5.3 THE INTERRUPT VECTOR TABLE

All types of interrupts, whether hardware or software generated, point to a single entry in the processor's **interrupt vector table.** This table is a collection of 4-byte addresses (two for CS and two for IP) that indicate where the processor should jump to execute the associated interrupt service routine, the code designed to handle the specific interrupt. Because 256 interrupts are available, the interrupt vector table is 1,024 bytes long. The 1KB block of memory reserved for the table is located in the address range 00000 to 003FFH. Some earlier processors automatically loaded their first instruction from address 0000 after a reset. The 80x86 fetches its first address from location FFFF0H. This indicates that we are able to begin program execution without first having to place values into the interrupt vector table. If we plan on using any interrupts, it will be necessary to initialize the required vectors within the table. This can be done easily with a few MOV instructions.

Figure 5.1 shows the organization of the interrupt vector table. Each 4-byte entry consists of a 2-byte IP register value followed by a 2-byte CS register value. This indicates that interrupt service routines are considered far routines. Notice that some of the vectors are predefined. Vector 0 (the same as type-0) has been chosen to handle division-by-zero errors. Vector 1 (type-1) helps to implement single-step operation. Vector 2 is used when NMI is activated. Vector 3 (breakpoint) is normally used when troubleshooting a new program. Vector 4 is associated with the INTO instruction. Vector 5 is used when the BOUND instruction reports an out-of-range index. Vectors 6 through 18 perform various housekeeping duties. Some of these vectors (10, 11, 14) are operational only in protected or virtual-8086 mode. Table 5.2 shows the associated vector assignments. Vectors 19 through 31 are reserved by Intel for use in their products. This does not mean that these interrupt vectors are unavailable to us, but that we should refrain from using them in an Intel machine unless we know how they have been assigned.

TABLE 5.2 Interrupt/exception vectors

Vector	*Description*
0	Divide error
1	Debugger call (single-step)
2	NMI
3	Breakpoint
4	INTO
5	BOUND range exceeded
6	Invalid opcode
7	Device not available
8	Double fault
9	Reserved
10	Invalid task state segment
11	Segment not present
12	Stack exception
13	General protection
14	Page fault
15	Reserved
16	Floating-point error
17	Alignment check
18	Machine check
19–31	Reserved
32–255	Maskable interrupts (nonreserved)

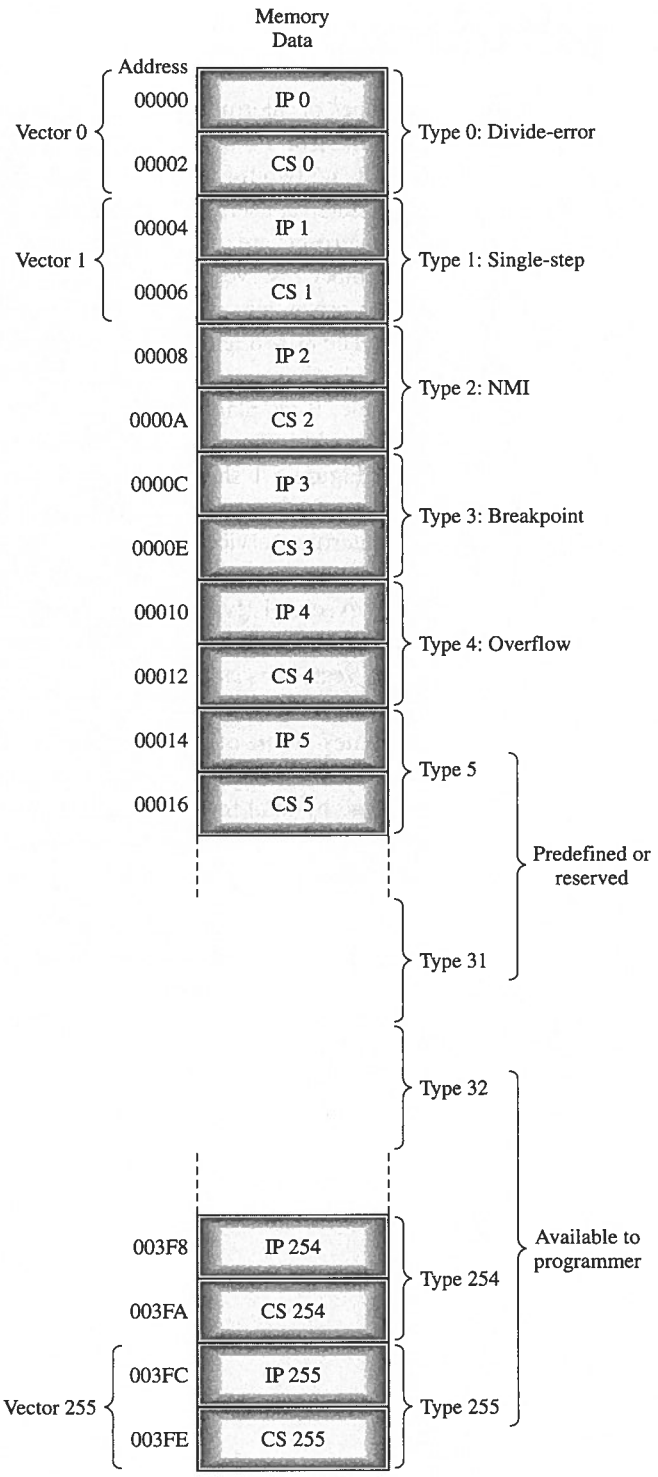

FIGURE 5.1 Interrupt vector table

Vectors 32 through 255 are unassigned and free for us to use. To initialize an interrupt vector, we must write the 4 bytes of the interrupt service routine address into the table locations reserved for the interrupt. A short example shows one way this can be done.

EXAMPLE 5.1 The interrupt service routine for a type-40 interrupt is located at address 28000H. How is the interrupt vector table set up to handle this interrupt?

Solution: An easy way to determine the address within the interrupt vector table that is used by an interrupt is to multiply the interrupt number by 4. Multiplying 40 by 4 and converting into hexadecimal gives 000A0H as the starting address of the vector for INT 40. The interrupt service routine address 28000H can be generated by many different combinations of CS and IP values. If CS is loaded with 2800H and IP with 0000, we get the correct address of 28000H. Thus, it is necessary to write these two address values into memory starting at 000A0H. The short section of code shown here is one way to do this:

```
PUSH    DS                          ;save current DS address
MOV     AX,0                        ;set new DS address at 0000
MOV     DS,AX
MOV     DI,00A0H                    ;offset for INT 40 vector
MOV     WORD PTR [DI],0             ;store IP address
MOV     WORD PTR [DI + 2],2800H     ;store CS address
POP     DS                          ;get old DS address back
```

The PUSH DS instruction is used to save the current value of the DS register on the stack. Because we need to access memory in the 00000 to 003FFH range, it is convenient to load DS with 0000 and use DI as the offset into the vector table. Notice that the first word written is 0000, which goes into locations 000A0H and 000A1H. Then the CS value 2800H is written into locations 000A2H and 000A3H. Figure 5.2 provides a snapshot of memory after these instructions execute. Now, when INT 40 executes, it will cause a jump to the interrupt service routine located at address 28000H. The POP DS instruction restores the old value of the DS register.

Figure 5.2 also shows the contents of memory for the vector associated with INT 41. What is the address of the ISR? As usual, the two words have been byte-swapped. The

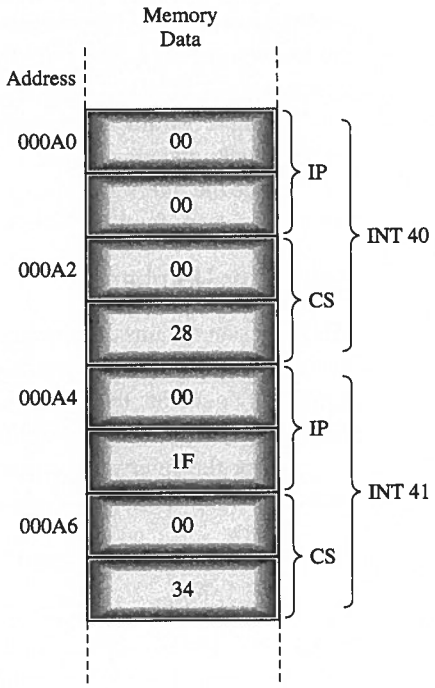

FIGURE 5.2 ISR address for INT 40H and INT 41H

word at address 000A4H is 1F00H. The word at address 000A6H is 3400H. The effective address created by the addition of these two words is 35F00H, which is the address of the ISR for INT 41.

The machine code for INT 40 is CD 28 (note that 28H equals 40 decimal). The machine code for INT 41 is CD 29. All INT instructions begin with CD as their first byte and have the interrupt number as the second byte. The only exception to this rule is INT 3, *breakpoint,* which has only the byte CC as its opcode. ■

In the examples presented later, we will see other ways in which the interrupt vector table can be initialized.

5.4 THE INTERRUPT PROCESSING SEQUENCE

In Chapter 4 we were introduced to the interrupt process in the coverage of the INT and INTO instructions. These software interrupts initiate a sequence of steps in which the flags and return address are saved prior to loading CS and IP with the ISR address. The same process is followed for hardware interrupt NMI, which automatically generates a type-2 interrupt. The sequence initiated by INTR (when interrupts are enabled) is slightly different, because the processor must first read the interrupt number from the data bus. When interrupts are enabled, INTR causes the processor to perform two **interrupt acknowledge cycles.** In the 8088, external devices recognize these cycles by examining the state of the $\overline{\text{INTA}}$ output, which goes low during each cycle. The first low-going pulse on $\overline{\text{INTA}}$ is used to indicate to other devices on the system bus that the processor is beginning an interrupt acknowledge cycle. In minimum mode this indicates that the processor will not acknowledge a hold request until the interrupt acknowledge cycle completes. In maximum mode the processor activates its $\overline{\text{LOCK}}$ output to prevent a system bus takeover during the interrupt acknowledge cycle.

The second low pulse on $\overline{\text{INTA}}$ indicates that the interrupt number should be placed onto the lower byte of the processor's data bus. A special peripheral designed to respond to the 8088's interrupt acknowledge cycle is the 8259A programmable interrupt controller, which we will cover in Chapter 11.

In the Pentium, output signals M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, W/$\overline{\text{R}}$, and $\overline{\text{ADS}}$ all go low to indicate an interrupt acknowledge cycle.

Once the interrupt number is read from the data bus, the processor performs all of the steps that we are familiar with. Let us review the overall process once more.

Get Vector Number

The processor obtains the interrupt number in one of three ways. First, the interrupt type number may be specified directly using one of the INT instructions. Second, the processor may automatically generate the interrupt type number, as it does for INTO, NMI, and divide-error. Third, it may have to read the interrupt type number from the data bus (after receiving INTR).

Once the interrupt number is obtained, it is used to form the location within the interrupt vector table that contains the requested ISR address.

Save Processor Information

Once the interrupt vector is known, the processor pushes the flag register onto the stack. This is done to preserve some of the processor's internal state at the time of the interrupt (a very necessary step if we are to resume normal execution later). Once the flags are pushed,

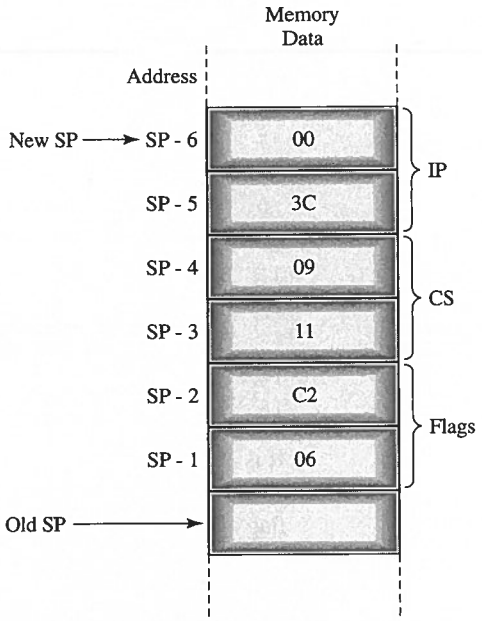

FIGURE 5.3 Stack contents during an interrupt

the processor clears the interrupt enable and trace flags to disable INTR while interrupt processing is taking place. Next, the IP and CS values at the time of the interrupt are pushed onto the stack. Figure 5.3 shows how the stack is used when an interrupt occurs. The contents of the flag register at the time of the interrupt were 06C2H. The address of the instruction that was about to be fetched when the interrupt occurred was 1109:3C00 (in CS:IP format).

Fetch New Instruction Pointer

Once the return address has been pushed, the processor can fetch the new values of IP and CS out of the interrupt vector table and begin execution of the interrupt service routine. The address generated by the interrupt number is used to read the two ISR address words out of the table.

One word of caution: Because the stack contains the information needed to return to the interrupt point, we must be careful not to change the contents of stack memory or alter the stack pointer in any way that would prevent the correct information from being popped off. The processor will not remember anything about the interrupt and relies only on the data popped off the stack for a proper return.

5.5 MULTIPLE INTERRUPTS

In the course of program execution, chances are good that eventually two interrupts might request the processor's attention at the same time. For example, just as division-by-zero is attempted in an executing program, NMI is also activated. The processor needs to "break the tie" when this happens and recognize one of the two interrupts first. When we examined Table 5.1, we saw that divide-error has a higher priority than NMI. So, in our current example, divide-error will be recognized first, and the following sequence of steps will occur:

1. Divide-error is recognized.
2. The flags are pushed.