which shares some characteristics with each of the above two techniques. The edge lockout input requires that a request transition from the inactive to the active



NOTES

1. MASTER CLEAR ACTIVE ONLY DURING ICW1
2. FREEZE/ IS ACTIVE DURING INTAI AND POLL SEOUENCES ONLY
3. TRUTH TABLE FOR D-LATCH

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

Figure 4. Priority Cell

state (as in edge sensitive) and then remain active (as in level sensitive) until the request is acknowledged. The inactive-to-active transition locks out all further requests on that input until the request has been acknowledged and the input has returned to the inactive state. Thus, the user need not worry about quickly removing the request after acknowledgement, in fear of generating a second interrupt. Figure 5 illustrates the timing required for the edge lockout input.

PRIORITY CELL

Refer back to Figure 4 and follow an interrupt request thru the priority cell. First, notice that an inactive IR input sets the edge sense latch, arming that input. Then, an active IR input combinatorially propagates the request (assuming the input is not masked) to the Priority Resolver. The PR looks at the incoming requests and the currently in-service interrupts to ascertain whether an interrupt should be issued to the 8080. Assume for clarity 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 8080 high, interrupting the processor. When the 8080 is finished with the instruction being executed, it signals the 8228 to return an $\overline{\text{INTA}}$. This $\overline{\text{INTA}}$ causes the 8259 to place a CALL instruction on the data bus and to freeze the IRR (note the INTA-Freeze Request timing diagram). Thus. the requesting IR input must remain active at least until after the first $\overline{\text{INTA}}$. With the input frozen and latched, the priority is again resolved by the PR, this time to determine the appropriate destination address for the CALL. The CALL instruction causes the 8080 to generate two additional $\overline{\text{INTA}}$s. During these $\overline{\text{INTA}}$s the destination address of the interrupt service routine is placed on the data bus by the 8259. (Don't worry for now about where the address comes from.) Immediately after the $\overline{\text{INTA}}$ sequence, the PR then sets the corresponding bit in the ISR and simultaneously clears the edge sense latch, which clears the IRR bit. Notice the state of the edge sense latch (don't forget that the IR input may still be active). With the edge sense latch cleared, the still active IR input can not propagate thru the gate at the IRR input, thus further requests from this level are inhibited. The IR input must return to the inactive state, setting the edge sense latch and "opening" the IRR gate, before another request on the input can be recognized.

While off in the interrupt service routine, don't forget that the ISR bit is set. This prevents subsequent requests from this, and lower priority levels, from causing interrupts. It is the service routine's responsibility to clear the ISR bit with an End-of-Interrupt (EOI) command at the end of the service routine, telling the 8259 that it is complete. (How this is done is explained when 8259 programming is covered.)

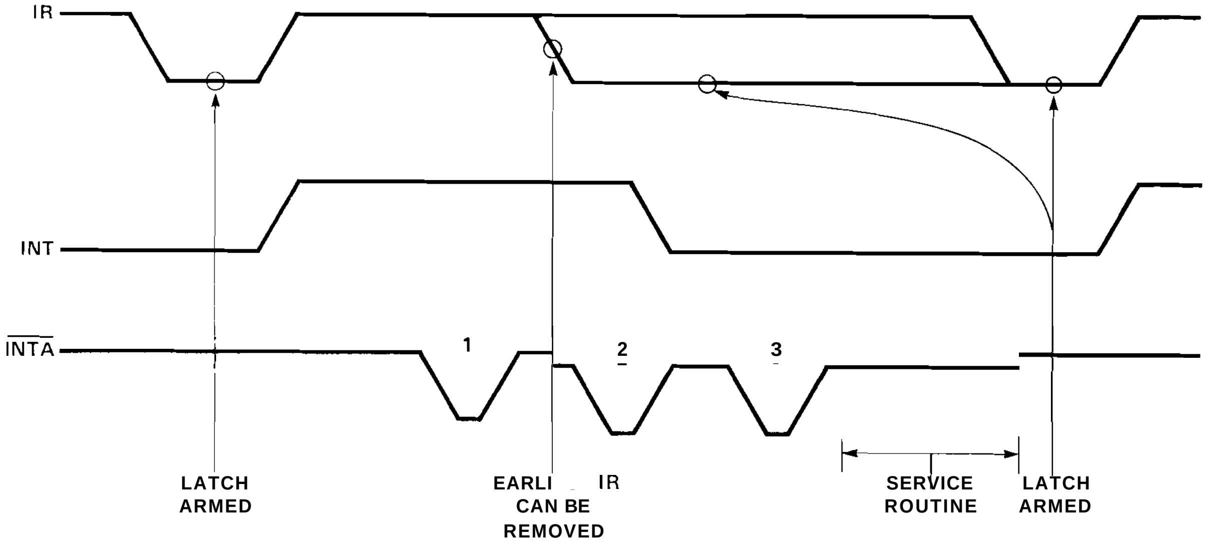


Figure 5. Edge Lockout Timing

What would have happened if the input had been masked; i.e., the Interrupt Mask Register bit was set? Nothing. The active state of the IR input would propagate thru the IRR but the set IMR bit would stop it before entering the PR. Thus, no interrupt could be generated. The IMR only acts on the output of the IRR, however, and if the program being executed somehow resets the IMR bit, the PR would then see our active request and an interrupt would be generated if appropriate.

Now that the functional blocks directly related to interrupt request processing have been discussed, let us discuss the remaining blocks.

DATA BUS BUFFER

This 3-state, bidirectional, 8-bit buffer is used to interface the 8259 to the 8080 system data bus. Conlrol words, status information, and the destination addresses are transferred through the Data Bus Buffer.

READ/WRITE CONTROL LOGIC

The function of this block is to control the programming of the 8259 by accepting OUTput commands from the 8080. The Initialization and Operation Command Word Registers which store the various control formats are located in this block. Status reads are also controlled by this block using 8080 INput commands.

CASCADE BUFFER/COMPARATOR

As alluded to earlier, multiple 8259s can be combined to expand the number of interrupt levels. A master-with-slaves relationship of cascaded 8259s is used for the expansion. The cascading of 8259s will be the subject of a complete section later in this note.

PIN DEFINITIONS

Name (pin)	I/O	Definition
V_{CC} (28)	I	+5 volt supply
GND (14)	I	Ground
$\overline{CS}$ (1)	I	*Chip Select.* A low on this pin enables communication between the. CPU and the PIC.
$\overline{WR}$ (2)	I	A low on this input when $\overline{CS}$ is low enables the PIC to accept command words from the CPU.
$\overline{RD}$ (3)	I	A low on this input causes the PIC to output its status on the data bus when $\overline{CS}$ is low.
DB_7–DB_0 (4–11)	I/O	The DB pins form a 3-state, bidirectional data bus which is connected to the CPU group (8080, 8224, 8228) data bus. Control and status information are transferred over this bus.
CAS_0–CAS_2 (12,13,15)	I/O	*Cascade Lines.* The CAS pins form a private 8259 bus to control multiple 8259s. These pins are outputs for a master 8259 and are inputs for a slave 8259.
$\overline{SP}$ (16)	I	*Slave Program.* The state of this pin defines whether the 8259 is a master ($\overline{SP}$=1) or a slave ($\overline{SP}$=0). $\overline{SP}$ controls the I/O direction of the CAS pins.
INT (17)	O	*Interrupt.* This pin goes high whenever a valid interrupt request is asserted. INT is connected to the interrupt pin of the CPU.
IR_0–IR_7 (18–25)	I	*Interrupt Request.* Interrupt requests are asserted by the peripherals. A request is made by pulling one of the IR pins high.
$\overline{INTA}$ (26)	I	*Interrupt Acknowledge.* This pin is connected to the CPU group interrupt acknowledge output. Three low pulses on this pin causes the 8259 to place a CALL instruction and a destination address on the DB pins. (One byte for each $\overline{INTA}$ pulse.)
A_0 (27)	I	This pin acts in conjunction with the CS, WR, and RD pins when Command Words are written and status is read from the 8259. It is typically connected to the CPU A_0 address line.

PROGRAMMING THE 8259

As the name implies, the 8259 is programmable; operation is controlled via software thru command words. There are two types of command words used for the 8259: Initialization Command Words (ICWs) and Operation Command Words (OCWs).

INITIALIZATION COMMAND WORDS (ICWs)

Before normal operation begins (i.e., after a system power-up), each 8259 in the system must be initialized by two or three ICWs. The ICWs tell each 8259:

1. If there are other 8259s in the system, and how they are connected.
2. The starting address of the service routines.
3. Whether the service routines are spaced 4 or 8 bytes apart.

Issuing an ICWl starts the 8259 initialization sequence. Once started, the initialization sequence must be completed before the 8259 can process interrupt requests. This applies to each 8259 in a multiple 8259 system. During the initialization sequence, the following occur automatically:

1. Each edge sense circuit is reset. Thus an IR input must make an inactive to active transition, after initialization, to generate an interrupt.
2. The Interrupt Mask Register is reset (no IR inputs masked).
3. IR7 is assigned priority level 7
4. The Status Read and Special Mask mode flip-flops (explained later) are reset.

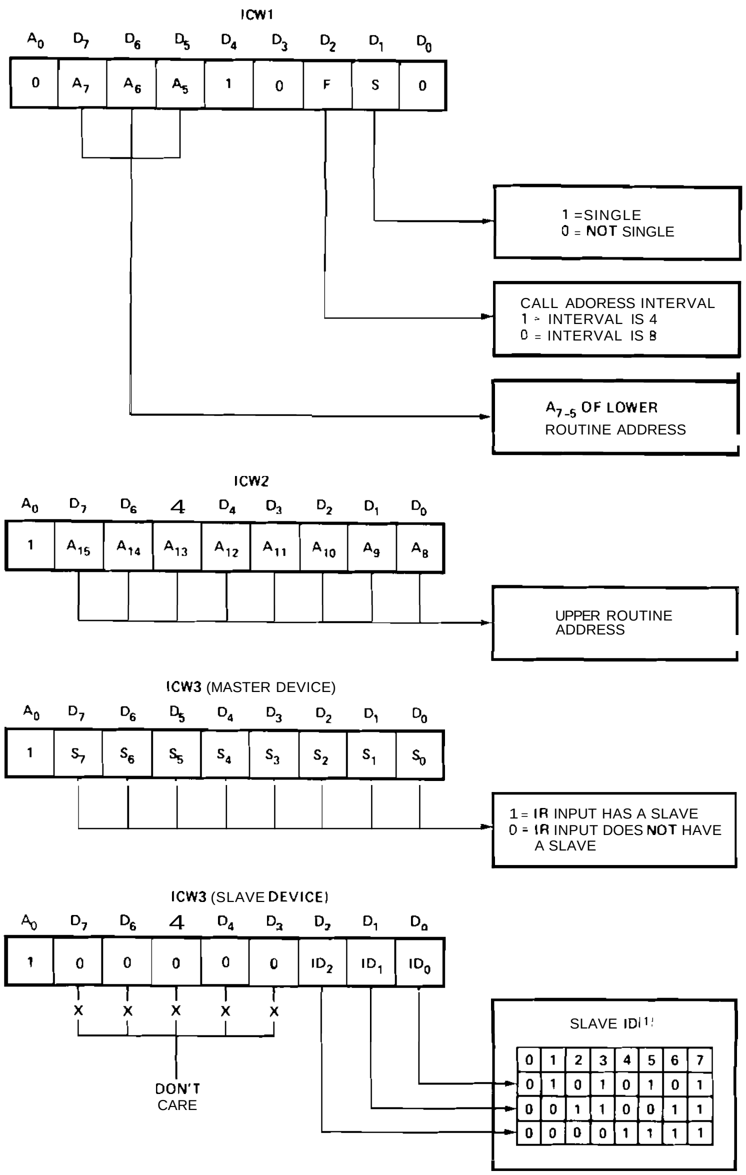
Each IR input has an address in memory associated with it. It is this address that is placed on the bus by the 8259 in response to the INTA pulses after the CALL is placed on the data bus. The addresses for all eight IR inputs are formatted in equally spaced intervals of either 4 or 8 bytes. If the service routine for a device is short, it may be possible to fit the entire routine within an 8-byte interval. Usually, however, the service routines require more than 8 bytes and the 4-byte interval is used to store a Jump (JMP) instruction which directs the 8080 to the appropriate routine. The 8-byte interval maintains compatibility with current 8080 RESTART instructions software, while the 4-byte interval is best for a compact Jump table. For each 8259, the starting address for this 32 or 64-byte page is programmable during initialization and can be located anywhere in the memory map, starting on an even page boundary. To form the 16 bits needed for each address, address bits $A_{15}-A_6$ are user supplied in the ICWs and bits A_4-A_0 are inserted by the 8259. A_5's generation depends upon whether 4 or 8-byte intervals are programmed. For 4-byte intervals, you program A_5 in ICWl. The 8259 supplies A_5 for the 8-byte interval selection. Figure 6 shows how the address is developed for each IR input.

REQUEST INPUT	4 BYTE INTERVAL— A15–A5 SUPPLIED IN ICWI AND ICW2					8-BYTE INTERVAL— A15–A6 SUPPLIED IN ICWI AND ICW2						
	A4	A3	A2	A1	A0	A5	A4	A3	A2	A1	A0	
IR0	0	0	0	0	0	0	0	0	0	0	0	
IR1	0	0	1	0	0	0	0	1	0	1	1	0
IR2	0	1	0	0	0	0	1	0	0	0	0	
IR3	0	1	1	0	0	0	1	1	0	0	0	
IR4	1	0	0	0	0	1	0	0	0	1	1	0
IR5	1	0	1	0	0	1	0	1	0	0	0	
IR6	1	1	0	0	0	1	1	0	0	0	0	
IR7	1	1	1	0	0	1	1	1	0	0	0	

Figure 6. Address Development

The formats for ICW1 and ICW2 are shown in Figure 7. The 8259 interprets any command with A_0=0, D_0=0, and D_4=1 as an ICWI. Note that address bit A_0 is used as an additional control input for all command words. Bits F and S are the only yet undefined bits. Bit F (Format) determines the CALL address interval. If F=1, then addresses are in 4-byte intervals; if F=0, then the interval is 8 bytes. Bit S (Single) indicates if there is more than one 8259 in the system. If S=1, there is only a single 8259; S=O means multiple 8259s. ICW2 simply supplies the MSB of the address used as the start of the service routine page and is sent with A_0=1.

If the system contains multiple 8259s (ICWI bit S=0), an additional ICW is needed: ICW3. This word controls the master-slave relationship to ensure the correct 8259 places the service routine address on the bus. Multiple 8259 systems in general, and ICW3 in particular, are discussed in another section.



NOTE 1. SLAVE ID IS EQUALTOTHE CORRESPONDING MASTER IR INPUT.

Figure 7. Initialization Command Word Format

Figure 8 shows the flow required for initialization. ICW1 is issued first, initiating the sequence. ICW2 must follow as the next command. With a single 8259, no ICW3 is required and the 8259 is ready to process interrupt requests immediately following ICW2. In order to ensure the integrity of any initialization or command sequence, interrupts must be disabled (by executing a DI instruction) over the initialization section of code. (Don't forget that interrupts are disabled automatically after the 8080 is reset.) Two typical initialization sequences are shown in Example 1.

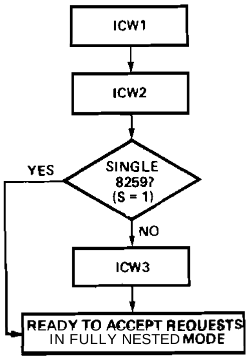
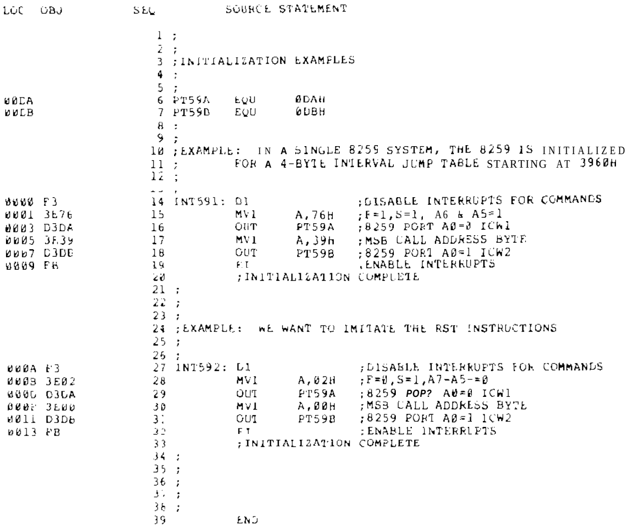


Figure 8. Initialization Flow



```
LOC  OBJ        SEQ        SOURCE STATEMENT

                  1 ;
                  2 ;
                  3 ;INITIALIZATION EXAMPLES
                  4 ;
                  5 ;
ØØDA              6 PT59A    EQU      ØDAH
ØØDB              7 PT59D    EQU      ØDBH
                  8 ;
                  9 ;
                 1Ø ;EXAMPLE:  IN A SINGLE 8259 SYSTEM, THE 8259 IS INITIALIZED
                 11 ;        FOR A 4-BYTE INTERVAL JUMP TABLE STARTING AT 396ØH
                 12 ;
                 13 ;
ØØØØ F3          14 INT591:  DI                ;DISABLE INTERRUPTS FOR COMMANDS
ØØØ1 3E76        15          MVI      A,76H    ;F=1,S=1, A6 & A5=1
ØØØ3 D3DA        16          OUT      PT59A    ;8259 PORT AØ=Ø ICW1
ØØØ5 3E39        17          MVI      A,39H    ;MSB CALL ADDRESS BYTE
ØØØ7 D3DB        18          OUT      PT59B    ;8259 PORT AØ=1 ICW2
ØØØ9 FB          19          EI                ,ENABLE INTERRUPTS
                 2Ø          ;INITIALIZATION COMPLETE
                 21 ;
                 22 ;
                 23 ;
                 24 ;EXAMPLE:  WE WANT TO IMITATE THE RST INSTRUCTIONS
                 25 ;
                 26 ;
ØØØA F3          27 INT592:  DI                ;DISABLE INTERRUPTS FOR COMMANDS
ØØØB 3EØ2        28          MVI      A,Ø2H    ;F=Ø,S=1,A7-A5-=Ø
ØØØD D3DA        29          OUT      PT59A    ;8259 POP? AØ=Ø ICW1
ØØØF 3EØØ        3Ø          MVI      A,ØØH    ;MSB CALL ADDRESS BYTE
ØØ11 D3DB        31          OUT      PT59B    ;8259 PORT AØ=1 ICW2
ØØ13 FB          32          EI                ;ENABLE INTERRUPTS
                 33          ;INITIALIZATION COMPLETE
                 34 ;
                 35 ;
                 36 ;
                 37 ;
                 38 ;
                 39          END
```

Example 1. Initialization Sequences

Once initialized, the 8259 is controlled using Operation Command Words. These words control the changing of priority modes, interrupt masks, and perform the End-of-Interrupt housekeeping.

OPERATION COMMAND WORDS (OCWs)

After initialization, the 8259 is ready to accept interrupt requests on the IR inputs. However, during operation, the 8259 can be commanded to operate in a variety of priority modes through the Operation Command Words (OCWs). The various modes and their associated OCWs are described below.

Fully Nested Mode

The 8259 handles requests in the Fully Nested mode without any OCW being written. In this mode, the IR inputs are assigned priorities such that IRO has the highest priority while 1R7 has the lowest. When an interrupt is acknowledged, the highest priority request is determined and its address vector is placed on the data bus. In addition, the corresponding bit in the ISR is set. This bit remains set until an End-of-Interrupt command is received by the 8259 from the service routine. While the ISR bit is set, all further requests of the same and lower priority are inhibited from generating an interrupt to the 8080. Higher priority requests can generate an interrupt. However, these interrupts are only acknowledged if the 8080 has enabled interrupts, by executing an EI instruction, since the preceding interrupt. Figure 9 illustrates this point.

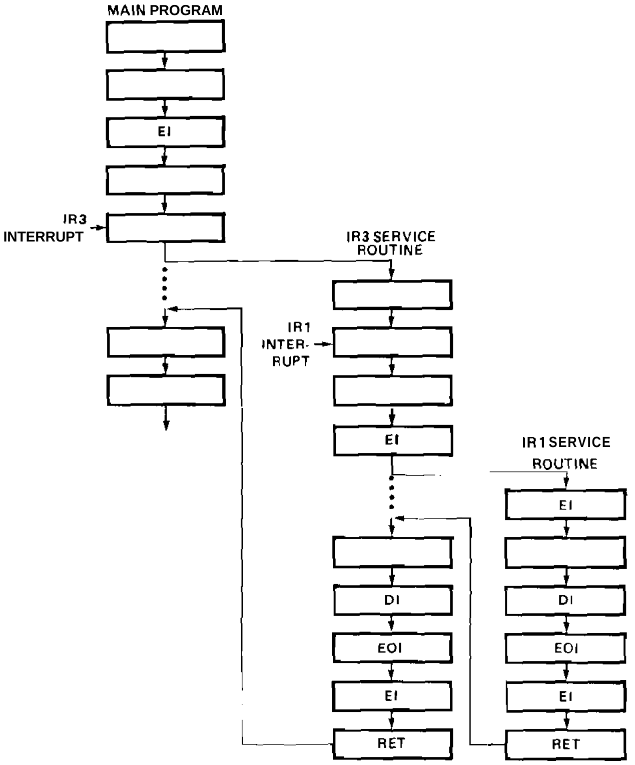
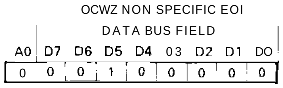


Figure 9. Fully Nested Example

During the main program, IR3 makes a request. Since interrupts are enabled, the 8080 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. Because the 8080 disabled interrupts in response to the IR3 interrupt, the IR1 interrupt is not acknowledged until an EI instruction is executed. Thus the IR3 routine has a "protected" section of code over which no interrupts are allowed. The IR1 routine has no such "protected" section since an EI instruction is the first one in its service routine.

What is happening to the ISR register? While in the main program, no ISR bits are set since no interrupts are in-service. When the 1R3 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 IRO could generate an interrupt since it is the only higher priority input from those presently in-service.

To terminate the IR1 routine, the routine must infonn the 8259 that it is complete by resetting its ISR bit. It does this by executing the EOI command. The format for this command is shown in Figure 10. Note that the format is independent of the interrupt level and is thus called a Non-Specific EOI. The command simply resets the highest priority ISR bit which is set. This is necessarily the correct bit since, in the Fully Nested mode, the highest ISR bit corresponds to the last level acknowledged; which must have been a higher priority than other in-service levels in order to generate the interrupt in the first place.



OCWZ NON SPECIFIC EOI

A0	D7	D6	D5	D4	03	D2	D1	DO
0	0	0	1	0	0	0	0	0

Figure 10. Non-specific EOI Command Format

Getting back to the example, the EOI command for the IR1 routine has been executed, resetting the ISR1 bit.

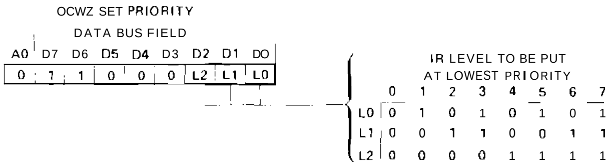
The RET instruction transfers execution back to the IR3 routine. IRO—IR2 could now interrupt the IR3 routine again, since only the IR3 bit in the ISR is set. No further interrupts occur in the example, so the Non-Specific EOI command in the routine resets the ISR3 bit this time and the RET instruction causes the main program to resume at the pre-interrupt location. One important thing to remember: the non-specific EOI command should only be used when in the Fully Nested mode. Other EOI-type commands are used when in other modes. Let us discuss those other modes now.

Rotating Priority Commands

The Rotating Priority Commands serve in applications where the interrupting devices are of equal priority such as communication channels. The concept underlying rotating priority 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 can be accomplished by assigning a peripheral the lowest priority after being serviced. Thus, in the worst case, the device would have to wait until all other devices are serviced before being serviced again. OCW2 contains three commands which support rotating priority: two involve End-of-Interrupt [Rotate-at-EOI (Auto) and Rotate-at-EOI (Specific)] and one (Set-Priority), is independent of EOI. OCW2 contains one additional command which is not directly related to rotating priority but is sometimes used in conjunction with it: Specific EOI.

Set-Priority Command

The Set-Priority Command in OCW2 allows the programmer to select the bottom priority device independently of an EOI; that is, without affecting the ISR. Figure 11 shows the format for the Set-Priority Command. L2, L1, and LO code (in BCD) the IR input to be assigned the lowest priority. The priority of the remaining inputs are assigned accordingly. Example 2 illustrates the use of the Set-Priority Command.



OCWZ SET PRIORITY

DATA BUS FIELD

A0	D7	D6	D5	D4	D3	D2	D1	DO
0	1	1	0	0	0	L2	L1	L0

IR LEVEL TO BE PUT AT LOWEST PRIORITY

	0	1	2	3	4	5	6	7
L0	0	1	0	1	0	1	0	1
L1	0	0	1	1	0	0	1	1
L2	0	0	0	0	1	1	1	1

Figure 11. Set-Priority Command Format

EXAMPLE: STARTING WITH ANY PRIORITY STRUCTURE, ASSIGN IR2 PRIORITY LEVEL 4

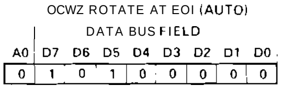
BOTTOM PRIORITY IR6 CORRESPONDS TO IR2 BEING LEVEL 4, THUS L2 = 1, L1 = 1, AND L0=0 IN THF SET PRIORITY COMMAND

PRIORITY	BEFORE INPUT	AFTER INPUT
HIGHEST	5	7
	6	0
	7	1
	0	2
	1	3
	2	4
	3	5
LOWEST	4	6

Example 2. Set-Priority Example

Rotate-at-EOI (Auto) Command

This command represents the "general purpose" implementation of Rotating Priority. When the Rotate-at-EOI (Auto) command is executed, the highest priority ISR bit is reset and priorities are rotated so that the request input of the ISR bit just reset is assigned the lowest priority. The format for the Rotate-at-EOI (Auto) command is shown in Figure 12. Since rotating priority implies that all peripherals are of equal importance, the service routines are usually sacrosanct; that is, the EI instruction is placed at the end of the routine (after the EOI) to ensure that the routine will not be interrupted. Example 3 shows the effect of executing a Rotate-at-EOI (Auto) command.

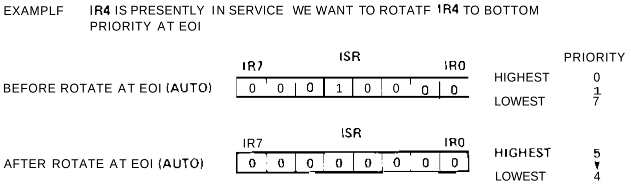


OCWZ ROTATE AT EOI (AUTO)

DATA BUS FIELD

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	1	0	0	0	0	0

Figure 12. Rotate-at-EOI (Auto) Command Format



EXAMPLF IR4 IS PRESENTLY IN SERVICE WE WANT TO ROTATF IR4 TO BOTTOM PRIORITY AT EOI

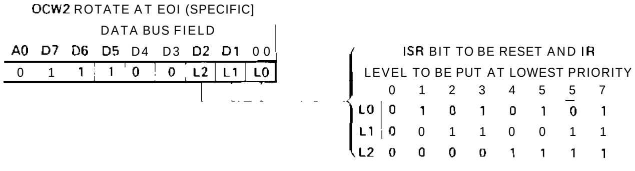
	IR7			ISR				IR0	PRIORITY
BEFORE ROTATE AT EOI (AUTO)	0	0	0	1	0	0	0	0	HIGHEST 0 LOWEST 7
AFTER ROTATE AT EOI (AUTO)	0	0	0	0	0	0	0	0	HIGHEST 5 LOWEST 4

Example 3. Rotate-at-EOI (Auto)

When using the commands that rotate priorities, it is possible that the 8259 will not be able to determine the last level acknowledged (especially if nesting is allowed). If Rotate-at-EOI (Auto) is the only command used to reset ISR bits, then there is no problem. When a number of different commands are used a problem could occur. To prevent the 8259 from becoming confused. two commands that reset specific ISR bits are provided: the Rotate-at-EOI (Specific) and the Specific EOI commands.

Rotate-at-EOI (Specific) Command

This command ensures that the correct ISR bit is reset at the end of a service routine because the bit to be reset is specified in the command itself. Additionally, the priorities are rotated so that the specified level is at the bottom. The format for the Rotate-at-EOI (Specific) command is shown in Figure 13. Example 4 illustrates this command.



OCW2 ROTATE AT EOI (SPECIFIC)

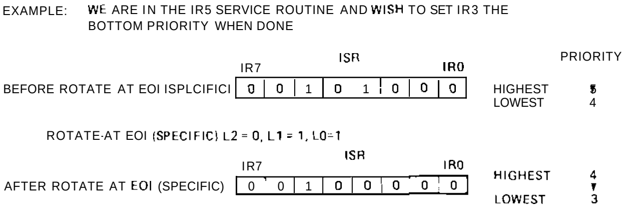
DATA BUS FIELD

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	1	0	0	L2	L1	L0

ISR BIT TO BE RESET AND IR LEVEL TO BE PUT AT LOWEST PRIORITY

	0	1	2	3	4	5	6	7
L0	0	1	0	1	0	1	0	1
L1	0	0	1	1	0	0	1	1
L2	0	0	0	0	1	1	1	1

Figure 13. Rotate-at-EOI (Specific) Command Format



EXAMPLE: WE ARE IN THE IR5 SERVICE ROUTINE AND WISH TO SET IR3 THE BOTTOM PRIORITY WHEN DONE

	IR7			ISR				IR0	PRIORITY
BEFORE ROTATE AT EOI ISPLCIFICI	0	0	1	0	1	0	0	0	HIGHEST 5 LOWEST 4

ROTATE-AT EOI (SPECIFIC) L2 = 0, L1 = 1, L0=1

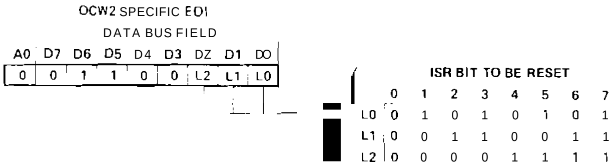
	IR7			ISR				IR0	
AFTER ROTATE AT EOI (SPECIFIC)	0	0	1	0	0	0	0	0	HIGHEST 4 LOWEST 3

Example 4. Rotate-at-EOI (Specific)

If the rotation of priorities is not desired, the Specific-EOI command is used.

Specific-EOI Command

The Specific-EOI command is identical to the Rotate-at-EOI (Specific) command except that priorities are not rotated after the ISR bit is reset. The Specific-EOI command format is shown in Figure 14.



OCW2 SPECIFIC EOI

DATA BUS FIELD

A0	D7	D6	D5	D4	D3	DZ	D1	DO
0	0	1	1	0	0	L2	L1	L0

ISR BIT TO BE RESET

	0	1	2	3	4	5	6	7
L0	0	1	0	1	0	1	0	1
L1	0	0	1	1	0	0	1	1
L2	0	0	0	0	1	1	1	1

Figure 14. Specific EOI Command Format

In summarizing the various commands which reset ISR bits, some words of caution are appropriate. If only the Fully Nested mode is used, the Non-Specific EOI can be used without problems. For any other mode, it is good practice to use the End-of-Interrupt commands which specify the ISR bit to be reset. No additional code is required and the reassurance of an unconfused 8259 during system debug is worth the effort. The OCW2 command words are summarized in Figure 15.

OCW2 COMMAND SUMMARY

COMMAND	A0	D7	D6	D5	D4	D3	D2	D1	DO	OPERATION
		DATA BUS FIELD								
NON-SPECIFIC EOI	0	0	0	1	0	0	0	0	0	RESET HIGHEST ISR BIT
SPECIFIC EOI	0	0	1	1	0	0	L2	L1	LO	RESET ISR SPECIFIED BY L2–LO
ROTATE-AT-EOI (AUTO)	0	1	0	1	0	0	0	0	0	RESET HIGHEST ISR BIT AND ASSIGN LOWEST PRIORITY
ROTATE-AT-EOI (SPECIFIC)	0	1	1	1	0	0	L2	L1	LO	RESET ISR SPECIFIED BY L2–LO AND ASSIGN LOWEST PRIORITY
SET-PRIORITY	0	1	1	0	0	0	L2	L1	LO	SET L2–LO LOWEST PRIORITY

Figure 15. OCW2 Command Summary

Interrupt Masks (OCW1)

OCWl controls the Interrupt Mask Register (IMR). Through OCWI, individual bits in the IMR may be set or reset by the software at any time. As stated earlier, the IMR acts only on the output of the Interrupt Request Register (IRR). Even with an IR input masked, it is still possible to set the IRR bit. However, no interrupt can be generated from the request since the IMR blocks the Priority Resolver from seeing the set IRR bit. If the IMR bit is reset while the IRR bit is set, the Priority Resolver can then see the IRR bit and an interrupt could be generated. After initialization, any command with A_0=1 is interpreted as an OCWI, see Figure 16.

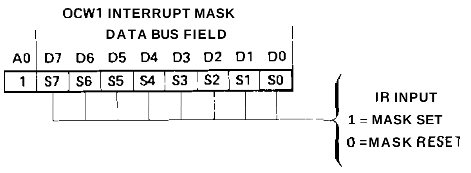


Figure 16. Interrupt Mask Command Format

Special Mask Mode (OCW3j

The last Operation Command Word is OCW3. This word controls two additional modes plus the reading of the various registers. The first mode is the Special Mask Mode (SMM).

Let us say that you are in a service routine that contains a section of code where you want all interrupts enabled: that is, you want to allow your lower priority devices to generate interrupts. You could accomplish this by using an EOI command to reset the ISR bit corresponding to the routine we are in. But resetting the ISR bit is irreversible and the lower priority devices remain enabled until another interrupt on your level occurred. The effect of the ISR bit can be temporarily suspended by first masking the input that is in-service and then setting the Special Mask Mode. Once SMM is set, it remains in affect until it is reset. The format to set and reset SMM is shown in Figure 17. The only requirements for SMM are that the level corresponding to the routine setting SMM must be masked through OCW1 and that interrupts are enabled. Example 5 shows how to enable interrupts over a particular section of code.

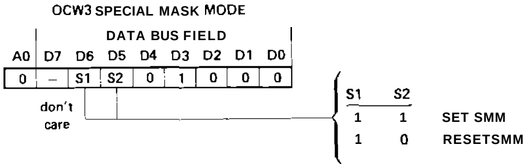


Figure 17. Special Mask Mode Command Formats

```
;EXAMPLE: IR4 IS IN-SERVICE AND WE WISH TO ENABLE LOWER
          PRIORITY INPUTS OVER A PARTICULAR SECTION OF COLE.

;IR4 SERVICE ROUTINE WHICH CONTAINS SPECIAL MASK MODE

IR4:      EI                    ;ENABLE INTERRUPTS

          ;1ST PART OF SERVICE ROUTINE -
          ;LOWER PRIORITY INPUTS DISABLED.

          DI                    ;DISABLE INTERRUPTS FOR COMMANDS
          MVI       A,10H       ;MASK IR4
          OUT       PT59B       ;8259 PUHT A0=1
          MVI       A,48H       ;SET SMM
          OUT       PT59A       ;8259 PORT A0=0
          EI                    ;ENABLE INTERRUPTS
          ;
          ;2ND PART OF SERVICE ROUTINE-
          ;LOWER PRIORITY INTERRUPTS ENABLED
          ;
          DI                    ;DISABLE INTERRUPTS FOR COMMANDS
          MVI       A,68H       ;RESET SMM
          OUT       PT59A       ;8259 PORT A0=0
          MVI       A,00H       ;REMOVE MASK ON IR4
          OUT       PT59B       ;8259 PORT A0=1
          EI                    ;ENABLE INTERRUPTS

          ;3RD PART OF SERVICE ROUTINE -
          ;LOWER PRIORITY INPUTS DISABLED
          ;MUST CONTAIN APPROPRIATE EOI.
          ;
          RET                   ;RETURN
```

Example 5. Special Mask Mode

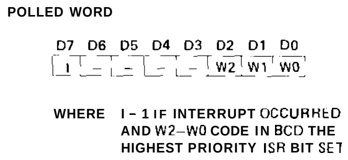
Note that SMM applies to all masked levels when set. If IR1 interrupts the IR4 routine in the above example while SMM is set, and then masks itself, 1R2 and 1R3 are enabled.

Polled Mode (OCW3)

The 8259 also supports the polled interrupt method of I/O cervicing mentioned earlier. Rather than having the processor poll the peripherals in order to find the actual interrupting device, the processor polls the 8259. This allows the use of all of the aforementioned priority modes. Additionally, both the polled and vectored interrupt methods can be used within the same program.

Bacically. the polling is implemented by allowing the programmer to initiate a software controlled interrupt acknowledge through the "P" bit in OCW3. This interrupt acknowledge behaves exactly as the first "normal" hardware acknowledge; that is, the ISR bit of the highest priority input is set. The 8259 then enables a special word onto the data bus. This word shows whether an interrupt has occurred and what the highest ISR bit is.

To initiate a poll, interrupts must first be disabled; either by executing a DI instruction or from having an interrupt occur. Then an OCW3 with P=1 is sent to the 8259 using an OUTput command (or a $\overline{WR}$ pulse). The next $\overline{RD}$ pulse (possibly fron an INput command) is treated as an interrupt acknowledge, and the following word is placed on the data bus:


POLLED WORD
D7 D6 D5 D4 D3 D2 D1 D0
I
–
–
–
–
W2
W1
W0
WHERE I – 1 IF INTERRUPT OCCURRED AND W2–W0 CODE IN BCD THE HIGHEST PRIORITY ISR BIT SET

Service to the requesting device is achieved by the software decoding this word and branching to the appropriate service routine. Every time a poll is to be performed, the OCW3 must be written before the $\overline{RD}$ pulse. If a poll is performed without an interrupt having occured or with no request inputs in-service. the returned word is 1=0 and LO, L1, and L2=1. The format for OCW3 Poll Command is shown in Figure 18.

To illustrate the Polled mode, consider a system where the 8259 and the 8080 are on different cards, and the system bus does not contain a line for the $\overline{INTA}$ interrupt acknowledge, although interrupt request lines are provided. In this instance, the Polled mode is the only way to take advantage of the 8259's prioritizing features. The INT pin of the 8259 is connected to the Interrupt Request line of the system bus while the 8259 $\overline{INTA}$ pin is simply held high. The 8080 card must contain logic to jam either a CALL or a RST instruction on the card's data bus in response lo an interrupt on the system bus (either an 8359 on the processor card or an 8228 would accomplish this). The RST or the CALL vectors the 8080 to a polling routine. The polling routine simply writes an OCW3 with P=1 to the off-board 8259 port followed by an input at the same port. The 8259 then releases the above word onto the system data bus. The polling routine then decodes the returned word and vectors the 8080 to the appropriate service routine.

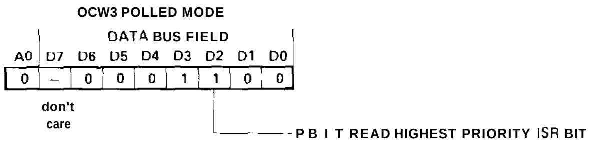

OCW3 POLLED MODE
DATA BUS FIELD
A0 D7 D6 D5 D4 D3 D2 D1 D0
0 – 0 0 0 1 1 0 0
don't care
P B I T READ HIGHEST PRIORITY ISR BIT

Figure 18. Polled Mode Command Format

This method can be extended to multiple off-board 8259s. Each 8259 is polled and the returned word indicates whether the selected 8259 is the one which generated the interrupt. Do not forget that even though the CALL features of the off-board 8259 are not being used, each 8259 must receive an initialization sequence. In this case, the starting address specified in the ICWs could be a "fake".

Reading the 8259 Status (OCW3)

The contents of the IRR, the ISR, and the IMR can be read to update the user information on the system. The registers are read by issuing the appropriate OCW3 and then reading with an INput instruction or $\overline{RD}$ pulse. The OCW3 words for reading the IRR and the ISR are shown in Figure 19.

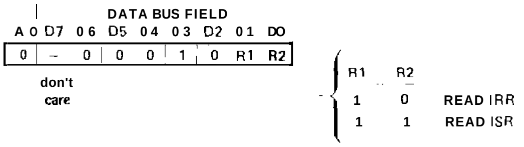


Figure 19. Read Status Command Formats

There is no need to write an OCW3 before every status read as long as the status read corresponds with the previous one; i.e., the 8259 "remembers" whether the ISR or the IRR has been previously selected by the OCW3.

For reading the IMR, an OUTput instruction (or $\overline{\text{WR}}$ pulse) is not necessary to precede the INput instruction (or $\overline{\text{RD}}$ pulse). The 8259 data lines contain the IMR whenever $\overline{\text{RD}}$ is active and A_0=1. Thus an INput instruction to the 8259 A_0=1 port reads the IMR at any time.

A summary of OCW3 command words is shown in Figure 20.

OCW3 COMMAND SUMMARY

COMMAND	A0	D7	D6	D5	D4	D3	D2	D1	D0	OPERATION
		DATA BUS FIELD								
POLL MODE	0	–	0	0	0	1	1	0	0	POLL ON NEXT $\overline{\text{RD}}$
READ ISR	0	–	0	0	0	1	0	1	1	READ ISR ON NEXT $\overline{\text{RD}}$
READ IRR	0	–	0	0	0	1	0	1	0	READ IRR ON NEXT $\overline{\text{RD}}$
SET SMM	0	–	1	1	0	1	0	0	0	SET SMM
RESET SMM	0	–	1	0	0	1	0	0	0	RESET SMM

Figure 20. OCW3 Command Summary

CASCADING THE 8259

As mentioned earlier, more than one 8259 can be used to expand the priority interrupt scheme to up to 64 levels without additional hardware. In such cases, one 8259 acts as a master, and the others serve as slaves. Figure 21 shows a system containing a master and two slaves providing a total of 22 levels of interrupt.

Hardware-wise, the master is designated by a "high" on the $\overline{\text{SP}}$ pin, while the $\overline{\text{SP}}$ pins of the slaves are grounded. Additionally, the INT output pins of the slaves are connected to the IR input pins of the master. Any IR master pin can be used to support a slave. The CASO—2 pins for all 8259s are paralleled. These pins act as outputs when the 8259 is a master and act as inputs for the slaves. The CAS0–2 pins serve as a private 8259 bus to control which slave has control of the system data bus when the destination address is issued to the 8080.

The sequence of events for a valid interrupt request on a slave is covered here. The slave IR input makes an inactive-to-active transition. Assuming this request is higher priority than other requests and in-service levels on the slave, the slave's INT pin is pulled high, signaling the master of the request. Assuming that this request to the master is higher priority than other master requests (possibly from other slaves) and master in-service levels, the master's INT pin is pulled high, interrupting the 8080. When this interrupt is acknowledged by the 8080, the master places the CALL instruction on the data bus. The master knows that the original request was on a slave (from ICW3 that will be covered shortly) and then puts the interrupted slave's ID on the CAS lines. This causes the slave to

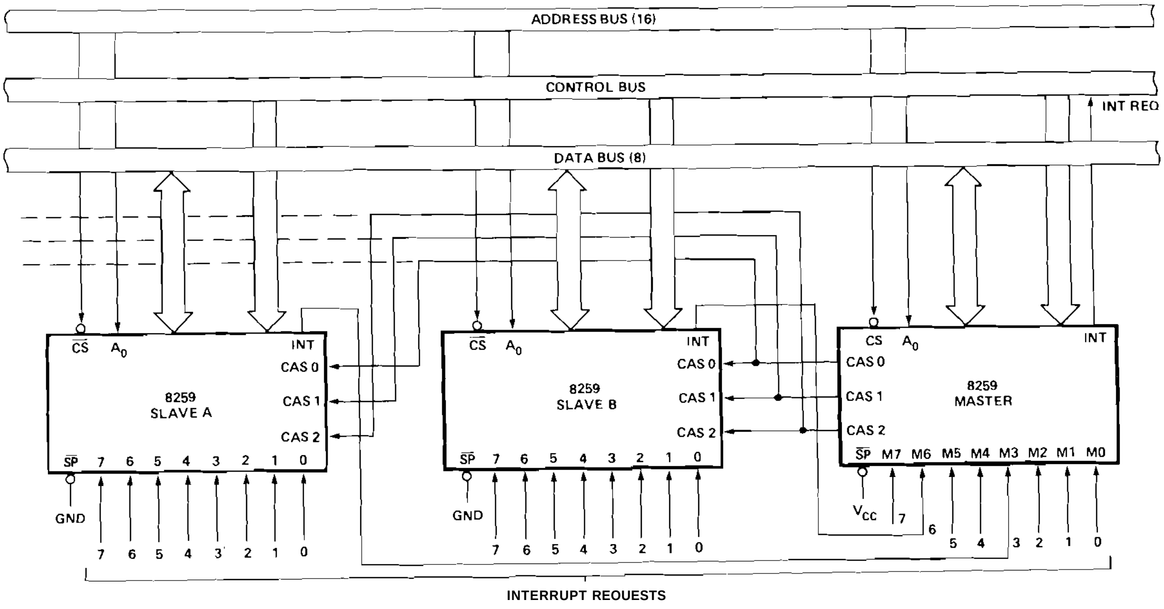


Figure 21. Cascaded System Diagram

place on the bus its preprogrammed address for the requesting input during the second and third $\overline{\text{INTA}}$s. The appropriate ISR bits for both the master and slave are set. This completes the interrupt request.

Several things should become evident from the above sequence. First, because there are two ISR bits that are set by an acknowledged slave interrupt, two EOI commands must be issued; one for the master and one for the slave. And second, each 8259 must have a separate initialization sequence. This gives each IR input a unique address plus defines how the master and slaves are interconnected. This interconnection is specified in ICW3. The master ICW3 tells the master which of its IR inputs are connected to slaves. The slave ICW3 tells the slave which IR master input it is connected to. This IR input is the slave's ID. The format for ICW3 is shown in Figure 7. Also note that each slave could receive commands to operate in different modes: i.e., one slave could be in Rotating Priority while the other is in Fully Nested mode.

An initialization sequence is illustrated in Example 6. The master's jump table starts at OOH, slave A's at 20H, and slave B's at 40H; all with 4-byte intervals. The master ICW3 shows that there are slaves on IR inputs 3 and 6. Slave A ICW3 shows its ID as 3, indicating that it is the slave connected to the master IR3. Slave B's ID is 6 and it is connected to the master IR6. The priority levels are now arranged as shown.

		ICW									
		DATA BUS FIELD									
		A0	D7	D6	D5	D4	D3	D2	D1	DO	HEX
MASTER	ICW1	0	0	0	0	1	0	1	0	0	14
	ICW2	1	0	0	0	0	0	0	0	0	00
	ICW3	1	0	1	0	0	1	0	0	0	48
SLAVEA	ICW1	0	0	0	1	1	0	1	0	0	32
	ICW2	1	0	0	0	0	0	0	0	0	00
	ICW3	1	0	0	0	0	0	0	1	1	03
SLAVE B	ICW1	0	0	1	0	1	0	1	0	0	54
	ICW2	1	0	0	0	0	0	0	0	0	00
	ICW3	1	0	0	0	0	0	1	1	0	06

PRIORITY STRUCTURE

LOWEST HIGHEST

M7 SB7–SB0 M5 M4 SA7–SA0 M2 M1 M0

Example 6. Cascaded Initialization

Some special housekeeping software in the slave interrupt service routines is required in order to preserve a truly Fully Nested structure. Why? Notice that if level SA5 (IR5 on slave A) is in-service (both the Slave A ISR5 bit and the master ISR3 bit are set) and level SA2 is asserted, then the priority structure of the slave will assert an interrupt to the master. But the master's ISR bit for that level is already set from the SA5 request. This will prohibit the request from being acknowledged until the master receives an EOI, thus losing the true Fully Nested structure since a request on SA2 should interrupt a SA5 service routine.

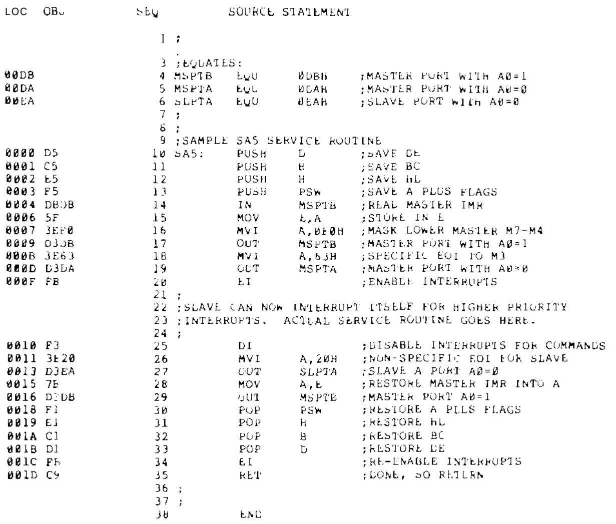
To solve this dilemma, the first task upon entering a service routine of a device connected to a slave is to mask off the lower priority master IR inputs. (in this case, M7, M6, M5, and M4). Then issue an EOI to the master for the input the slave is connected to (Specific EOI M3). This enables the master to accept higher priority interrupts from the slave. The masking process allows any interrupt request from a higher priority (higher than SA5) to be acknowledged and any lower priority request (M7 thru SA6) to be ignored. If the lower priority master inputs were not masked, the master would acknowledge a request on, for instance M7. since the M3 ISR bit is reset by the master EOI.

Software must also maintain the information that level SA5 is the lowest priority slave in-service. This is because the masks on the lower priority master inputs must be removed upon completing a service routine, but only by the lowest in-senrice slave level. If SA2 is the only in-service level then it resets the masks. However, in the main example. the SA2 routine returns to the SA5 routine. In this case, SA2 should not reset the masks, but allow SA5 to reset them just before returning. This can be accomplished by reading and saving the master IMR upon entering a slave input service routine arid then restoring it upon leaving. Figure 22 is an example of how the SA5 service routine should look. This form should be followed for all service routines of devices connected to slave IR inputs.

APPLICATION EXAMPLES

POWER FAIL/AUTO-START WITH BATTERY BACKED-UP RAM

The first application illustrates the 8259 used in the Fully Nested mode in 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.



```
LOC  OBJ          SEQ         SOURCE STATEMENT

                    1 ;

                    3 ;EQUATES:
00DB                4 MSPTB    EQU     0DBH     ;MASTER PORT WITH A0=1
00DA                5 MSPTA    EQU     0DAH     ;MASTER PORT WITH A0=0
00EA                6 SLPTA    EQU     0EAH     ;SLAVE PORT WITH A0=0
                    7 ;
                    8 ;
                    9 ;SAMPLE SA5 SERVICE ROUTINE
0000 D5            10 SA5:     PUSH    D        ;SAVE DE
0001 C5            11          PUSH    B        ;SAVE BC
0002 E5            12          PUSH    H        ;SAVE HL
0003 F5            13          PUSH    PSW      ;SAVE A PLUS FLAGS
0004 DBDB          14          IN      MSPTB    ;REAL MASTER IMR
0006 5F            15          MOV     E,A      ;STORE IN E
0007 3EF0          16          MVI     A,0F0H   ;MASK LOWER MASTER M7-M4
0009 D3DB          17          OUT     MSPTB    ;MASTER PORT WITH A0=1
000B 3E63          18          MVI     A,63H    ;SPECIFIC EOI TO M3
000D D3DA          19          OUT     MSPTA    ;MASTER PORT WITH A0=0
000F FB            20          EI               ;ENABLE INTERRUPTS
                   21 ;
                   22 ;SLAVE CAN NOW INTERRUPT ITSELF FOR HIGHER PRIORITY
                   23 ;INTERRUPTS.  ACTUAL SERVICE ROUTINE GOES HERE.
                   24 ;
0010 F3            25          DI               ;DISABLE INTERRUPTS FOR COMMANDS
0011 3E20          26          MVI     A,20H    ;NON-SPECIFIC EOI FOR SLAVE
0013 D3EA          27          OUT     SLPTA    ;SLAVE A PORT A0=0
0015 7B            28          MOV     A,E      ;RESTORE MASTER IMR INTO A
0016 D3DB          29          OUT     MSPTB    ;MASTER PORT A0=1
0018 F1            30          POP     PSW      ;RESTORE A PLUS FLAGS
0019 E1            31          POP     H        ;RESTORE HL
001A C1            32          POP     B        ;RESTORE BC
001B D1            33          POP     D        ;RESTORE DE
001C FB            34          EI               ;RE-ENABLE INTERRUPTS
001D C9            35          RET              ;DONE, SO RETURN
                   36 ;
                   37 ;
                   38          END
```

Figure 22. Sample Slave Service Routine

The vehicle used in this application is the Intel® SBC 80/20 Single Board Computer. The SBC 80/20 contains an 8259 on-board 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 bc 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 8259 (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 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 1031206 VAC (RMS).

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

1. An AC power failure occurs and the ACLO goes high (ACLO is pulled up by the battery supply). This indicates that DC power will be reliable for at most 7.5 ms. The power fail circuitry generates a Power Fail Interrupt (PFI) signal. This signal sets the PFI latch, which is connected to the IRO input of the 8259, 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 (cool start).
2. The processor is interrupted by the 8259 when the PFI latch is set. This pushes the pre-power-down program counter onto the stack and calls the service routine for the IRO input. The IRO 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 the RAM.
4. DC power goes down.
5. AC power returns. The power-on reset circuitry on the 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. Ths PFS latch is cleared, the MPRO signal is removed, and the PFI latch driving JRO 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 23 illustrates this timing.

Figure 24 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 25. The design is very straightforward and uses CMOS logic to minimize the battery current requirements. The Cold Start switch is necessary to ensure that during a cold start, the PFS line is indicating "cold start" sense (PFS high). Thus, for a cold start, the Cold Start switch is depressed during power on. After that, no further action is needed. Notice that the PFI signal sets the on-board PFI latch. The output of this latch drives the 8259 IRO input. This latch is cleared during the restart routine by executing an OUTput D4 H instruction. The state of the PFS line may be read on the least significant data bus line (DBO) by executing an INput D4 H instruction. An 8255 Port (8255 #1, Port C, bit 0) is used to control the PFSR line.

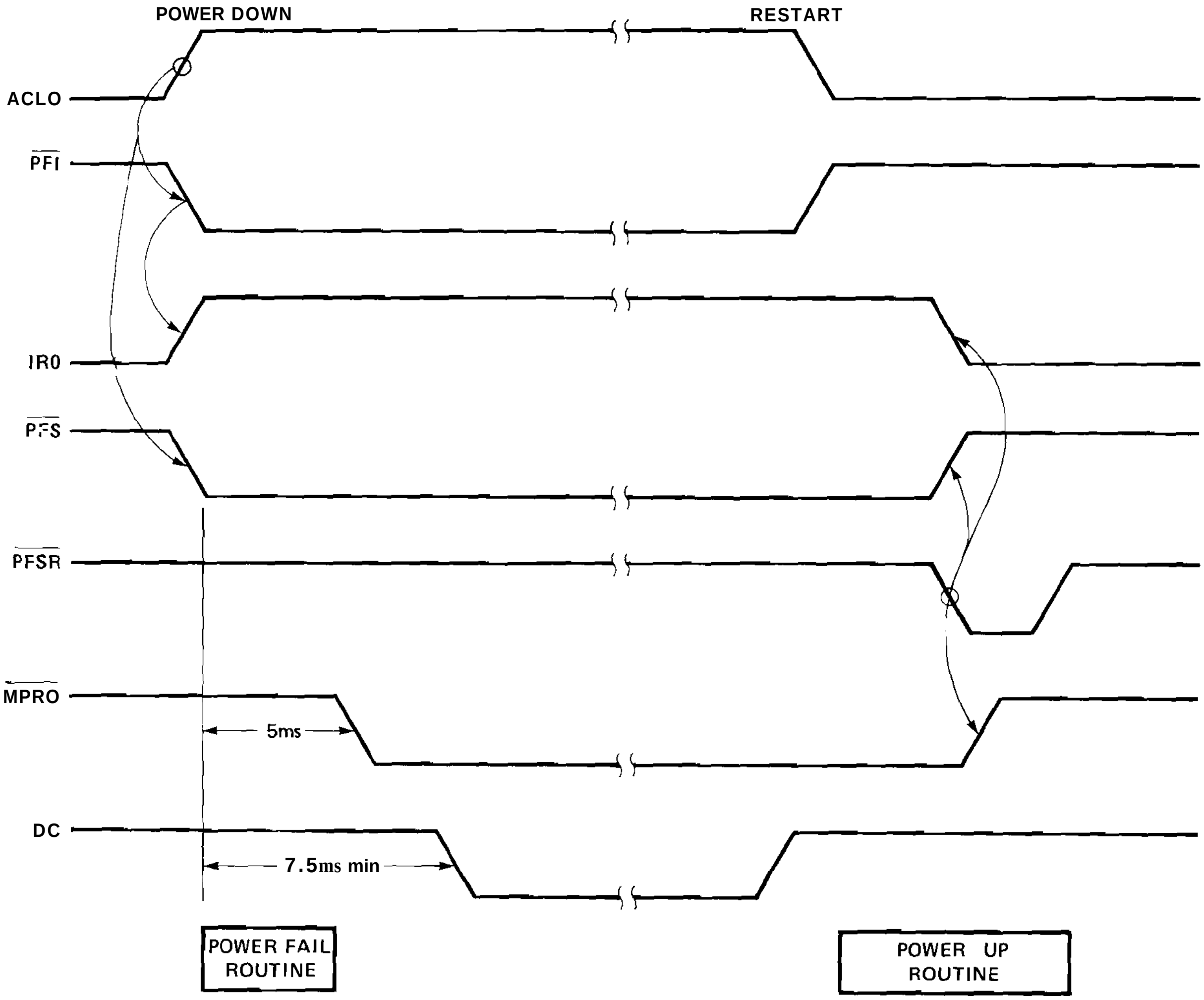


Figure 23. Power Down - Restart Timing

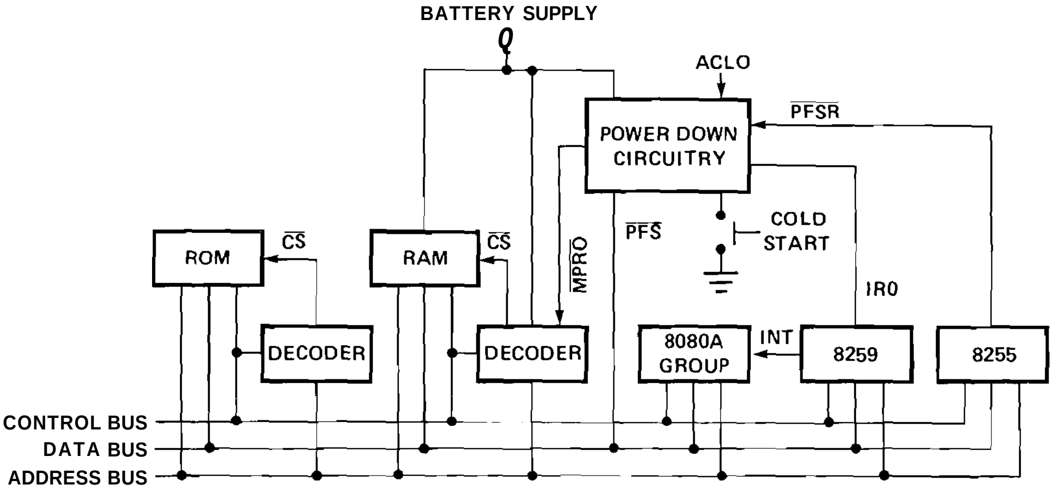


Figure *24.* SBC *80120* with Power Down

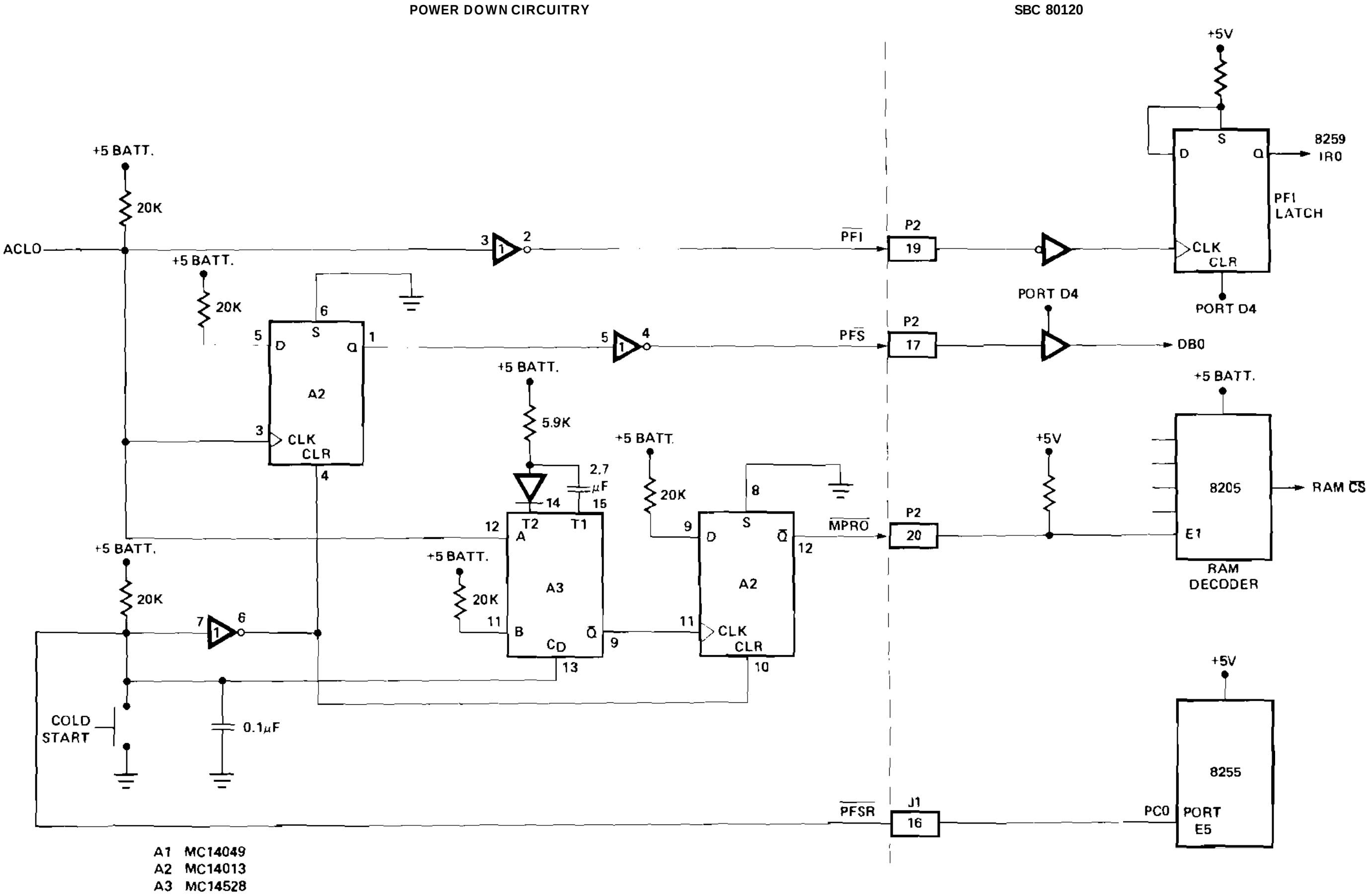
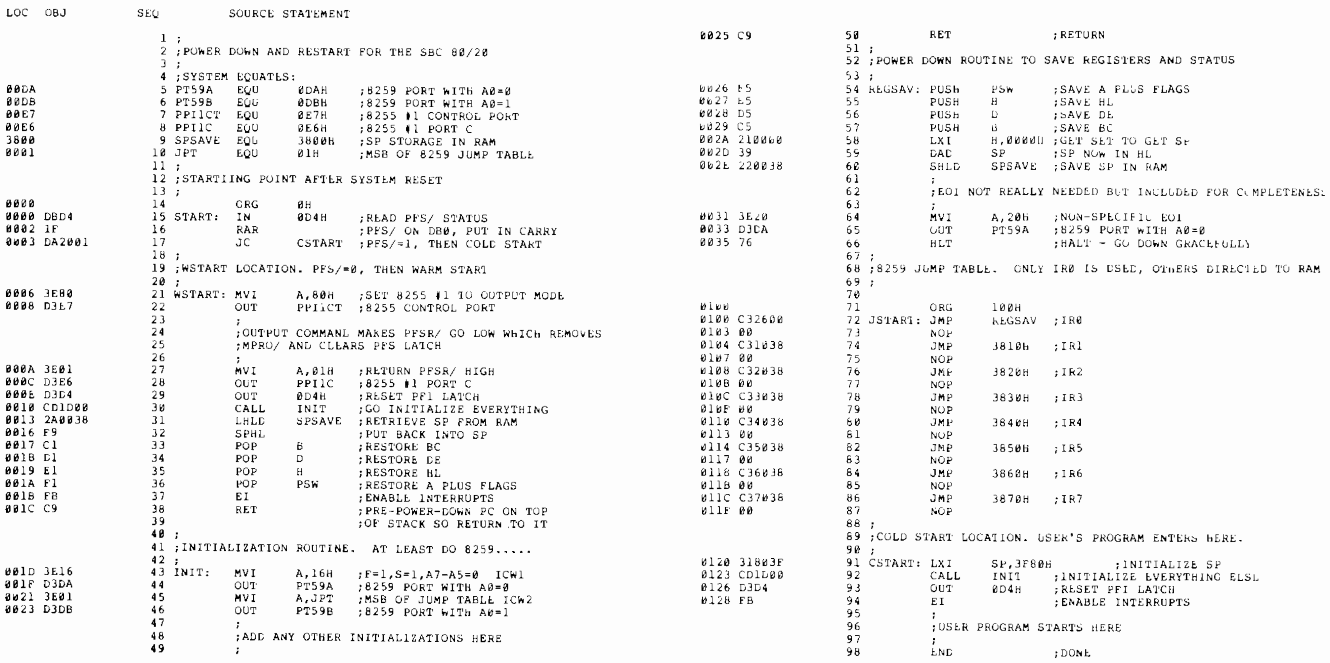


Figure *25.* Power Down – SBC 80/20 Interface

The Fully Nested mode for the 8259 is used to ensure that 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 8259 could be added as a slave and be programmed to operate in a rotating mode. The master would remain in 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 26. The flow for each label will be discussed.



```
LOC  OBJ        SEQ        SOURCE STATEMENT

                  1 ;
                  2 ;POWER DOWN AND RESTART FOR THE SBC 8Ø/2Ø
                  3 ;
                  4 ;SYSTEM EQUATES:
ØØDA              5 PT59A    EQU      ØDAH     ;8259 PORT WITH AØ=Ø
ØØDB              6 PT59B    EQU      ØDBH     ;8259 PORT WITH AØ=1
ØØE7              7 PPI1CT   EQU      ØE7H     ;8255 #1 CONTROL PORT
ØØE6              8 PPI1C    EQU      ØE6H     ;8255 #1 PORT C
38ØØ              9 SPSAVE   EQU      38ØØH    ;SP STORAGE IN RAM
ØØØ1             1Ø JPT      EQU      Ø1H      ;MSB OF 8259 JUMP TABLE
                 11 ;
                 12 ;STARTIING POINT AFTER SYSTEM RESET
                 13 ;
ØØØØ             14          ORG      ØH
ØØØØ DBD4        15 START:   IN       ØD4H     ;READ PFS/ STATUS
ØØØ2 1F          16          RAR               ;PFS/ ON DBØ, PUT IN CARRY
ØØØ3 DA2ØØ1      17          JC       CSTART   ;PFS/=1, THEN COLD START
                 18 ;
                 19 ;WSTART LOCATION. PFS/=Ø, THEN WARM START
                 2Ø ;
ØØØ6 3E8Ø        21 WSTART:  MVI      A,8ØH    ;SET 8255 #1 TO OUTPUT MODE
ØØØ8 D3E7        22          OUT      PPI1CT   ;8255 CONTROL PORT
                 23          ;
                 24          ;OUTPUT COMMAND MAKES PFSR/ GO LOW WHICH REMOVES
                 25          ;MPRO/ AND CLEARS PFS LATCH
                 26          ;
ØØØA 3EØ1        27          MVI      A,Ø1H    ;RETURN PFSR/ HIGH
ØØØC D3E6        28          OUT      PPI1C    ;8255 #1 PORT C
ØØØE D3D4        29          OUT      ØD4H     ;RESET PFI LATCH
ØØ1Ø CD1DØØ      3Ø          CALL     INIT     ;GO INITIALIZE EVERYTHING
ØØ13 2AØØ38      31          LHLD     SPSAVE   ;RETRIEVE SP FROM RAM
ØØ16 F9          32          SPHL              ;PUT BACK INTO SP
ØØ17 C1          33          POP      B        ;RESTORE BC
ØØ18 D1          34          POP      D        ;RESTORE DE
ØØ19 E1          35          POP      H        ;RESTORE HL
ØØ1A F1          36          POP      PSW      ;RESTORE A PLUS FLAGS
ØØ1B FB          37          EI                ;ENABLE INTERRUPTS
ØØ1C C9          38          RET               ;PRE-POWER-DOWN PC ON TOP
                 39                            ;OF STACK SO RETURN TO IT
                 4Ø ;
                 41 ;INITIALIZATION ROUTINE.  AT LEAST DO 8259.....
                 42 ;
ØØ1D 3E16        43 INIT:    MVI      A,16H    ;F=1,S=1,A7-A5=Ø  ICW1
ØØ1F D3DA        44          OUT      PT59A    ;8259 PORT WITH AØ=Ø
ØØ21 3EØ1        45          MVI      A,JPT    ;MSB OF JUMP TABLE ICW2
ØØ23 D3DB        46          OUT      PT59B    ;8259 PORT WITH AØ=1
                 47          ;
                 48          ;ADD ANY OTHER INITIALIZATIONS HERE
                 49          ;
ØØ25 C9          5Ø          RET               ;RETURN
                 51 ;
                 52 ;POWER DOWN ROUTINE TO SAVE REGISTERS AND STATUS
                 53 ;
ØØ26 F5          54 REGSAV:  PUSH     PSW      ;SAVE A PLUS FLAGS
ØØ27 E5          55          PUSH     H        ;SAVE HL
ØØ28 D5          56          PUSH     D        ;SAVE DE
ØØ29 C5          57          PUSH     B        ;SAVE BC
ØØ2A 21ØØØØ      58          LXI      H,ØØØØH  ;GET SET TO GET SP
ØØ2D 39          59          DAD      SP       ;SP NOW IN HL
ØØ2E 22ØØ38      6Ø          SHLD     SPSAVE   ;SAVE SP IN RAM
                 61          ;
                 62          ;EOI NOT REALLY NEEDED BUT INCLUDED FOR COMPLETENESS
                 63          ;
ØØ31 3E2Ø        64          MVI      A,2ØH    ;NON-SPECIFIC EOI
ØØ33 D3DA        65          OUT      PT59A    ;8259 PORT WITH AØ=Ø
ØØ35 76          66          HLT               ;HALT - GO DOWN GRACEFULLY
                 67 ;
                 68 ;8259 JUMP TABLE.  ONLY IRØ IS USED, OTHERS DIRECTED TO RAM
                 69 ;
                 7Ø
Ø1ØØ             71          ORG      1ØØH
Ø1ØØ C326ØØ      72 JSTART:  JMP      REGSAV   ;IRØ
Ø1Ø3 ØØ          73          NOP
Ø1Ø4 C31Ø38      74          JMP      381ØH    ;IR1
Ø1Ø7 ØØ          75          NOP
Ø1Ø8 C32Ø38      76          JMP      382ØH    ;IR2
Ø1ØB ØØ          77          NOP
Ø1ØC C33Ø38      78          JMP      383ØH    ;IR3
Ø1ØF ØØ          79          NOP
Ø11Ø C34Ø38      8Ø          JMP      384ØH    ;IR4
Ø113 ØØ          81          NOP
Ø114 C35Ø38      82          JMP      385ØH    ;IR5
Ø117 ØØ          83          NOP
Ø118 C36Ø38      84          JMP      386ØH    ;IR6
Ø11B ØØ          85          NOP
Ø11C C37Ø38      86          JMP      387ØH    ;IR7
Ø11F ØØ          87          NOP
                 88 ;
                 89 ;COLD START LOCATION. USER'S PROGRAM ENTERS HERE.
                 9Ø ;
Ø12Ø 318Ø3F      91 CSTART:  LXI      SP,3F8ØH         ;INITIALIZE SP
Ø123 CD1DØØ      92          CALL     INIT     ;INITIALIZE EVERYTHING ELSE
Ø126 D3D4        93          OUT      ØD4H     ;RESET PFI LATCH
Ø128 FB          94          EI                ;ENABLE INTERRUPTS
                 95          ;
                 96          ;USER PROGRAM STARTS HERE
                 97          ;
                 98          END               ;DONE
```

Figure 26. Power-down Software

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 Pointer (SP) and device initialization was included just to remind the reader that these must occur. The first EI instruction must appear after the 8259 has received its initialization sequence. The 8259 (and other devices) are initialized in the INIT subroutine. Four-byte intervals are selected for the 8259 since a jump table is being used (F=1) and S=1 since there is only one 8259 in the system. After initialization, the user's program is executed.

When a power failure occurs, execution is vectored by the 8259 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 is not really necessary to include an EOI command in REGSAV since power will be removed from the 8259, 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 when 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 8259 IR0 input. Then the 8259 is reinitialized 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 pre-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 8259 (and the SBC 80/20) in implementing a power failure protected microcomputer system, the above application should also point out a way of preserving the processor status when using interrupts.

78 LEVEL INTERRUPT SYSTEM

The second application illustrates the use of both the Fully Nested and Polled modes in implementing an interrupt structure with greater than 64 levels. The 8259 supports up to 64 levels with direct vectoring to the service routine. Extending the structure to greater than 64 levels requires the use of polling. A 78 level 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 8259s are used. Nine 8259s are cascaded in the master-slave scheme giving 64 levels at tier 2. Two additional 8259s 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 27 shows the tiered structure used in this example. Notice that the tier 3 8259s are connected to the bottom level slave (SA7). This simplifies the housekeeping required in the service routines since the IMR of the master does not have to be changed as discussed in the cascading section. The master-slaves are interconnected as shown before, while the tier 3 8259s are connected as "masters"; that is, the SP pins are pulled high and the CAS pins are left unconnected. Since these 8259s are only going to be used in the polled mode, no $\overline{\text{INTA}}$ is required, therefore the $\overline{\text{INTA}}$ pins are pulled high.

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 8259, the service routine for that input will poll the tier 3 8259 and branch to the tier 3 input service routine based on the word returned during the poll. Figure 28 shows how the jump table is organized assuming a starting location of 1000H and contiguous tables for all the tier 2 8259s. 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 8259s, 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.

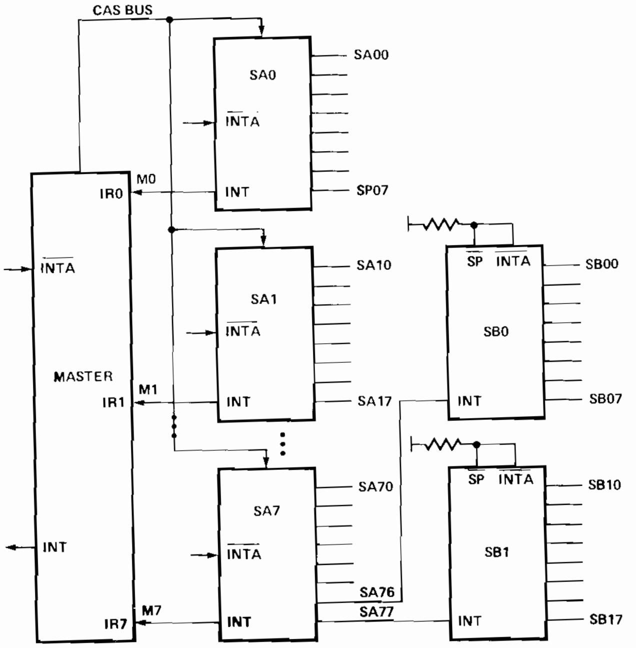


Figure 27. 78 Level Diagram

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 28. Jump Table Organization

Each 8259 must receive an initialization sequence regardless of the mode. Since the tier 1 and 2 8259s are in cascade, they require all three ICWs. The tier 3 8259s require only ICW1 and ICW2 since only polling will be used on them and they are connected as masters. The initialization sequence for each tier is shown in Figure 29. 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.

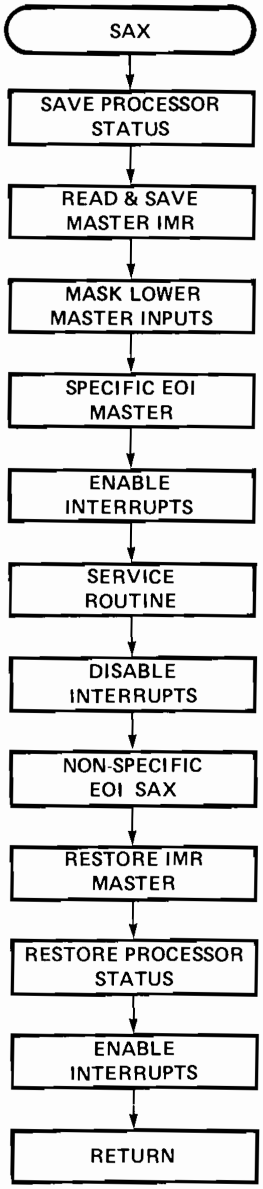
```
;  INITIALIZATION SEOUENCE
;
;  INITIALIZE MASTER
;
MINT:      MVI   A, 14H           ;  F-1, S-0, A7–A5-0  ICW1
           OUT   MPTA             ;  MASTER PORT A0=0
           MVI   A, 00H           ;  DUMMY ADR  ICW2
           OUT   MPTB             ;  MASTER PORT  A0=1
           MVI   A, FFH           ;  S7–S0=1  ICW3
           OUT   MPTB             ;  MASTER PORT  A0=1
;
;  INITIALIZE SA SLAVES – X DENOTES SLAVE ID  (SEE KEY)
SAXINT:    MVI   A, α             ;  SEE KEY  ICW1
           OUT   SAXPTA           ;  SAXPORT  A0=0
           MVI   A, 10H           ;  ADR MSB  ICW2
           OUT   SAXPTB           ;  SAXPORT  A0=1
           MVI   A, 0XH           ;    SA ID  ICW3
           OUT   SAXPTB           ;  SAXPORT  A0 =1
;
;  REPEAT ABOVE FOR EACH SA SLAVE
;
;  INITIALIZE SB SLAVES – INITIALIZE AS MASTERS (SINGLE)
;
SB0INT:    MVI   A, 16H           ;  F=1, S=1, A7–A5=0  ICW1
           OUT   SB0PTA           ;  SB0 PORT  A0=0
           MVI   A, 00H           ;  DUMMY ADR  ICW2
           OUT   SB0PTB           ;  SB0  PORT  A0=1
SB1INT:    MVI   A,16H            ;  F=1, S=1,  A7–A5=0  ICW1
           OUT   SB1PTA           ;  SB1  PORT  A0=0
           MVI   A,  00H          ;  DUMMY ADR  ICW2
           OUT   SB1PTB           ;  SB1  PORT  A0=1
```

SA INITIALIZATION KEY

SAX	α (ICW1)	JUMP TABLE START (H)
0	14	1000
1	34	1020
2	54	1040
3	74	1060
4	94	1080
5	B4	10A0
6	D4	10C0
7	F4	10E0

Figure 29. Initialization Sequence

As shown in the cascading section, some housekeeping is required by the service routines to preserve the Fully Nested structure. For the tier 2 inputs which do not have tier 3 8259s, the housekeeping is similar to that shown in Figure 22. Figure 30 shows this format generalized for any tier 2 service routine without a tier 3 8259. The housekeeping for the tier 2 service routines with tier 3 8259s is only slightly more complex. The additional complexity is due to the masking required on the slave itself since the tier 2–tier 3 situation is analogous to the master-slave situation described in the cascading section. In this case, if for example, SB05 is in-service, the M7 and SA76 ISR bits must be reset and SA77 masked off to enable a higher priority input (SB04–SB00) to generate an interrupt. Figure 31 shows the form for the SA76 service routine (labeled SB0 in the jump table) which polls tier 3 8259 SB0. Since a PCHL instruction is used to transfer execution to the appropriate SB0 service routine, by way of the jump table at 1200H, a separate return routine is used to end the interrupt for all SB0 inputs, thus all SB0 service routines should jump to SB0RET when complete. SB0RET restores the masks, executes an EOI, and restores the processor status.

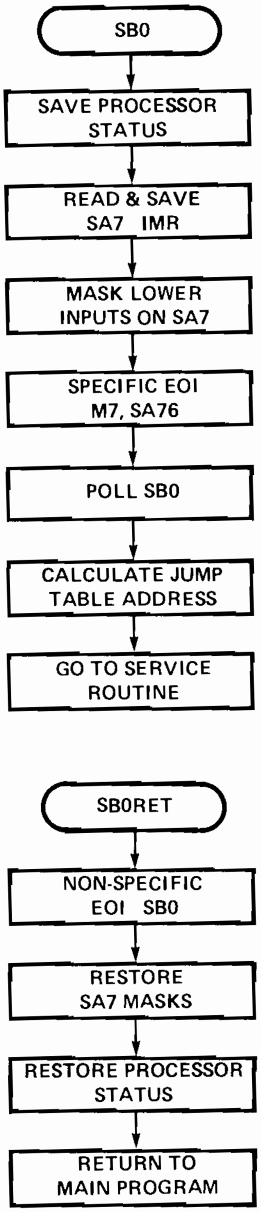


```
;
; GENERAL SAX SERVICE HOUSEKEEPING – USE KEY BELOW
; TO FIND VARIABLES
;
SAX :       PUSH  D               ; SAVE DE
            PUSH  B               ; SAVE BC
            PUSH  H               ; SAVE HL
            PUSH  PSW             ; SAVE A+FLAGS
            IN    MPTB            ; GET MASTER IMR
            MOV   E, A            ; SAVE IN E
            MVI   A, α            ; MASK LOWER MASTER, SEE KEY
            OUT   MPTB            ; MASTER PORT A0=1
            MVI   A, β            ; SPECIFIC EOI MASTER, SEE KEY
            OUT   MPTA            ; MASTER PORT A0=0
            EI                    ; ENABLE INTERRUPTS
;
; SERVICE ROUTINE GOES HERE
;
            DI                    ; DISABLE INTERRUPTS
            MVI   A, 20H          ; NON-SPECIFIC EOI SAX
            OUT   SAXPTA          ; SAX PORT  A0=0
            MOV   A,  E           ; RESTORE IMR MASTER
            OUT   MPTB            ; MASTER PORT  A0=1
            POP   PSW             ; RESTORE A+FLAGS
            POP   H               ; RESTORE HL
            POP   B               ; RESTORE BC
            POP   D               ; RESTORE DE
            EI                    ; RE-ENABLE INTERRUPTS
            RET                   ; DONE
```

KEY:

SAX	α(OCW1)	β (OCW2)
0	FE	60
1	FC	61
2	F8	62
3	F0	63
4	E0	64
5	C0	65
6	80	66
7	00	67

Figure 30. Generalizaed Slave Service Routine



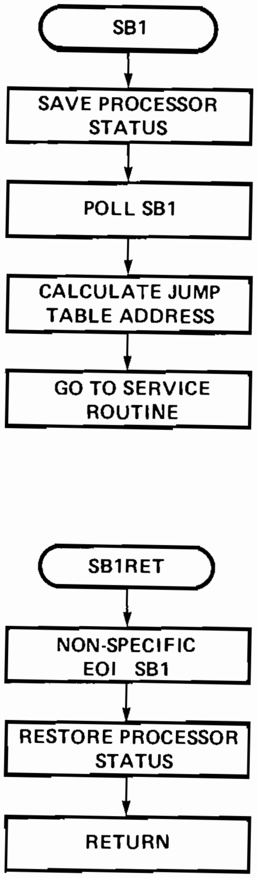
```
; SB0 ROUTINE – FINDS REQUESTING INPUT AND
; BRANCHS TO CORRESPONDING SERVICE ROUTINE
;
SB0:        PUSH  D               ; SAVE DE
            PUSH  B               ; SAVE BC
            PUSH  H               ; SAVE HL
            PUSH  PSW             ; SAVE A+FLAGS
            IN    SA7PTB          ; READ SA7 IMR
            MOV   D,A             ; SAVE IN D
            MVI   A, 80 H         ; MASK SA77
            OUT     SA7PTB        ; SA7 PORT A0=1
            MVI   A, 66H          ; SPECIFIC EOI SA76
            OUT   SA7PTA          ; SA7 PORT A0=0
            MVI   A, 67H          ; SPECIFIC EOI M7
            OUT   MPTA            ; MASTER PORT A0=0
            LXI   H, 1200H        ; JUMP TABLE START IN HL
            MVI   B, 00H          ; CLEAR B
            MVI   A, 0CH          ; POLL SB0
            OUT   SB0PTA          ; SB0 PORT A0=0
            IN    SB0PTA          ; GET POLL WORD
            ANI   07H             ; LIMIT TO 3 BITS
            ADD   A               ; GET TABLE OFFSET
            ADD   A               ;
            MOV   C,A             ; OFFSET IN C
            DAD   B               ; HL NOW HAS TABLE ADR
            PUSH  D               ; SAVE IMR
            EI                    ; ENABLE INTERRUPTS
            PCHL                  ; JUMP TO ROUTINE VIA TABLE
;
; SB0RET – DOES CLEAN UP AND EOI AFTER SB0 INTERRUPT
;
SB0RET:     DI                    ; DISABLE INTERRUPTS
            MVI   A, 20H          ; NON-SPECIFIC EOI SB0
            OUT   SB0PTA          ; SB0 PORT A0=0
            POP   D               ; RESTORE IMR
            MOV   A,D             ; SA7 IMR
            OUT   SB0PTB          ; SB0 PORT A0=1
            POP   PSW             ; RESTORE PSW
            POP   H               ; RESTORE HL
            POP   B               ; RESTORE BC
            POP   D               ; RESTORE DE
            EI                    ; RE-ENABLE INTERRUPTS
            RET                   ; BACK TO MAIN PROGRAM
```

Figure 31. SB0 Housekeeping

The SB1 service routine can be simplified somewhat since it is the bottom priority and no masks need to be changed. Figure 32 shows the SB1 routine. Like the SB0 routine, a PCHL instruction is used to transfer execution, therefore a separate return routine is provided for all SB1 inputs.

The above format can be followed for any number of inputs up to the limit of 512 although once tier 3 8259s are connected to tier 2 8259s above the master 7 input, it becomes necessary to include a section of code in the service routine to mask off and restore the master lower priority inputs.

This application has expanded the presentation of the cascading of 8259s and explained how to easily increase the number of interrupt levels by simply increasing the number of 8259s without adding additional hardware.



```
;
; SB1 ROUTINE – LOWEST PRIORITY SO NO MASKING
; REQUIRED
;
SB1  :    PUSH  D           ; SAVE DE
          PUSH  B           ;   AVE BC
          PUSH  H           ; SAVE HL
          PUSH  PSW         ; SAVE A+FLAGS
          LXI   H,1220H     ; JUMP TABLE START IN HL
          MVI   B, 00H      ; CLEAR B
          MVI   A, 0CH      ; POLL SB1
          OUT   SB1PTA      ; SB1 PORT A0=0
          IN    SB1PTA      ; GET POLL WORD
          ANI   07H         ; LIMIT TO 3 BITS
          ADD   A           ; CALCULATE OFFSET
          ADD   A
          MOV   C, A        ; MOVE OFFSET TO C
          DAD   B           ; HL HAS TABLE ADR
          EI                ; ENABLE INTERRUPTS
          PCHL              ; GO TO SERVICE ROUTINE
;
; SB1RET – DOES CLEAN UP AND EOI FOR ALL SB1
; INTERRUPTS
;
SB1RET:   DI                ; DISABLE INTERRUPTS
          MVI   A, 20H      ; NON-SPECIFIC EOI SB1
          OUT   SB1PTA      ; SB1 PORT A0=0
          POP   PSW         ; RESTORE A+FLAGS
          POP   H           ; RESTORE HL
          POP   B           ; RESTORE BC
          POP   D           ; RESTORE DE
          EI                ;    RE-ENABLE INTERRUPTS
          RET               ; RETURN
```

Figure 32. SB1 Housekeeping

CONCLUSION

This application note has explained the 8259 in detail and gives two applications illustrating the use of some of the numerous programmable features available. It should be evident from these discussions that the 8259 is an extremely flexible and easily programmed member of the Intel® MCS 80/85 Family.

Intel Corporation
3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 246-7501
TWX: 910-338-0026
TELEX: 34-6372

U.S. REGIONAL SALES OFFICES

WESTERN

CALIFORNIA
Intel Corp.*
1651 East 4th Street
Suite 150
Santa Ana 92701
Tel: (714) 835-9642
TWX: 910-595-1114

MID-WESTERN

ILLINOIS
Intel Corp.*
1000 Jorie Boulevard
Suite 224
Oakbrook 60521
Tel: (312) 325-9510
TWX: 910-651-5881

EASTERN

OHIO
Intel Corp.*
8312 North Main Street
Dayton 45415
Tel: (513) 890-5350
TELEX: 288-004

ATLANTIC

MASSACHUSETTS
Intel Corp.*
187 Billerica Road, Suite 14A
Chelmsford 01824
Tel: (617) 256-6567
TWX: 710-343-6333

*Field Application Location

OVERSEAS MARKETING OFFICES

ORIENT

JAPAN
Intel Japan Corporation*
Flower Hill-Shinmachi East Bdg.
1-23-9, Shinmachi, Setagaya-ku
Tokyo 154
Tel: (03) 426-9261
TELEX: 781-28426

EUROPE

BELGIUM
Intel International*
Rue du Moulin à Papier
51-Boite 1
B-1160 Brussels
Tel: (02) 660 30 10
TELEX: 24814

Printed in U.S.A/A253/0977/I0K BL