Intel reserves the first 32 interrupt vectors for the present and future microprocessor products. The remaining interrupt vectors (32–255) are available for the user. Some of the reserved vectors are for errors that occur during the execution of software, such as the divide error interrupt. Some vectors are reserved for the coprocessor. Still others occur for normal events in the system. In a personal computer, the reserved vectors are used for system functions, as detailed later in this section. Vectors 1–6, 7, 9, 16, and 17 function in the real mode and protected mode; the remaining vectors function only in the protected mode.

Interrupt Instructions

The microprocessor has three different interrupt instructions that are available to the programmer: INT, INTO, and INT 3. In the real mode, each of these instructions fetches a vector from the vector table, and then calls the procedure stored at the location addressed by the vector. In the protected mode, each of these instructions fetches an interrupt descriptor from the interrupt descriptor table. The descriptor specifies the address of the interrupt service procedure. The interrupt call is similar to a far CALL instruction because it places the return address (IP/EIP and CS) on the stack.

INTs. There are 256 different software interrupt instructions (INTs) available to the programmer. Each INT instruction has a numeric operand whose range is 0 to 255 (00H–FFH). For example, the INT 100 uses interrupt vector 100, which appears at memory address 190H–193H. The address of the interrupt vector is determined by multiplying the interrupt type number by 4. For example, the INT 10H instruction calls the interrupt service procedure whose address is stored beginning at memory location 40H (10H × 4) in the real mode. In the protected mode, the interrupt descriptor is located by multiplying the type number by 8 instead of 4 because each descriptor is 8 bytes long.

Each INT instruction is 2 bytes long. The first byte contains the opcode, and the second byte contains the vector type number. The only exception to this is INT 3, a 1-byte special software interrupt used for breakpoints.

Whenever a software interrupt instruction executes, it (1) pushes the flags onto the stack, (2) clears the T and I flag bits, (3) pushes CS onto the stack, (4) fetches the new value for CS from the interrupt vector, (5) pushes IP/EIP onto the stack, (6) fetches the new value for IP/EIP from the vector, and (7) jumps to the new location addressed by CS and IP/EIP.

The INT instruction performs as a far CALL except that it not only pushes CS and IP onto the stack, but it also pushes the flags onto the stack. The INT instruction performs the operation of a PUSHF, followed by a far CALL instruction.

Notice that when the INT instruction executes, it clears the interrupt flag (I), which controls the external hardware interrupt input pin INTR (interrupt request). When I = 0, the microprocessor disables the INTR pin; when I = 1, the microprocessor enables the INTR pin.

Software interrupts are most commonly used to call system procedures because the address of the system function need not be known. The system procedures are common to all system and application software. The interrupts often control printers, video displays, and disk drives. Besides relieving the program from remembering the address of the system call, the INT instruction replaces a far CALL that would otherwise be used to call a system function. The INT instruction is 2 bytes long, whereas the far CALL is 5 bytes long. Each time that the INT instruction replaces a far CALL, it saves 3 bytes of memory in a program. This can amount to a sizable saving if the INT instruction often appears in a program, as it does for system calls.

IRET/IRETD. The interrupt return instruction (IRET) is used only with software or hardware interrupt service procedures. Unlike a simple return instruction (RET), the IRET instruction will (1) pop stack data back into the IP, (2) pop stack data back into CS, and (3) pop stack data back into the flag register. The IRET instruction accomplishes the same tasks as the POPF, followed by a far RET instruction.

Whenever an IRET instruction executes, it restores the contents of I and T from the stack. This is important because it preserves the state of these flag bits. If interrupts were enabled before an interrupt service procedure, they are automatically re-enabled by the IRET instruction because it restores the flag register.

In the 80386 through the Core2 processors, the IRETD instruction is used to return from an interrupt service procedure that is called in the protected mode. It differs from the IRET because it pops a 32-bit instruction pointer (EIP) from the stack. The IRET is used in the real mode and the IRETD is used in the protected mode.

INT 3. An INT 3 instruction is a special software interrupt designed to function as a breakpoint. The difference between it and the other software interrupts is that INT 3 is a 1-byte instruction, while the others are 2-byte instructions.

It is common to insert an INT 3 instruction in software to interrupt or break the flow of the software. This function is called a breakpoint. A breakpoint occurs for any software interrupt, but because INT 3 is 1 byte long, it is easier to use for this function. Breakpoints help to debug faulty software.

INTO. Interrupt on overflow (INTO) is a conditional software interrupt that tests the overflow flag (O). If O = 0, the INTO instruction performs no operation; if O = 1 and an INTO instruction executes, an interrupt occurs via vector type number 4.

The INTO instruction appears in software that adds or subtracts signed binary numbers. With these operations, it is possible to have an overflow. Either the JO instruction or INTO instruction detects the overflow condition.

An Interrupt Service Procedure. Suppose that, in a particular system, a procedure is required to add the contents of DI, SI, BP, and BX and then save the sum in AX. Because this is a common task in this system, it may occasionally be worthwhile to develop the task as a software interrupt. Realize that interrupts are usually reserved for system events and this is merely an example showing how an interrupt service procedure appears. Example 6–18 shows this software interrupt. The main difference between this procedure and a normal far procedure is that it ends with the IRET instruction instead of the RET instruction, and the contents of the flag register are saved on the stack during its execution. It is also important to save all registers that are changed by the procedure using USES.

EXAMPLE 6–18

```
0000                INTS    PROC  FAR USES AX
0000 03 C3                  ADD   AX,BX
0002 03 05                  ADD   AX,BP
0004 03 C7                  ADD   AX,DI
0006 03 C6                  ADD   AX,SI
0008 CF                     IRET
0009                INTS    ENDP
```

Interrupt Control

Although this section does not explain hardware interrupts, two instructions are introduced that control the INTR pin. The **set interrupt flag** instruction (STI) places a 1 into the I flag bit, which enables the INTR pin. The **clear interrupt flag** instruction (CLI) places a 0 into the I flag bit, which disables the INTR pin. The STI instruction enables INTR and the CLI instruction disables INTR. In a software interrupt service procedure, hardware interrupts are enabled as one of the first steps. This is accomplished by the STI instruction. The reason interrupts are enabled early in an interrupt service procedure is that just about all of the I/O devices in the personal computer are interrupt-processed. If the interrupts are disabled too long, severe system problems result.

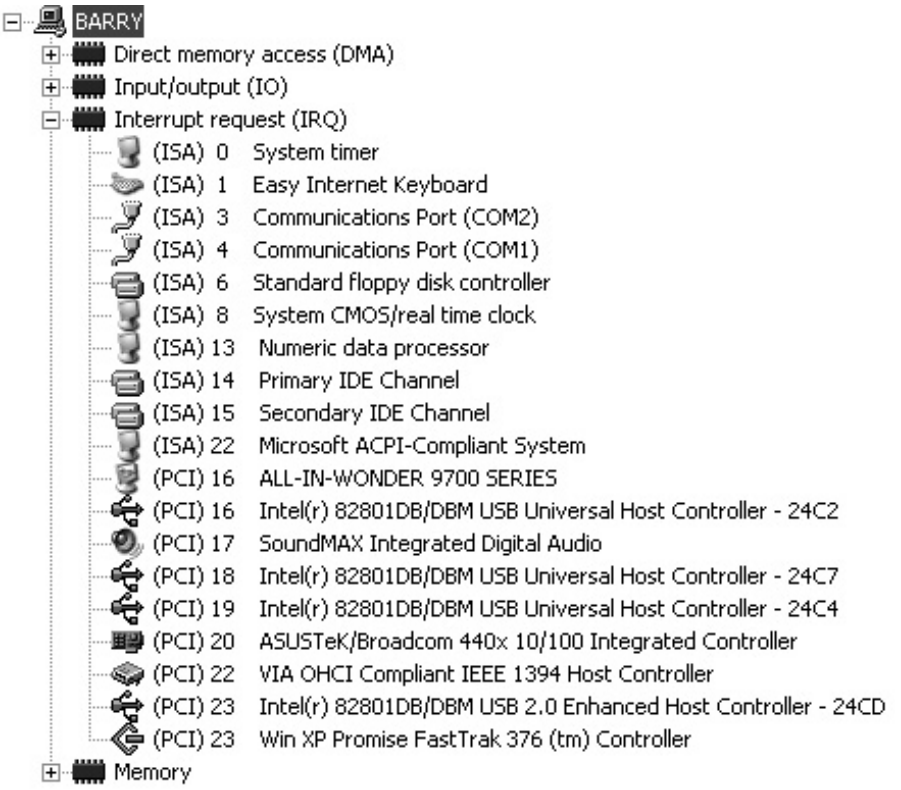

FIGURE 6–9 Interrupts in a typical personal computer.

Interrupts in the Personal Computer

The interrupts found in the personal computer differ somewhat from the ones presented in Table 6–4. The reason that they differ is that the original personal computers are 8086/8088-based systems. This meant that they only contained Intel-specified interrupts 0–4. This design has been carried forward so that newer systems are compatible with the early personal computers.

Access to the protected mode interrupt structure in use by Windows is accomplished through kernel functions Microsoft provides and cannot be directly addressed. Protected mode interrupts use an interrupt descriptor table, which is beyond the scope of the text at this point. Protected mode interrupts are discussed completely in later chapters.

Figure 6–9 illustrates the interrupts available in the author's computer. The interrupt assignments are viewable in the control panel of Windows under Performance and Maintenance by clicking on System and selecting Hardware and then Device Manager. Now click on View and select Device by Type and finally Interrupts.

64-Bit Mode Interrupts

The 64-bit system uses the IRETQ instruction to return from an interrupt service procedure. The main difference between IRET/IRETD and the IRETQ instruction is that IRETQ retrieves an 8-byte return address from the stack. The IRETQ instruction also retrieves the 32-bit EFLAG register from the stack and places it into the RFLAG register. It appears that Intel has no plans for using the leftmost 32 bits of the RFLAG register. Otherwise, 64-bit mode interrupts are the same as 32-bit mode interrupts.

6–5 MACHINE CONTROL AND MISCELLANEOUS INSTRUCTIONS

The last category of real mode instructions found in the microprocessor is the machine control and miscellaneous group. These instructions provide control of the carry bit, sample the BUSY/TEST pin, and perform various other functions. Because many of these instructions are used in hardware control, they need only be explained briefly at this point.

Controlling the Carry Flag Bit

The carry flag (C) propagates the carry or borrow in multiple-word/doubleword addition and subtraction. It also can indicate errors in assembly language procedures. Three instructions control the contents of the carry flag: STC (set carry), CLC (clear carry), and CMC (complement carry).

Because the carry flag is seldom used except with multiple-word addition and subtraction, it is available for other uses. The most common task for the carry flag is to indicate an error upon return from a procedure. Suppose that a procedure reads data from a disk memory file. This operation can be successful, or an error such as file-not-found can occur. Upon return from this procedure, if C = 1, an error has occurred; if C = 0, no error occurred. Most of the DOS and BIOS procedures use the carry flag to indicate error conditions. This flag is not available in Visual C/C++ for use with C++.

WAIT

The WAIT instruction monitors the hardware $\overline{BUSY}$ pin on the 80286 and 80386, and the $\overline{TEST}$ pin on the 8086/8088. The name of this pin was changed beginning with the 80286 microprocessor from $\overline{TEST}$ to $\overline{BUSY}$. If the WAIT instruction executes while the $\overline{BUSY}$ pin = 1, nothing happens and the next instruction executes. If the $\overline{BUSY}$ pin = 0 when the WAIT instruction executes, the microprocessor waits for the $\overline{BUSY}$ pin to return to a logic 1. This pin inputs a busy condition when at a logic 0 level.

The $\overline{BUSY}/\overline{TEST}$ pin of the microprocessor is usually connected to the $\overline{BUSY}$ pin of the 8087 through the 80387 numeric coprocessors. This connection allows the microprocessor to wait until the coprocessor finishes a task. Because the coprocessor is inside an 80486 through the Core2, the $\overline{BUSY}$ pin is not present in these microprocessors.

HLT

The halt instruction (HLT) stops the execution of software. There are three ways to exit a halt: by an interrupt, by a hardware reset, or during a DMA operation. This instruction normally appears in a program to wait for an interrupt. It often synchronizes external hardware interrupts with the software system. Note that DOS and Windows both use interrupts extensively, so HLT will not halt the computer when operated under these operating systems.

NOP

When the microprocessor encounters a no operation instruction (NOP), it takes a short time to execute. In early years, before software development tools were available, a NOP, which performs absolutely no operation, was often used to pad software with space for future machine language instructions. If you are developing machine language programs, which are extremely rare, it is recommended that you place 10 or so NOPS in your program at 50-byte intervals. This is done in case you need to add instructions at some future point. A NOP may also find application in time delays to waste time. Realize that a NOP used for timing is not very accurate because of the cache and pipelines in modem microprocessors.

LOCK Prefix

The LOCK prefix appends an instruction and causes the $\overline{LOCK}$ pin to become a logic 0. The $\overline{LOCK}$ pin often disables external bus masters or other system components. The LOCK prefix causes the $\overline{LOCK}$ pin to activate for only the duration of a locked instruction. If more than one sequential instruction is locked, the $\overline{LOCK}$ pin remains a logic 0 for the duration of the sequence of locked instructions. The LOCK:MOV AL,[SI] instruction is an example of a locked instruction.

ESC

The escape (ESC) instruction passes instructions to the floating-point coprocessor from the microprocessor. Whenever an ESC instruction executes, the microprocessor provides the memory address, if required, but otherwise performs a NOP. Six bits of the ESC instruction provide the opcode to the coprocessor and begin executing a coprocessor instruction.

The ESC opcode never appears in a program as ESC and in itself is considered obsolete as an opcode. In its place are a set of coprocessor instructions (FLD, FST, FMUL, etc.) that assemble as ESC instructions for the coprocessor. More detail is provided in Chapter 13, which details the 8087–Core2 numeric coprocessors.

BOUND

The BOUND instruction, first made available in the 80186 microprocessor, is a comparison instruction that may cause an interrupt (vector type number 5). This instruction compares the contents of any 16-bit or 32-bit register against the contents of two words or doublewords of memory: an upper and a lower boundary. If the value in the register compared with memory is not within the upper and lower boundary, a type 5 interrupt ensues. If it is within the boundary, the next instruction in the program executes.

For example, if the BOUND SI,DATA instruction executes, word-sized location DATA contains the lower boundary, and word-sized location DATA+2 bytes contains the upper boundary. If the number contained in SI is less than memory location DATA or greater than memory location DATA+2 bytes, a type 5 interrupt occurs. Note that when this interrupt occurs, the return address points to the BOUND instruction, not to the instruction following BOUND. This differs from a normal interrupt, where the return address points to the next instruction in the program.

ENTER and LEAVE

The ENTER and LEAVE instructions, first made available to the 80186 microprocessor, are used with stack frames, which are mechanisms used to pass parameters to a procedure through the stack memory. The stack frame also holds local memory variables for the procedure. Stack frames provide dynamic areas of memory for procedures in multiuser environments.

The ENTER instruction creates a stack frame by pushing BP onto the stack and then loading BP with the uppermost address of the stack frame. This allows stack frame variables to be accessed through the BP register. The ENTER instruction contains two operands: The first operand specifies the number of bytes to reserve for variables on the stack frame, and the second specifies the level of the procedure.

Suppose that an ENTER 8,0 instruction executes. This instruction reserves 8 bytes of memory for the stack frame and the zero specifies level 0. Figure 6–10 shows the stack frame set up by this instruction. Note that this instruction stores BP onto the top of the stack. It then subtracts 8 from the stack pointer, leaving 8 bytes of memory space for temporary data storage. The uppermost location of this 8-byte temporary storage area is addressed by BP. The LEAVE instruction reverses this process by reloading both SP and BP with their prior values. The ENTER and LEAVE instructions were used to call C++ functions in Windows 3.1, but since then, CALL has been used in modern versions of Windows for C++ functions.

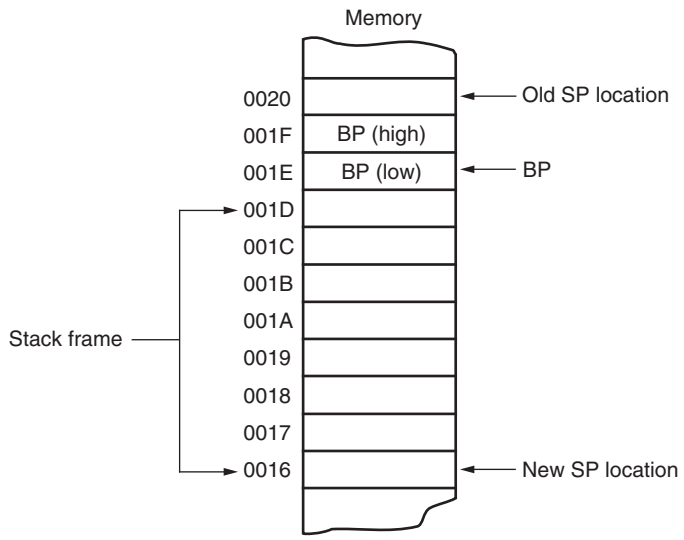

FIGURE 6–10 The stack frame created by the ENTER 8,0 instruction. Notice that BP is stored beginning at the top of the stack frame. This is followed by an 8-byte area called a stack frame.

6–6 SUMMARY

1. There are three types of unconditional jump instructions: short, near, and far. The short jump allows a branch to within +127 and −128 bytes. The near jump (using a displacement of ±32K) allows a jump to any location in the current code segment (intrasegment). The far jump allows a jump to any location in the memory system (intersegment). The near jump in an 80386 through a Core2 is within ±2G bytes because these microprocessors can use a 32-bit signed displacement.
2. Whenever a label appears with a JMP instruction or conditional jump, the label, located in the label field, must be followed by a colon (LABEL:). For example, the JMP DOGGY instruction jumps to memory location DOGGY:.
3. The displacement that follows a short or near jump is the distance from the next instruction to the jump location.
4. Indirect jumps are available in two forms: (1) jump to the location stored in a register and (2) jump to the location stored in a memory word (near indirect) or doubleword (far indirect).
5. Conditional jumps are all short jumps that test one or more of the flag bits: C, Z, O, P, or S. If the condition is true, a jump occurs; if the condition is false, the next sequential instruction executes. Note that the 80386 and above allow a 16-bit signed displacement for the conditional jump instructions. In 64-bit mode, the displacement is 32 bits allowing a range of ±2G.
6. A special conditional jump instruction (LOOP) decrements CX and jumps to the label when CX is not 0. Other forms of loop include LOOPE, LOOPNE, LOOPZ, and LOOPNZ. The LOOPE instruction jumps if CX is not 0 and if an equal condition exists. In the 80386 through the Core2, the LOOPD, LOOPED, and LOOPNED instructions also use the ECX register as a counter. In the 64-bit mode, these instructions use the RCX register as for iteration.
7. The 80386 through the Core2 contain conditional set instructions that either set a byte to 01H or clear it to 00H. If the condition under test is true, the operand byte is set to 01H; if the condition under test is false, the operand byte is cleared to 00H.