CHAPTER 7
INTERRUPTS

7.1 INTRODUCTION

The Z8® MCU allows 6 different interrupts from a variety of sources; up to four external inputs, the on-chip Counter/Timer(s), software, and serial I/O peripherals. These interrupts can be masked and their priorities set by using the Interrupt Mask and the Interrupt Priority Registers. All six interrupts can be globally disabled by resetting the master Interrupt Enable, bit 7 in the Interrupt Mask Register, with a Disable Interrupt (DI) instruction. Interrupts are globally enabled by setting bit 7 with an Enable Interrupt (EI) instruction.

There are three interrupt control registers: the Interrupt Request Register (IRQ), the Interrupt Mask register (IMR), and the Interrupt Priority Register (IPR). Figure 7-1 shows addresses and identifiers for the interrupt control registers. Figure 7-2 is a block diagram showing the Interrupt Mask and Interrupt Priority logic.

The Z8 MCU family supports both vectored and polled interrupt handling. Details on vectored and polled interrupts can be found later in this chapter.

Register	HEX	Identifier
Interrupt Mask	FBH	IMR
Interrupt Request	FAH	IRQ
Interrupt Priority	F9H	IPR

Figure 7-1. Interrupt Control Registers

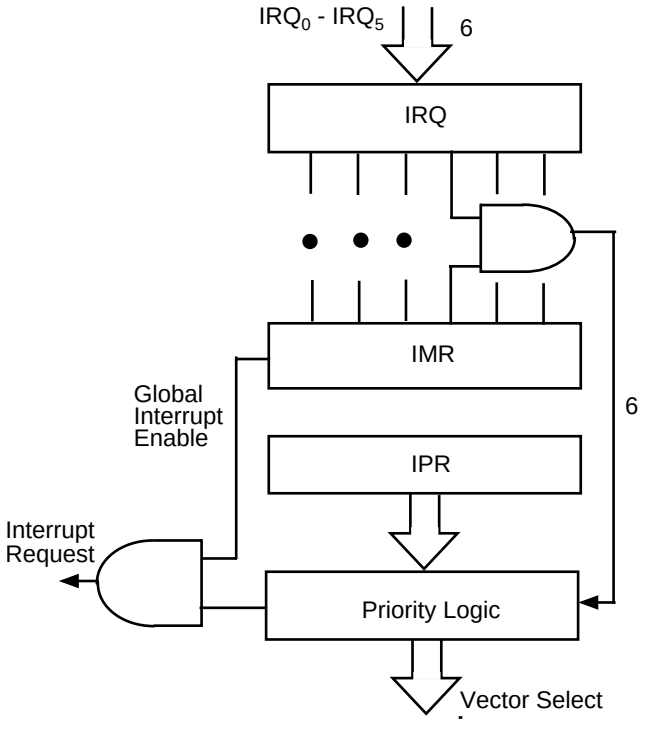

Figure 7-2. Interrupt Block Diagram

Note: See the selected Z8 MCU's product specification for the exact interrupt sources supported.

7.2 INTERRUPT SOURCES

Table 7-1 presents the interrupt types, sources, and vectors available in the Z8® family of processors.

Table 7-1. Interrupt Types, Sources, and Vectors *

Name	Sources	Vector Location	Comments
IRQ_0	DAV_0, IRQ_0, Comparator	0,1	External ($P3_2$), Edge Triggered; Internal
IRQ_1	DAV_1, IRQ_1	2,3	External ($P3_3$), Edge Triggered; Internal
IRQ_2	DAV_2, IRQ_2, TIN, Comparator	4,5	External ($P3_1$), Edge Triggered; Internal
	IRQ_3	6,7	External ($P3_0$) or ($P3_2$), Edge Triggered; Internal
	Serial In	6,7	Internal
	T_0	8,9	Internal
	Serial Out	8,9	Internal
IRQ_5	T_1	10,11	Internal

7.2.1 External Interrupt Sources

External sources involve interrupt request lines IRQ0-IRQ3. IRQ0, IRQ1, and IRQ2 can be generated by a transition on the corresponding Port 3 pin (P32, P33, and P31 correspond to IRQ0, IRQ1, and IRQ2, respectively).

Figure 7-3 is a block diagram for interrupt sources IRQ0, IRQ1, and IRQ2.

Note: The interrupt sources and trigger conditions are device dependent. See the device product specification to determine available sources (internal and external), triggering edge options, and exact programming details.

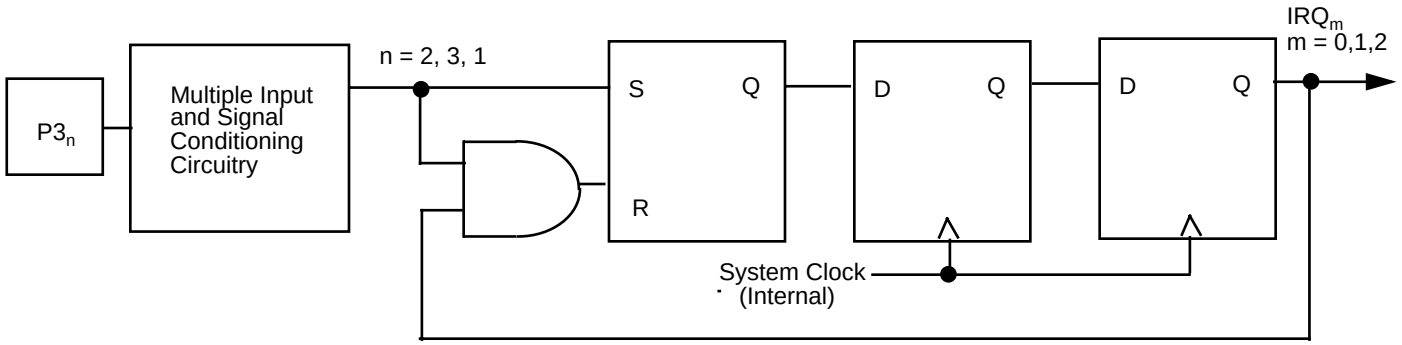

Figure 7-3. Interrupt Sources IRQ0-IRQ2 Block Diagram

When the Port 3 pin (P31, P32, or P33) transitions, the first flip-flop is set. The next two flip-flops synchronize the request to the internal clock and delay it by two internal clock periods. The output of the last flip-flop (IRQ0, IRQ1, or IRQ2) goes to the corresponding Interrupt Request Register.

IRQ3 can be generated from an external source only if Serial In is not enabled. Otherwise, its source is internal. The external request is generated by a Low edge signal on P30 as shown in Figure 7-4. Again, the external request is synchronized and delayed before reaching IRQ3. Some Z8® products replace P30 with P32 as the external source for IRQ3. In this case, IRQ3 interrupt generation follows the logic as illustrated in Figure 7-3.

Note: Although interrupts are edge triggered, minimum interrupt request Low and High times must be observed for proper operation. See the device product specification for exact timing requirements on external interrupt requests (T_WIL, T_WIH).

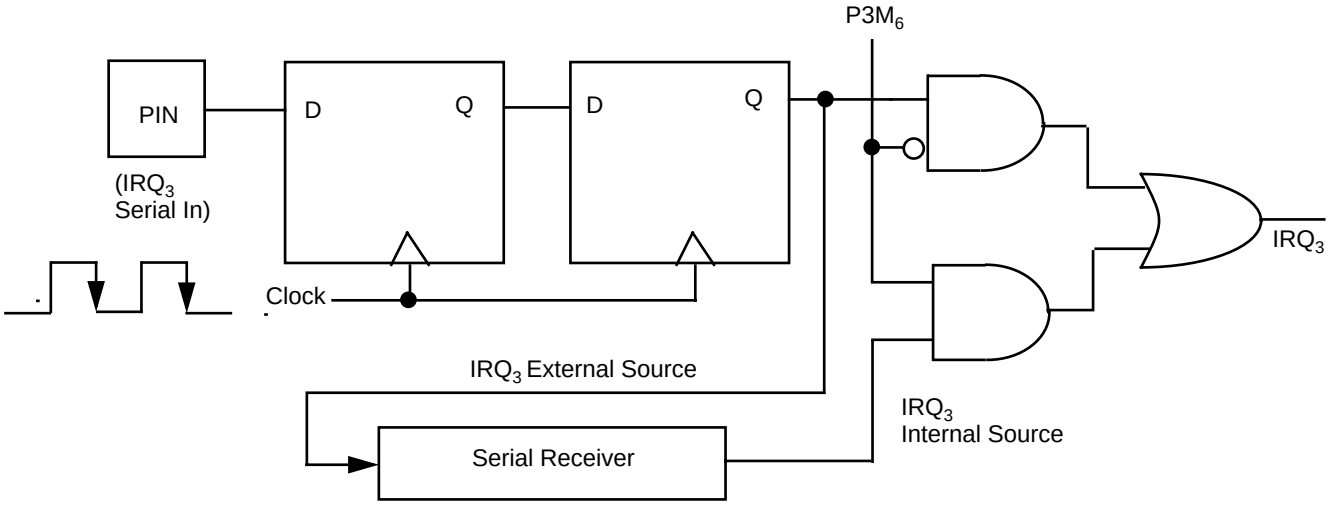

Figure 7-4. Interrupt Source IRQ3 Block Diagram

7.2.2 Internal Interrupt Sources

Internal sources involve interrupt requests IRQ0, IRQ2, IRQ3, IRQ4, and IRQ5. Internal sources are ORed with the external sources, so either an internal or external source can trigger the interrupt. Internal interrupt sources and trigger conditions are device dependent.

See the device product specification to determine available sources, triggering edge options, and exact programming details. For more details on the internal interrupt sources, refer to the chapters describing the Counter/Timer, I/O ports, and Serial I/O.

7.3 INTERRUPT REQUEST (IRQ) REGISTER LOGIC AND TIMING

Figure 7-5 shows the logic diagram for the Interrupt Request (IRQ) Register. The leading edge of the request will set the first flip-flop, that will remain set until interrupt requests are sampled.

Requests are sampled internally during the last clock cycle before an opcode fetch (Figure 7-6). External requests are sampled two internal clocks earlier, due to the synchronizing flip-flops shown in Figures 7-3 and 7-4.

At sample time the request is transferred to the second flip-flop in Figure 7-5, that drives the interrupt mask and priority logic. When an interrupt cycle occurs, this flip-flop will be reset only for the highest priority level that is enabled.

The user has direct access to the second flip-flop by reading and writing the IRQ Register. IRQ is read by specifying it as the source register of an instruction and written by specifying it as the destination register.

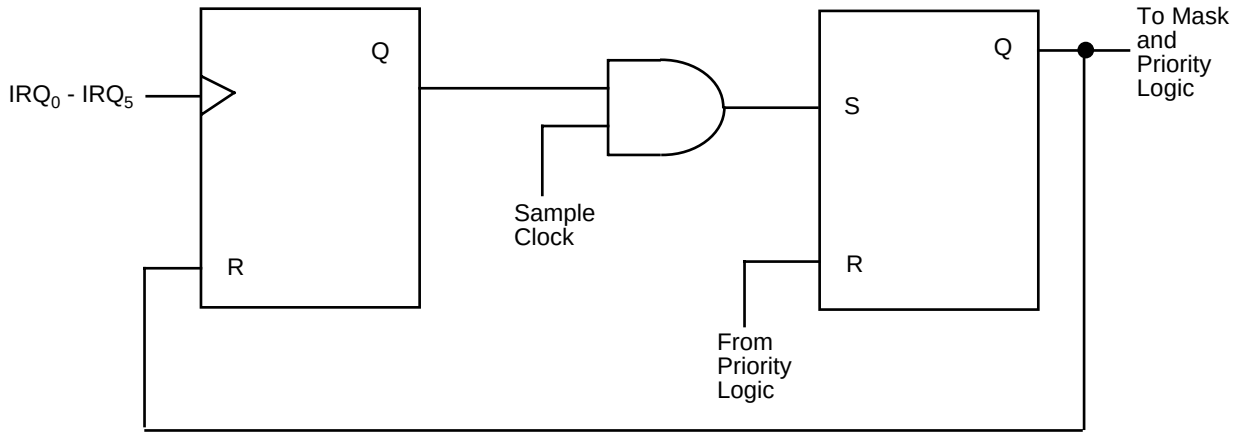

Figure 7-5. IRQ Register Logic

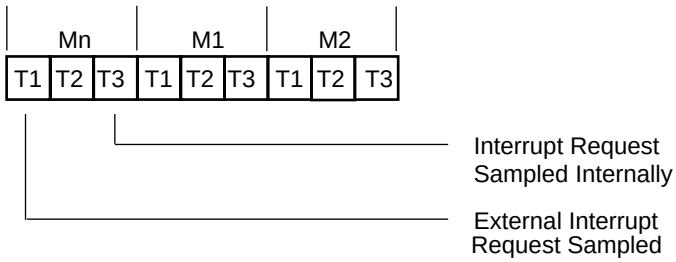

Figure 7-6. Interrupt Request Timing

7.4 INTERRUPT INITIALIZATION

After reset, all interrupts are disabled and must be initialized before vectored or polled interrupt processing can begin. The Interrupt Priority Register (IPR), Interrupt Mask Register (IMR), and Interrupt Request Register (IRQ) must be initialized, in that order, to start the interrupt process. However, IPR need not be initialized for polled proce7.4.1 Interrupt Priority Register (IPR) Initialization.

7.4.1 Interrupt Priority Register (IPR) Initialization

IPR (Figure 7-7) is a write-only register that sets priorities for the vectored interrupts in order to resolve simultaneous interrupt requests. (There are 48 sequence possibilities for interrupts.) The six interrupt levels IRQ0-IRQ5 are divided into three groups of two interrupt requests each. One group contains IRQ3 and IRQ5. The second group contains IRQ0 and IRQ2, while the third group contains IRQ1 and IRQ4.

Priorities can be set both within and between groups as shown in Tables 7-2 and 7-3. Bits 1, 2, and 5 define the priority of the individual members within the three groups. Bits 0, 3, and 4 are encoded to define six priority orders between the three groups. Bits 6 and 7 are reserved.

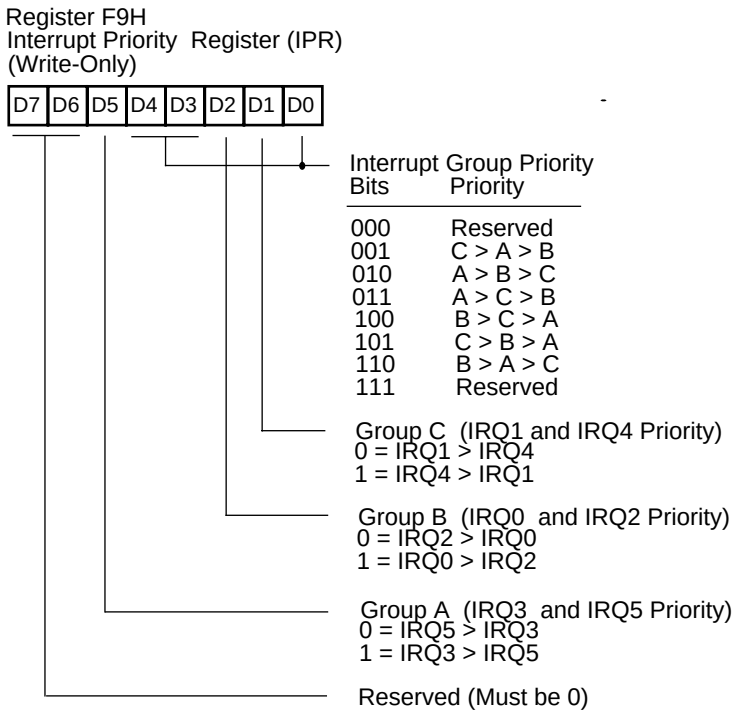

Figure 7-7. Interrupt Priority Register

Table 7-2. Interrupt Priority

Group	Bit	Value	Priority Highest	Lowest
C	Bit 1	0	IRQ1	IRQ4
		1	IRQ4	IRQ1
B	Bit 2	0	IRQ2	IRQ0
		1	IRQ0	IRQ2
A	Bit 5	0	IRQ5	IRQ3
		1	IRQ3	IRQ5

Table 7-3. Interrupt Group Priority

Bit Pattern			Group Priority		
Bit 4	Bit 3	Bit 0	High	Medium	Low
0	0	0	Not Used		
0	0	1	C	A	B
0	1	0	A	B	C
0	1	1	A	C	B
1	0	0	B	C	A
1	0	1	C	B	A
1	1	0	B	A	C
1	1	1	Not Used		

7.4 INTERRUPT INITIALIZATION (Continued)

7.4.2 Interrupt Mask Register (IMR) Initialization

IMR individually or globally enables or disables the six interrupt requests (Figure 7-8). When bit 0 to bit 5 are set to 1, the corresponding interrupt requests are enabled. Bit 7 is the master enable and must be set before any of the individual interrupt requests can be recognized. Resetting bit 7 globally disables all the interrupt requests. Bit 7 is set and reset by the EI and DI instructions. It is automatically reset during an interrupt service routine and set following the execution of an Interrupt Return (IRET) instruction.

Note: Bit 7 must be reset by the DI instruction before the contents of the Interrupt Mask Register or the Interrupt Priority Register are changed except:

- Immediately after a hardware reset.
- Immediately after executing an interrupt service routine and before IMR bit 7 has been set by any instruction.

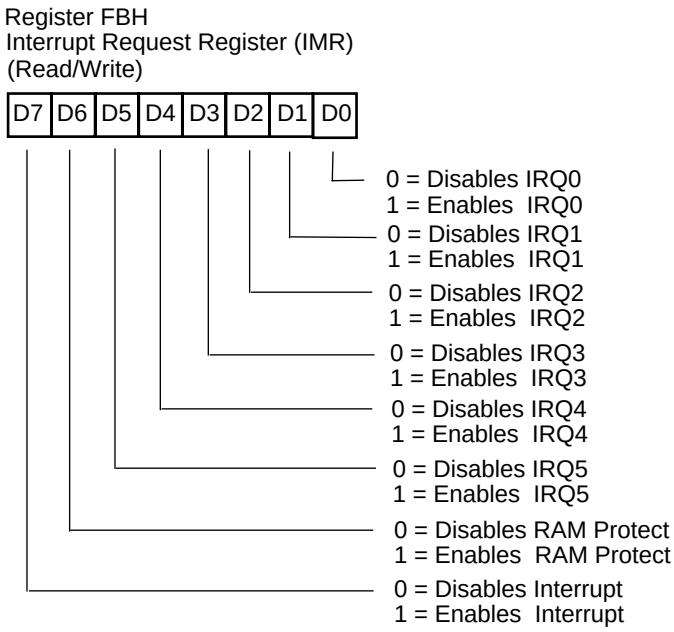

Figure 7-8. Interrupt Mask Register

Note: The RAM Protect option is selected at ROM mask submission time or at EPROM program time. If not selected or not an available option, this bit is reserved and must be 0.

7.4.3 Interrupt Request (IRQ) Register Initialization

IRQ (Figure 7-9) is a read/write register that stores the interrupt requests for both vectored and polled interrupts. When an interrupt is made on any of the six, the corresponding bit position in the register is set to 1. Bit 0 to bit 5 are assigned to interrupt requests IRQ0 to IRQ5, respectively.

Whenever Power-On Reset (POR) is executed, the IRQ resister is reset to 00H and disabled. Before the IRQ register will accept requests, it must be enabled by executing an ENABLE INTERRUPTS (EI) instruction.

Note: Setting the Global Interrupt Enable bit in the Interrupt Mask Register (IMR, bit 7) will not enable the IRQ. Execution of the EI instruction is required (Figure 7-10).

For polled processing, IRQ must still be initialized by an EI instruction.

To properly initialize the IRQ register, the following code is provided:

```
CLR     IMR     //make sure disabled vectored interrupts
EI              //enable IRQ register otherwise read
                only.
                //not needed if interrupts were
                previously enabled.
DI              //disable interrupt heading.
```

Note: IRQ is always cleared to 00Hex and is read only until the 1st EI instruction which enables the IRQ to be read/write.

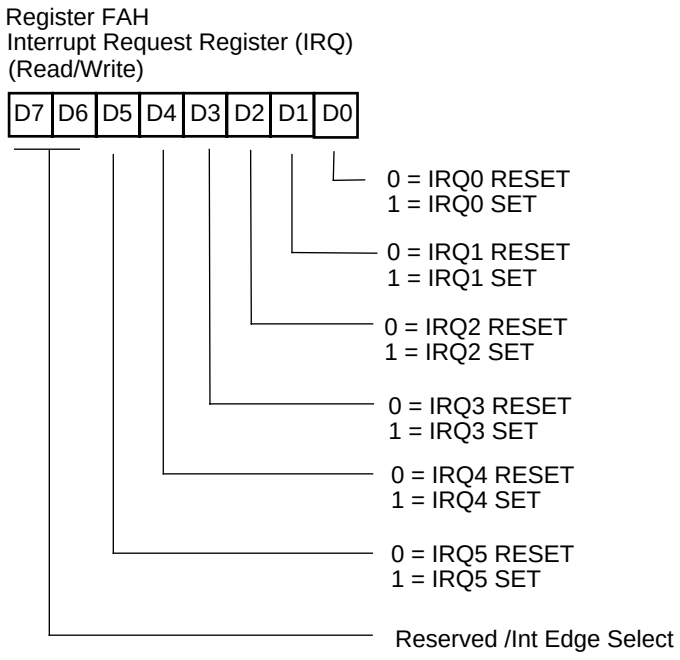

Figure 7-9. Interrupt Request Register

7.4 INTERRUPT INITIALIZATION (Continued)

IMR is cleared before the IRQ enabling sequence to insure no unexpected interrupts occur when EI is executed. This code sequence should be executed prior to programming the application required values for IPR and IMR.

Note: IRQ bits 6 and 7 are device dependent. When reserved, the bits are not used and will return a 0 when read. When used as the Interrupt Edge select bits, the configuration options are as show in Table 7-4.

Table 7-4. IRQ Register Configuration

IRQ		Interrupt Edge	
D7	**D6**	**P31**	**P32**
0	0	F	F
0	1	F	R
1	0	R	F
1	1	R/F	R/F

Notes:
F = Falling Edge
R = Rising Edge

The proper sequence for programming the interrupt edge select bits is (assumes IPR and IMR have been previously initialized):

```
DI                      ;Inhibit all interrupts
                        until input edges are
                        configured
OR      IRQ,#XX 000000B ;Configure interrupt
                        do not disturb
                        edges as needed -
                        IRQ 0-5.
EI                      ;Re-enable interrupts.
```

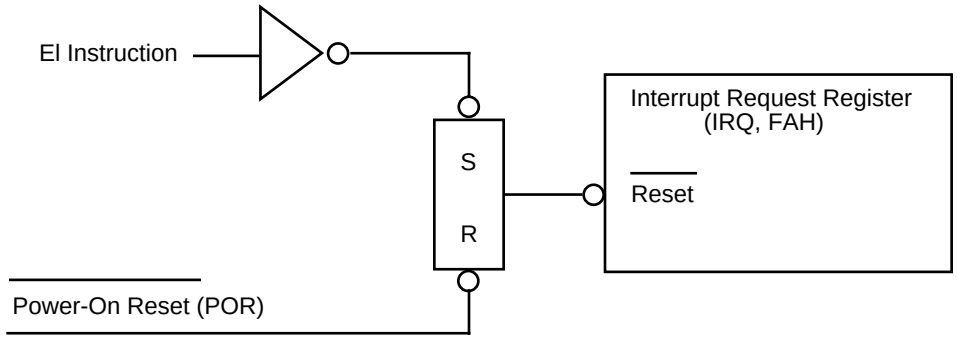

Figure 7-10. IRQ Reset Functional Logic Diagram

7.5 IRQ SOFTWARE INTERRUPT GENERATION

IRQ can be used to generate software interrupts by specifying IRQ as the destination of any instruction referencing the Z8® Standard Register File. These Software Interrupts (SWI) are controlled in the same manner as hardware generated requests (in other words, the IPR and the IMR control the priority and enabling of each SWI level).

To generate a SWI, the desired request bit in the IRQ is set as follows:

```
ORIRQ,          #NUMBER
```

where the immediate data, NUMBER, has a 1 in the bit position corresponding to the level of the SWI desired. For example, if an SWI is desired on IRQ5, NUMBER would have a 1 in bit 5:

```
OR              IRQ, #00100000B
```

With this instruction, if the interrupt system is globally enabled, IRQ5 is enabled, and there are no higher priority pending requests, control is transferred to the service routine pointed to by the IRQ5 vector.

7.6 VECTORED PROCESSING

Each Z8 interrupt level has its own vector. When an interrupt occurs, control passes to the service routine pointed to by the interrupt's vector location in program memory. The sequence of events for vectored interrupts is as follows:

- PUSH PC Low Byte on Stack
- PUSH PC High Byte on Stack
- PUSH FLAGS on Stack
- Fetch High Byte of Vector
- Fetch Low Byte of Vector
- Branch to Service Routine specified by Vector

Figures 7-11 and 7-12 show the vectored interrupt operation.

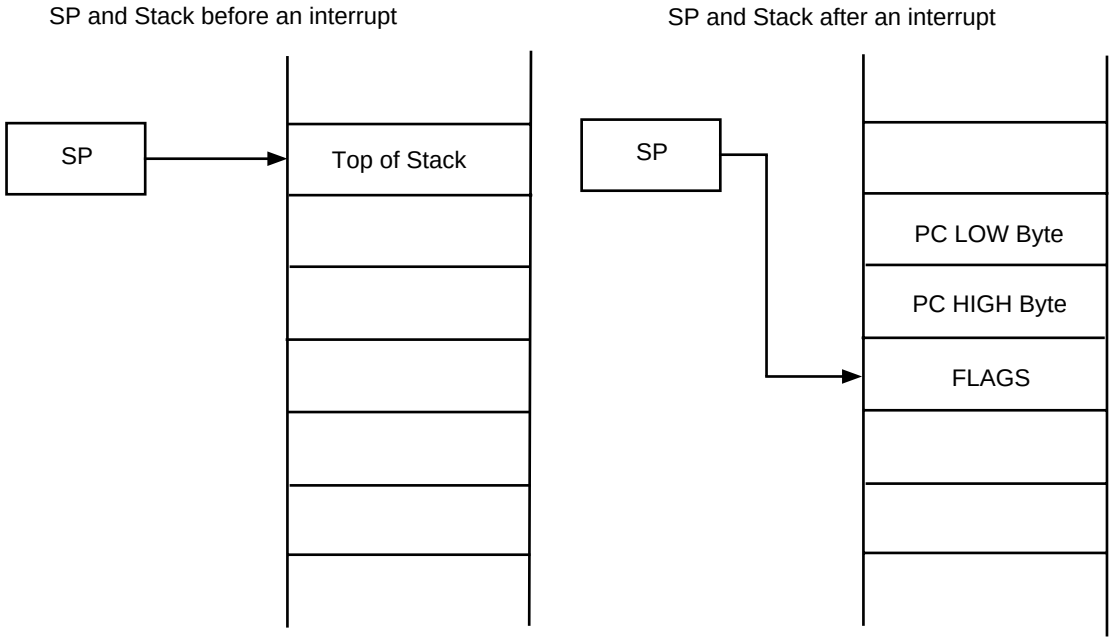

Figure 7-11. Effects of an Interrupt on the STACK

7.6 VECTORED PROCESSING (Continued)

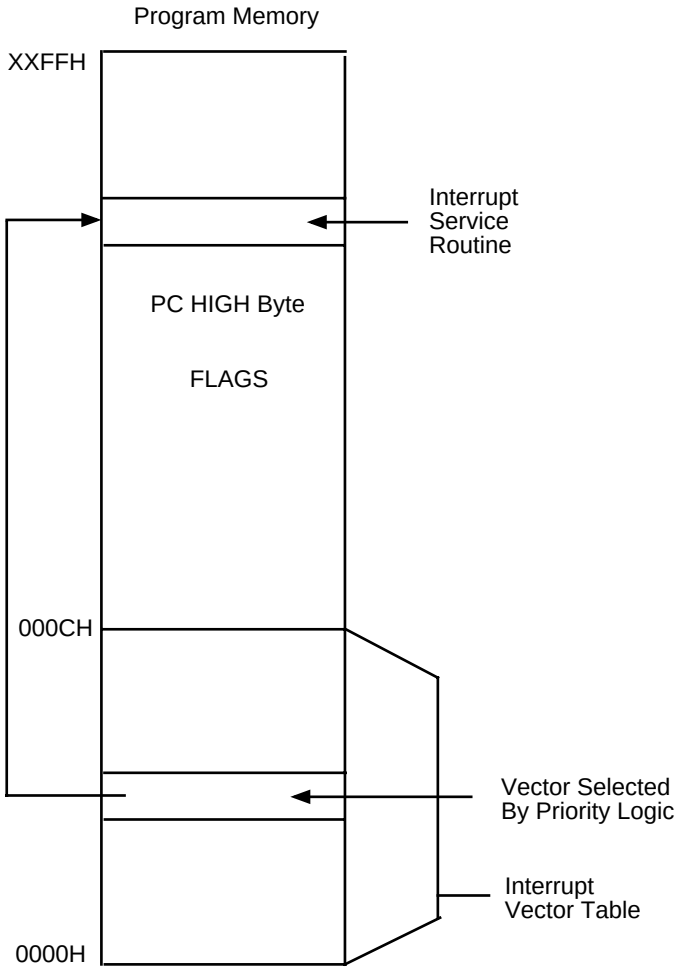

Figure 7-12. Interrupt Vectoring

7.6.1 Vectored Interrupt Cycle Timing

The interrupt acknowledge cycle time is 24 internal clock cycles and is shown in Figure 7-13. In addition, two internal clock cycles are required for the synchronizing flip-flops. The maximum interrupt recognition time is equal to the number of clock cycles required for the longest executing instruction present in the user program (assumes worst case condition of interrupt sampling, Figure 7-6, just prior to the interrupt occurrence). To calculate the worst case interrupt latency (maximum time required from interrupt generation to fetch of the first instruction of the interrupt service routine), sum these components:

Worst Case Interrupt Latency ≈ 24 INT CLK (interrupt acknowledge time) + # T_PC of longest instruction present in the user's application program + $2T_PC$ (internal synchronization time).

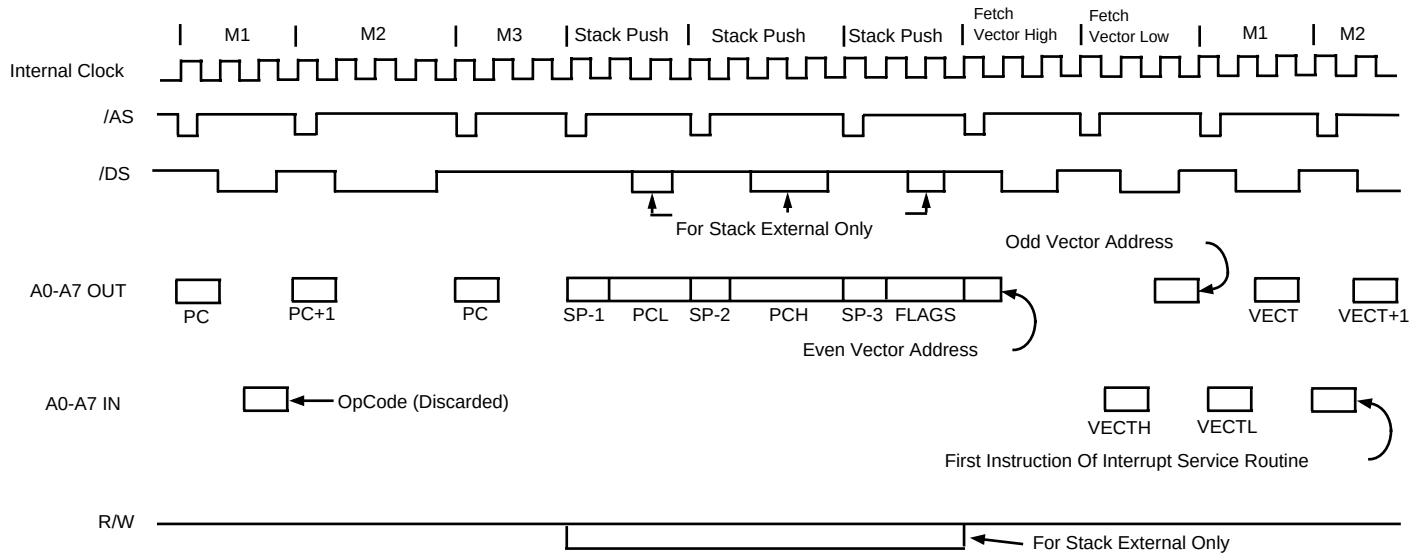

Figure 7-13. Z8 Interrupt Acknowledge Timing

7.6.2 Nesting of Vectored Interrupts

Nesting of vectored interrupts allows higher priority requests to interrupt a lower priority request. To initiate vectored interrupt nesting, do the following during the interrupt service routine:

- Push the old IMR on the stack.
- Load IMR with a new mask to disable lower priority interrupts.
- Execute EI instruction.
- Proceed with interrupt processing.
- After processing is complete, execute DI instruction.
- Restore the IMR to its original value by returning the previous mask from the stack.
- Execute IRET.

Depending on the application, some simplification of the above procedure may be possible.

7.7 POLLED PROCESSING

Polled interrupt processing is supported by masking off the IRQ to be polled. This is accomplished by clearing the corresponding bits in the IMR.

To enable any interrupt, first the interrupt mechanism must be engaged with an EI instruction. If only polled interrupts are to be serviced, execute:

```
EI    ;Enable interrupt mechanism
DI    ;Disable vectored interrupts.
```

To initiate polled processing, check the bits of interest in the IRQ using the Test Under Mask (TM) instruction. If the bit is set, call or branch to the service routine. The service routine services the request, resets its Request Bit in the IRQ, and branches or returns back to the main program. An example of a polling routine is as follows:

```
        TM    IRQ, #MASKA     ;Test for request
        JR    Z, NEXT         ;If no request go to
                              NEXT
        CALL SERVICE          ;If request is there, then
                              ;service it
NEXT:
        .
        .
        .
SERVICE:                      ;Process Request
        .
        .
        .
        AND IRQ, #MASKB       ;Clear Request Bit
        RET                   ;Return to next
```

In this example, if IRQ2 is being polled, MASKA will be 00000100B and MASKB will be 11111011B.

7.8 RESET CONDITIONS

Upon reset, all bits in IPR are undefined.

In IMR, bit 7 is 0 and bits 0-6 are undefined. The IRQ register is reset and held in that state until an enable interrupt (EI) instruction is executed.