CHAPTER 12

Interrupts

INTRODUCTION

In this chapter, we expand our coverage of basic I/O and programmable peripheral interfaces by examining a technique called interrupt-processed I/O. An **interrupt** is a hardware-initialed procedure that interrupts whatever program is currently executing.

This chapter provides examples and a detailed explanation of the interrupt structure of the entire Intel family of microprocessors.

CHAPTER OBJECTIVES

Upon completion of this chapter, you will be able to:

1. Explain the interrupt structure of the Intel family of microprocessors.
2. Explain the operation of software interrupt instructions INT, INTO, INT 3, and BOUND.
3. Explain how the interrupt enable flag bit (IF) modifies the interrupt structure.
4. Describe the function of the trap interrupt flag-bit (TF) and the operation of trap-generated tracing.
5. Develop interrupt-service procedures that control lower-speed, external peripheral devices.
6. Expand the interrupt structure of the microprocessor by using the 8259A programmable interrupt controller and other techniques.
7. Explain the purpose and operation of a real-time clock.

12-1 BASIC INTERRUPT PROCESSING

In this section, we discuss the function of an interrupt in a microprocessor-based system, and the structure and features of interrupts available to the Intel family of microprocessors.

The Purpose of Interrupts

Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data-transfer rates. In Chapter 11, for instance, we showed a keyboard example using strobed input operation of the 82C55. In that example, software polled the 82C55 and its

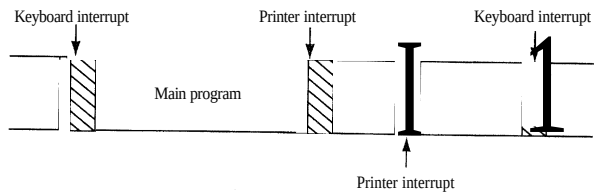


FIGURE 12-1 A time line that indicates interrupt usage in a typical system.

IBF bit to decide whether data were available from the keyboard. If the person using the keyboard typed one character per second, the software for the 82C55 waited an entire second between each keystroke for the person to type another key. This process was such a tremendous waste of time that designers developed another process, *interrupt processing,* to handle this situation.

Unlike the polling technique, interrupt processing allows the microprocessor to execute other software while the keyboard operator is thinking about what key to type next. As soon as a key is pressed, the keyboard encoder de-bounces the switch and puts out one pulse that interrupts the microprocessor. In this way, the microprocessor executes other software until the key is actually pressed when it reads a key and returns to the program that was interrupted. As a result, the microprocessor can print reports or complete any other task while the operator is typing a document and thinking about what to type next.

Figure 12-1 shows a time line that indicates a typist typing data on a keyboard, a printer removing data from the memory, and a program executing. The program is the main program that is interrupted for each keystroke and each character that is to print on the printer. Note that the keyboard interrupt service procedure, called by the keyboard interrupt, and the printer interrupt service procedure each take little time to execute.

Interrupts

The interrupts of the entire Intel family of microprocessors include two hardware pins that request interrupts (INTR and NMI), and one hardware pin (INTA) that acknowledges the interrupt requested through INTR. In addition to the pins, the microprocessor also has software interrupts INT, INTO, INT 3, and BOUND. Two flag bits, IF (interrupt flag) and TF (trap flag), are also used with the interrupt structure and a special return instruction IRET (or IRETD in the 80386, 80486, or Pentium-Pentium II).

Interrupt Vectors. The interrupt vectors and vector table are crucial to an understanding of hardware and software interrupts. The **interrupt vector table** is located in the first 1024 bytes of memory at addresses OOOOOOH-0003FFH. It contains 256 different 4-byte interrupt vectors. An **interrupt vector** contains the address (segment and offset) of the interrupt service procedure.

Figure 12-2 illustrates the interrupt vector table for the microprocessor. The first five interrupt vectors are identical in all Intel microprocessor family members, from the 8086 to the Pentium. Other interrupt vectors exist for the 80286 that are upward-compatible to the 80386, 80486, and Pentium-Pentium II, but not downward-compatible to the 8086 or 8088. Intel reserves the first 32 interrupt vectors for their use in various microprocessor family members. The last 224 vectors are available as user interrupt vectors. Each vector is four bytes long and contains the **starting address** of the interrupt service procedure. The first two bytes of the vector contain the offset address, and the last two bytes contain the segment address.

The following list describes the function of each dedicated interrupt in the microprocessor:

Type 0 Divide Error—Occurs whenever the result of a division overflows or whenever an attempt is made to divide by zero.

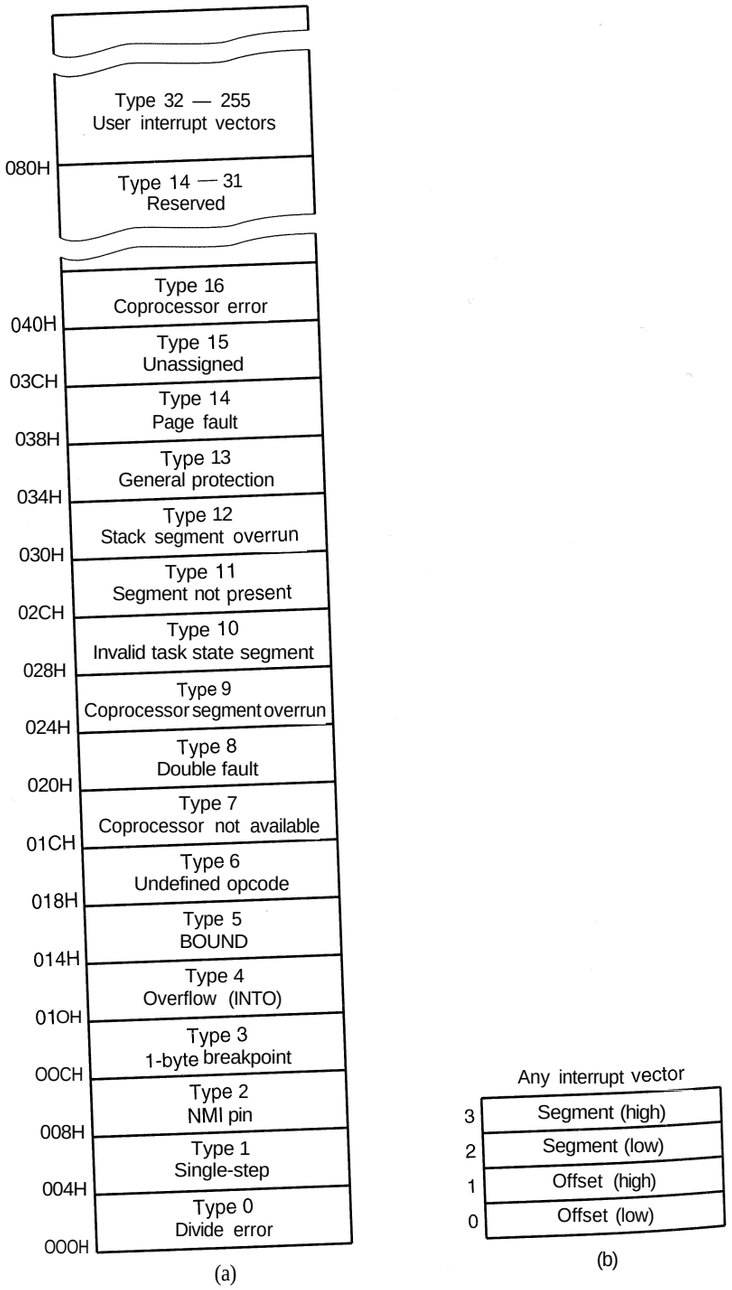


FIGURE 12-2 (a) The interrupt vector table for the microprocessor, and (b) the contents of an interrupt vector.

Type 1 Single-step or Trap—Occurs after the execution of each instruction if the trap (TF) flag bit is set. Upon accepting this interrupt, the TF bit is cleared so that the interrupt service procedure executes at full speed. (More detail is provided about this interrupt later in this section of the chapter.)

Type 2 Non-maskable Hardware Interrupt—A result of placing a logic 1 on the NMI input pin to the microprocessor. This input is non-maskable, which means that it cannot be disabled.

Type 3 One-Byte Interrupt—A special one-byte instruction (INT 3) that uses this vector to access its interrupt-service procedure. The INT 3 instruction is often used to store a breakpoint in a program for debugging.

Type 4 Overflow—A special vector used with the INTO instruction. The INTO instruction interrupts the program if an overflow condition exists, as reflected by the overflow flag (OF).

Type 5 BOUND—An instruction that compares a register with boundaries stored in the memory. If the contents of the register are greater than or equal to the first word in memory and less than or equal to the second word, no interrupt occurs because the contents of the register is within bounds. If the contents of the register are out-of-bounds, a type 5 interrupt ensues.

Type 6 Invalid Opcode—Occurs whenever an undefined opcode is encountered in a program.

Type 7 Coprocessor Not Available—Occurs when a coprocessor is not found in the system, as dictated by the machine status word (MSW) coprocessor control bits. If an ESC or WAIT instruction executes and the coprocessor is not found, a type 7 exception or interrupt occurs.

Type 8 Double Fault—Activated whenever two separate interrupts occur during the same instruction.

Type 9 Coprocessor Segment Overrun—Occurs if the ESC instruction (coprocessor opcode) memory operand extends beyond offset address FFFFH.

Type 10 Invalid Task State Segment—Occurs if the TSS is invalid because the segment limit field is not 002BH or higher. In most cases, this is caused because the TSS is not initialized.

Type 11 Segment not Present—Occurs when the P bit (P = 0) in a descriptor indicates that the segment is not present or not valid.

Type 12 Stack Segment Overrun—Occurs if the stack segment is not present (P = 0) or if the limit of the stack segment is exceeded.

Type 13 General Protection—Occurs for most protection violations in the 80286–Pentium II protected mode system. (These errors occur in Windows as **general protection faults.)** A list of these protection violations follows:

a. Descriptor table limit exceeded
b. Privilege rules violated
c. Invalid descriptor segment type loaded
d. Write to code segment that is protected
e. Read from execute-only code segment
f. Write to read-only data segment
g. Segment limit exceeded
h. CPL = IOPL when executing CTS, HLT, LGDT, LIDT, LLDT, LMSW, or LTR
i. CPL > IOPL when executing CLI, IN, INS, LOCK, OUT, OUTS, and STI

Type 14 Page Fault—Occurs for any page fault memory or code access in the 80386, 80486, and Pentium-Pentium II microprocessors.

Type 16 Coprocessor Error—Takes effect whenever a coprocessor error ($\overline{ERROR}$ = 0) occurs for the ESCape or WAIT instructions for the 80386, 80486, and Pentium-Pentium II microprocessors only.

Type 17 Alignment Check—Indicates that word and doubleword data are addressed at an odd memory location (or an incorrect location, in the case of a doubleword). This interrupt is active in the 80486 and Pentium-Pentium II microprocessors.

Type 18 Machine Check—Activates a system memory management mode interrupt in the Pentium-Pentium II microprocessors.

Interrupt Instructions: BOUND, INTO, INT, INT 3, and IRET

Of the five software interrupt instructions available to the microprocessor, INT and INT 3 are very similar, BOUND and INTO are conditional, and IRET is a special interrupt return instruction.

The BOUND instruction, which has two operands, compares a register with two words of memory data. For example, if the instruction BOUND AX,DATA is executed, AX is compared with the contents of DATA and DATA+1 and also with DATA+2 and DATA+3. If AX is less than the contents of DATA and DATA+1, a type 5 interrupt occurs. If AX is greater than DATA+2 and DATA+3, a type 5 interrupt occurs. If AX is within the bounds of these two memory words, no interrupt occurs.

The INTO instruction checks the overflow flag (OF). If OF = 1, the INTO instruction calls the procedure whose address is stored in interrupt vector type number 4. If OF = 0, then the INTO instruction performs no operation and the next sequential instruction in the program executes.

The INT n instruction calls the interrupt service procedure that begins at the address represented in vector number n. For example, an INT 80H or INT 128 calls the interrupt service procedure whose address is stored in vector type number 80H (000200H-00203H). To determine the vector address, just multiply the vector type number (n) by 4, which gives the beginning address of the 4-byte long interrupt vector. For example, an INT 5 = 4 x 5 or 20 (14H). The vector for INT 5 begins at address 000014H and continues to 000017H. Each INT instruction is stored in two bytes of memory: the first byte contains the opcode, and the second byte contains the interrupt type number. The only exception to this is the INT 3 instruction, a 1-byte instruction. The INT 3 instruction is often used as a breakpoint-interrupt because it is easy to insert a 1-byte instruction into a program. Breakpoints are often used to debug faulty software.

The IRET instruction is a special return instruction used to return for both software and hardware interrupts. The IRET instruction is much like a far RET, because it retrieves the return address from the stack. It is unlike the near return because it also retrieves a copy of the flag register from the stack. An IRET instruction removes six bytes from the stack: two for the IP, two for the CS, and two for the flags.

In the 80386–Pentium II, there is also an IRETD instruction because these microprocessors can push the EFEAG register (32 bits) on the stack, as well as the 32-bit EIP in the protected mode. If operated in the real mode, we use the IRET instruction with the 80386-Pentium II microprocessors.

The Operation of a Real Mode Interrupt

When the microprocessor completes executing the current instruction, it determines whether an interrupt is active by checking (1) instruction executions, (2) single-step, (3) NMI, (4) coprocessor segment overrun, (5) INTR, and (6) INT instruction in the order presented. If one or more of these interrupt conditions are present, the following sequence of events occurs:

1. The contents of the flag register are pushed onto the stack.
2. Both the interrupt (IF) and trap (TF) flags are cleared. This disables the INTR pin and the trap or single-step feature.
3. The contents of the code segment register (CS) are pushed onto the stack.
4. The contents of the instruction pointer (IP) are pushed onto the stack.
5. The interrupt vector contents are fetched, and then placed into both IP and CS so that the next instruction executes at the interrupt service procedure addressed by the vector.

Whenever an interrupt is accepted, the microprocessor stacks the contents of the flag register, CS and IP; clears both IF and TF; and jumps to the procedure addressed by the interrupt vector. After the flags are pushed onto the stack, IF and TF are cleared. These flags are returned to the state prior to the interrupt when the IRET instruction is encountered at the end of the interrupt service procedure. Therefore, if interrupts were enabled prior to the interrupt service procedure, they are automatically re-enabled by the IRET instruction at the end of the procedure.

The return address (in CS and IP) is pushed onto the stack during the interrupt. Sometimes, the return address points to the next instruction in the program; sometimes it points to the instruction or point in the program where the interrupt occurred. Interrupt type numbers 0, 5, 6, 7, 8, 10, 11, 12, and 13 push a return address that points to the offending instruction, instead of to the next instruction in the program. This allows the interrupt service procedure to possibly retry the instruction in certain error cases.

Some of the protected mode interrupts (types 8, 10, 11, 12, and 13) place an error code on the stack following the return address. The error code identifies the selector that caused the interrupt. In cases where no selector is involved, the error code is a 0.

Operation of a Protected Mode Interrupt

In the protected mode, interrupts have exactly the same assignments as in the real mode, but the interrupt vector table is different. In place of interrupt vectors, protected mode uses a set of 256 interrupt descriptors that are stored in an interrupt descriptor table (IDT). The interrupt descriptor table is 256 x 8 (2K) bytes long, with each descriptor containing eight bytes. The interrupt descriptor table is located at any memory location in the system by the interrupt descriptor table address register (IDTR).

Each entry in the IDT contains the address of the interrupt service procedure in the form of a segment selector and a 32-bit offset address. It also contains the P bit (present) and DPL bits to describe the privilege level of the interrupt. Figure 12-3 shows the contents of the interrupt descriptor.

Real mode interrupt vectors can be converted into protected mode interrupts by copying the interrupt procedure addresses from the interrupt vector table and converting them to 32-bit offset addresses that are stored in the interrupt descriptors. A single selector and segment descriptor can be placed in the global descriptor table that identifies the first 1M byte of memory as the interrupt segment.

Other than the IDT and interrupt descriptors, the protected mode interrupt functions like the real mode interrupt. We return from both interrupts by using the IRET or IRETD instruction. The only difference is that in protected mode the microprocessor accesses the IDT instead of the interrupt vector table.

Interrupt Flag Bits

The interrupt flag (IF) and the trap flag (TF) are both cleared after the contents of the flag register are stacked during an interrupt. Figure 12-4 illustrates the contents of the flag register and

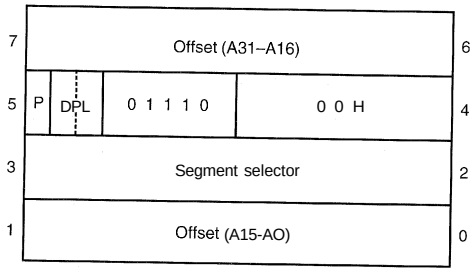


FIGURE 12-3 The protected mode interrupt descriptor.

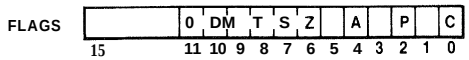


FIGURE 12-4 The flag register. (Courtesy of Intel Corporation.)

the location of IF and TF. When the IF bit is set, it allows the INTR pin to cause an interrupt; when the IF bit is cleared, it prevents the INTR pin from causing an interrupt. When TF = 1, it causes a trap interrupt (type number 1) to occur after each instruction executes. This is why we often call trap a *single-step*. When TF = 0, normal program execution occurs. This flag bit allows debugging, as explained in Chapters 17-19, which detail the 80386-Pentium II.

The interrupt flag is set and cleared by the STI and CLI instructions, respectively. There are no special instructions that set or clear the trap flag. Example 12-1 shows an interrupt service procedure that turns tracing on by setting the trap flag bit on the stack from inside the procedure. Example 12-2 shows an interrupt service procedure that turns tracing off by clearing the trap flag on the stack from within the procedure.

EXAMPLE 12-1

```
                        ;A procedure that sets TF to enable trap.
                        ;
0000                    TRON   PROC   NEAR

0000  50                       PUSH   AX              ;save registers
0001  55                       PUSH   BP
0002  8B EC                    MOV    BP,SP           ;get SP
0004  8B 46 08                 MOV    AX,[BP+8]       ;get flags from stack
0007  80 CC 01                 OR     AH,1            ;set TF
000A  89 46 08                 MOV    [BP+8],AX       ;save flags
000D  5D                       POP    BP              ;restore registers
000E  58                       POP    AX
000F  CF                       IRET

0010                    TRON   ENDP
```

EXAMPLE 12-2

```
                        ;A procedure that clears TF to disable trap.
                        ;
0000                    TROFF  PROC   NEAR

0000  50                       PUSH   AX              ;save registers
0001  55                       PUSH   BP
0002  8B EC                    MOV    BP,SP           ;get SP
0004  8B 46 08                 MOV    AX,[BP+8]       ;get TF
0007  80 E4 FE                 AND    AH,0FEH         ;clear TF
000A  89 46 08                 MOV    [BP+8],AX       ;save flags
000D  5D                       POP    BP              ;restore registers
000E  58                       POP    AX
000F  CF                       IRET

0010                    TROFF  ENDP
```

In both examples, the flag register is retrieved from the stack by using the BP register, which, by default, addresses the stack segment. After the flags are retrieved, the TF bit is either set (TRON) or clears (TROFF) before returning from the interrupt service procedure. The IRET instruction restores the flag register with the new state of the trap flag.

Trace Procedure. Assuming that TRON is accessed by an INT 40H instruction and TROFF is accessed by an INT 41H instruction, Example 12-3 traces through a program immediately following the INT 40H instruction. The interrupt service procedure illustrated in Example 12-3 responds to interrupt type number 1 or a trap interrupt. Each time that a trap occurs—after each

instruction executes following INT 40H-the TRACE procedure displays the contents of all the 16-bit microprocessor registers on the CRT screen. This provides a register trace of all the instructions between the INT 40H (TRON) and INT 41H (TROFF).

EXAMPLE 12-3

```
                        .MODEL TINY
                        .CODE
0000                    RNAME DB  'AX = ','BX = ','CX = ','DX = '
0000  41 58 20 3D 20 42
      58 20 3D 20 43 58
      20 3D 20 44 58 20
      3D 20
0014  53 50 20 3D 20 42         DB  'SP = ','BP = ','SI = ','DI = '
      50 20 3D 20 53 49
      0 3D 20 44 49 20
      3D 20
0028  49 50 20 3D 20 46         DB  'IP = ','FL = ','CS = ','DS = '
      4C 20 3D 20 43 53
      20 3D 20 44 53 20
      3D 20
003C  45 53 20 3D 20 53         DB  'ES = ','SS = '
      53 20 3D 20

                        DISP   MACRO PAR1
                               PUSH  AX
                               PUSH  DX
                               MOV   DL,PAR1
                               MOV   AH,6
                               INT   21H
                               POP   DX
                               POP   AX
                               ENDM

                        CRLF   MACRO
                               DISP  13
                               DISP  10
                               ENDM

0046                    TRACE  PROC  FAR USES AX BP BX

0049  BB 0000 R                 MOV   BX,OFFSET RNAME   ;address names
                               CRLF
0060  E8 004D                  CALL  DREG              ;display AX
0063  58                       POP   AX                ;get BX
0064  50                       PUSH  AX
0065  E8 0048                  CALL  DREG              ;display BX
0068  8B C1                    MOV   AX,CX
006A  E8 0043                  CALL  DREG              ;display CX
006D  8B C2                    MOV   AX,DX
006F  E8 003E                  CALL  DREG              ;display DX
0072  8B C4                    MOV   AX,SP
0074  83 C0 0C                 ADD   AX,12
0077  E8 0036                  CALL  DREG              ;display SP
007A  8B C5                    MOV   AX,BP
007C  E8 0031                  CALL  DREG              ;display BP
007F  8B C6                    MOV   AX,SI
0081  E8 002C                  CALL  DREG              ;display SI
0084  8B C7                    MOV   AX,DI
0086  E8 0027                  CALL  DREG              ;display DI
0089  8B EC                    MOV   BP,SP
008B  8B 46 06                 MOV   AX,[BP+6]
008E  E8 001F                  CALL  DREG              ;display IP
0091  8B 46 0A                 MOV   AX,[BP+10]
0094  E8 0019                  CALL  DREG              ;display Flags
0097  8B 46 08                 MOV   AX,[BP+8]
009A  E8 0013                  CALL  DREG              ;display CX
009D  8C D8                    MOV   AX,DS
```

```
009F  E8 000E                    CALL   DREG              ;display DS
00A2  8C C0                      MOV    AX,ES
00A4  E8 0009                    CALL   DREG              ;display ES
00A7  8C D0                      MOV    AX,SS
00A9  E8 0004                    CALL   DREG              ;display SS
                                 IRET

00B0                        TRACE ENDP

00B0                        DREG  PROC   NEAR USES CX
00B1  B9 0005                     MOV    CX,5            ; load count
00B4                  DREG1:
                                 DISP  CS: [BX]          ;display character
00BF  43                         INC    BX               ;address next
00C0  E2 F2                      LOOP   DREG1            ;repeat 5 times
00C2  B9 0004                    MOV   CX, 4             ;load count
00C5                  DREG2:
00C5  D3 C8                      ROL    AX,1             ;position digit
00C7  D3 C8                      ROL    AX,1
00C9  D3 C8                      ROL    AX,1
00CB  D3 C8                      ROL    AX,1
00CD  50                         PUSH   AX
00CE  24 OF                      AND    AL,0FH           ;convert to ASCII
                                 .IF AL > 9
00D4  04 07                         ADD AL,7
                                 .ENDIF
00D6  04 30                      ADD   AL, 3 0H
                                 DISP   AL
00E2  58                         POP    AX
00E3  E2 E0                      LOOP   DREG2            ;repeat 4 times
                                 DISP   ' '
                                 RET

00F1                        DREG  ENDP
                                 END
```

Storing an Interrupt Vector in the Vector Table

In order to install an interrupt vector—sometimes called a **hook**—the assembler must address absolute memory. Example 12-4 shows how a new vector is added to the interrupt vector table by using the assembler and a DOS function call. Here, INT 21H function call number 25H initializes the interrupt vector. Notice that the first thing done in this procedure is to save the old interrupt vector number by using DOS INT 21H function call number 35H to read the current vector. See Appendix A for more detail on DOS INT 21H function calls.

EXAMPLE12-4

```
                            .MODEL TINY
                            .CODE
                            ;A program that installs NEW40 at INT 40H.
                            ;
                            .STARTUP
0100  EB 05                       JMP    START
0102  00000000             OLD   DD     ?
                            ;
                            ;new interrupt procedure
                            ;
0106                        NEW40 PROC   FAR

0106  CF                          IRET

0107                        NEW40 ENDP

0107                  START:
0107  8C C8                      MOV    AX,CS     ;get data segment
0109  8E D8                      MOV    DS,AX
```



```
010B  B4 35                      MOV    AH,35H    ;get old interrupt vector
010D  B0 40                      MOV    AL,40H
010F  CD 21                      INT    21H
0111  89 1E 0102 R               MOV    WORD PTR OLD,BX
0115  8C 06 0104 R               MOV    WORD PTR OLD+2,ES
                            ;
                            ;install new interrupt vector 40H
                            ;
0119  BA 0106 R                  MOV    DX,OFFSET NEW40
011C  B4 25                      MOV    AH,25H
011E  B0 40                      MOV    AL,40H
0120  CD 21                      INT    21H
                            ;
                            ;leave NEW40 in memory
                            ;
0122  BA 0107 R                  MOV    DX,OFFSET START
0125  D1 EA                      SHR    DX,1
0127  D1 EA                      SHR    DX,1
0129  Dl EA                      SHR    DX,1
012B  Dl EA                      SHR    DX,1
012D  42                         INC    DX
012E  B8 3100                    MOV    AX,3100H
0131  CD 21                      INT    21H
                                 END
```

12-2 HARDWARE INTERRUPTS

The microprocessor has two hardware interrupt inputs: non-maskable interrupt (NMI) and interrupt request (INTR). Whenever the NMI input is activated, a type 2 interrupt occurs because NMI is internally decoded. The INTR input must be externally decoded to select a vector. Any interrupt vector can be chosen for the INTR pin, but we usually use an interrupt type number between 20H and FFH. Intel has reserved interrupts OOH through 1FH for internal and future expansion. The $\overline{\text{INTA}}$ signal is also an interrupt pin on the microprocessor, but it is an output that is used in response to the INTR input to apply a vector type number to the data bus connections D7-DO. Figure 12-5 shows the three user interrupt connections on the microprocessor.

The non-maskable interrupt (NMI) is an edge-triggered input that requests an interrupt on the positive edge (0-to-1 transition). After a positive edge, the NMI pin must remain a logic 1 until it is recognized by the microprocessor. Note that before the positive edge is recognized, the NMI pin must be a logic 0 for at least two clocking periods.

The NMI input is often used for parity errors and other major system faults, such as power failures. Power failures are easily detected by monitoring the AC power line and causing an NMI

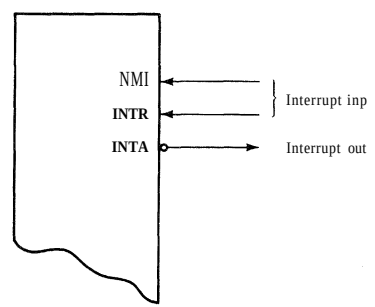


FIGURE 12-5 The interrupt pins on all versions of the Intel microprocessor.

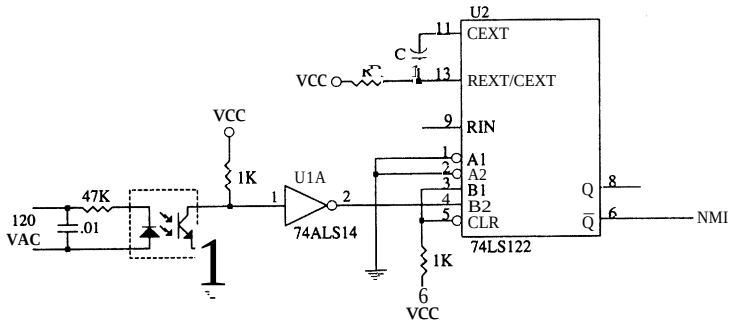


FIGURE 12–6 A power failure detection circuit.

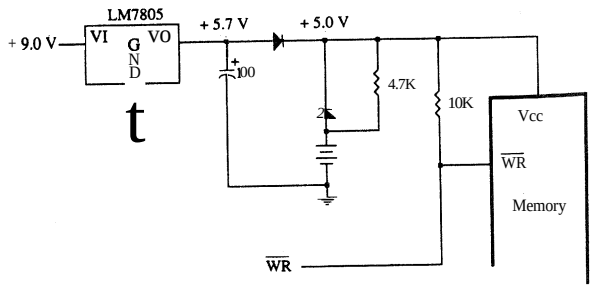
interrupt whenever AC power drops out. In response to this type of interrupt, the microprocessor stores all of the internal register in a battery backed-up-memory or an EEPROM. Figure 12–6 shows a power failure detection circuit that provides a logic 1 to the NMI input whenever AC power is interrupted.

In this circuit, an optical isolator provides isolation from the AC power line. The output of the isolator is shaped by a Schmitt-trigger inverter that provides a 60 Hz pulse to the trigger input of the 74LS122 retriggerable monostable multivibrator. The values of R and C are chosen so that the 74LS122 has an active pulse width of 33 ms or 2 AC input periods. Because the 74LS122 is retriggerable, as long as AC power is applied, the Q output remains triggered at a logic 1 and Q remains a logic 0.

If the AC power fails, the 74LS122 no longer receives trigger pulses from the 74ALS14, which means that Q returns to a logic 0 and Q returns to a logic 1, interrupting the microprocessor through the NMI pin. The interrupt service procedure, not shown here, stores the contents of all internal registers and other data into a battery-backed-up memory. This system assumes that the system power supply has a large enough filter capacitor to provide energy for at least 75 ms after the AC power ceases.

Figure 12-7 shows a circuit that supplies power to a memory after the DC power fails. Here, diodes are used to switch supply voltages from the DC power supply to the battery. The diodes used are standard silicon diodes because the power supply to this memory circuit is elevated above +5.0 V to +5.7 V. The resistor is used to trickle-charge the battery, which is either NiCAD, Lithium, or a gel cell.

FIGURE 12-7 A battery-backed-up memory system using a NiCad, lithium, or gel cell.



When DC power fails, the battery provides a reduced voltage to the Vcc connection on the memory device. Most memory devices will retain data with Vcc voltages as low as 1.5 V, so the battery voltage does not need to be +5.0 V. The WR pin is pulled to Vcc during a power outage, so no data will be written to the memory.

INTR and $\overline{\text{INTA}}$

The interrupt request input (INTR) is level-sensitive, which means that it must be held at a logic 1 level until it is recognized. The INTR pin is set by an external event and cleared inside the interrupt service procedure. This input is automatically disabled once it is accepted by the microprocessor and re-enabled by the IRET instruction at the end of the interrupt service procedure. The 80386–Pentium II use the IRETD instruction in the protected mode of operation.

The microprocessor responds to the INTR input by pulsing the $\overline{\text{INTA}}$ output in anticipation of receiving an interrupt vector type number on data bus connection D7-DO. Figure 12-8 shows the timing diagram for the INTR and INTA pins of the microprocessor. There are two INTA pulses generated by the system that are used to insert the vector type number on the data bus.

Figure 12-9 illustrates a simple circuit that applies interrupt vector type number FFH to the data bus in response to an INTR. Notice that the INTA pin is not connected in this circuit. Because resistors are used to pull the data bus connections (D0–D7) high, the microprocessor automatically sees vector type number FFH in response to the INTR input. This is possibly the least expensive way to implement the INTR pin on the microprocessor.

Using a Three-State Buffer for $\overline{\text{INTA}}$. Figure 12-10 shows how interrupt vector type number 80H is applied to the data bus (DO-D7) in response to an INTR. In response to the INTR, the microprocessor outputs the INTA that is used to enable a 74ALS244 three-state octal buffer. The octal buffer applies the interrupt vector type number to the data bus in response to the INTA pulse. The vector type number is easily changed with the DIP switches that are shown in this illustration.

Making the INTR Input Edge-triggered. Often, we need an edge-triggered input instead of a level-sensitive input. The INTR input can be converted to an edge-triggered input by using a D-type flip-flop, as illustrated in Figure 12–11. Here, the clock input becomes an edge-triggered interrupt request input, and the clear input is used to clear the request when the INTA signal is output by the microprocessor. The RESET signal initially clears the flip-flop so that no interrupt is requested when the system is first powered.

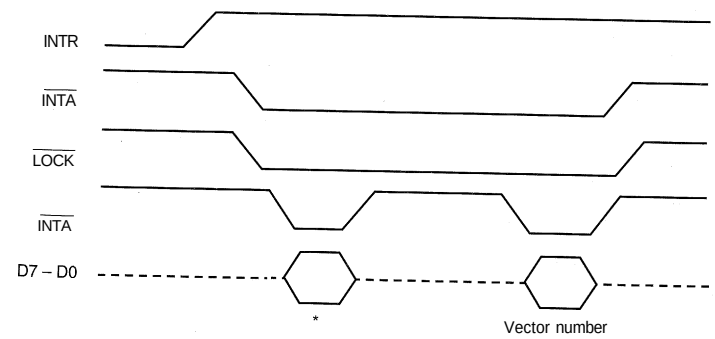


FIGURE 12–8 The timing of the INTR input and INTA output. *Note: This portion of the data bus is ignored and usually contains the vector number.

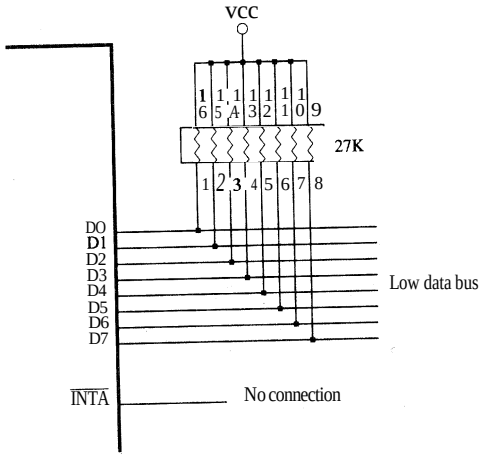


FIGURE 12-9 A simple method for generating interrupt vector type number FFH in response to INTR.

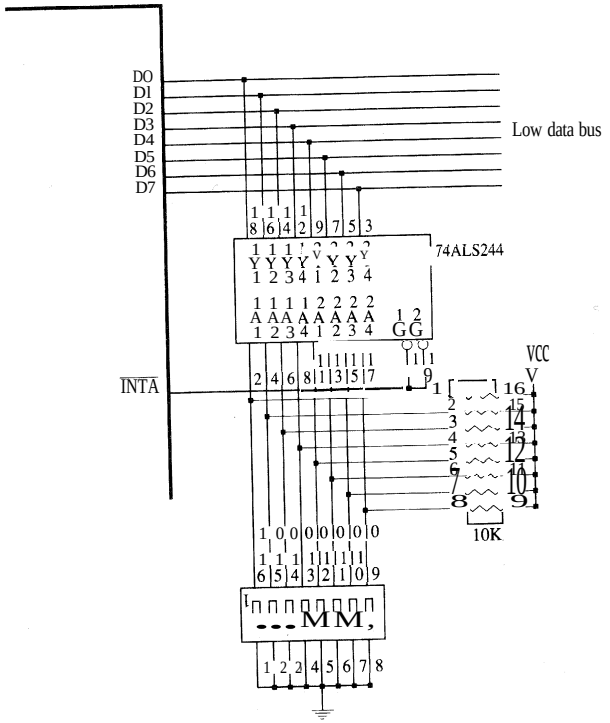


FIGURE 12–10 A circuit that applies any interrupt vector type number in response to INTA. Here the circuit is applying type number 80H.

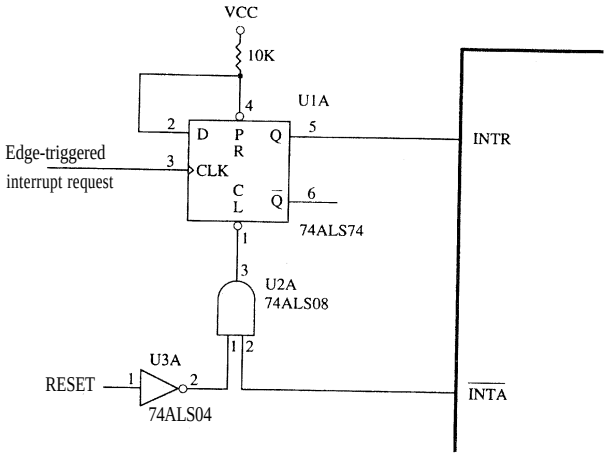


FIGURE 12–11 Converting INTR into an edge-triggered interrupt request input.

The 82C55 Keyboard Interrupt

The keyboard example presented in Chapter 11 provides a simple example of the operation of the INTR input and an interrupt. Figure 12-12 illustrates the interconnection of the 82C55 with the microprocessor and the keyboard. It also shows how a 74ALS244 octal buffer is used to provide the microprocessor with interrupt vector type number 40H in response to the keyboard interrupt during the INTA pulse.

The 82C55 is decoded at 80386SX I/O port address 0500H, 0502H, 0504H, and 0506H by a PAL16L8 (the program is not illustrated). The 82C55 is operated in mode 1 (strobed input mode), so whenever a key is typed, the INTR output (PC3) becomes a logic 1 and requests an interrupt through the INTR pin on the microprocessor. The INTR pin remains high until the ASCII data are read from port A. In other words, every time a key is typed, the 82C55 requests a type 40H interrupt through the INTR pin. The DAV signal from the keyboard causes data to be latched into port A and causes INTR to become a logic 1.

Example 12-5 illustrates the interrupt service procedure for the keyboard. It is very important that all registers affected by an interrupt are saved before they are used. In the software required to initialize the 82C55 (not shown here), the FIFO is initialized so that both pointers are equal, the INTR request pin is enabled through the INTE bit inside the 82C55, and the mode of operation is programmed.

EXAMPLE 12-5

```
                        ;An interrupt service procedure that reads a key
                        ;from the keyboard in Figure 12-12.
                        ;
= 0500                  PORTA EQU   500H
= 0506                  CNTR  EQU   506H

0000  0100 [            FIFO  DB 256 DUP (?)      ;queue
             00
               ]
0100  0000              INP   DW    ?             ;input pointer
0102  0000              OUTP  DW    ?             ;output pointer

0104                    KEY   PROC  FAR USES AX BX DI DX
```

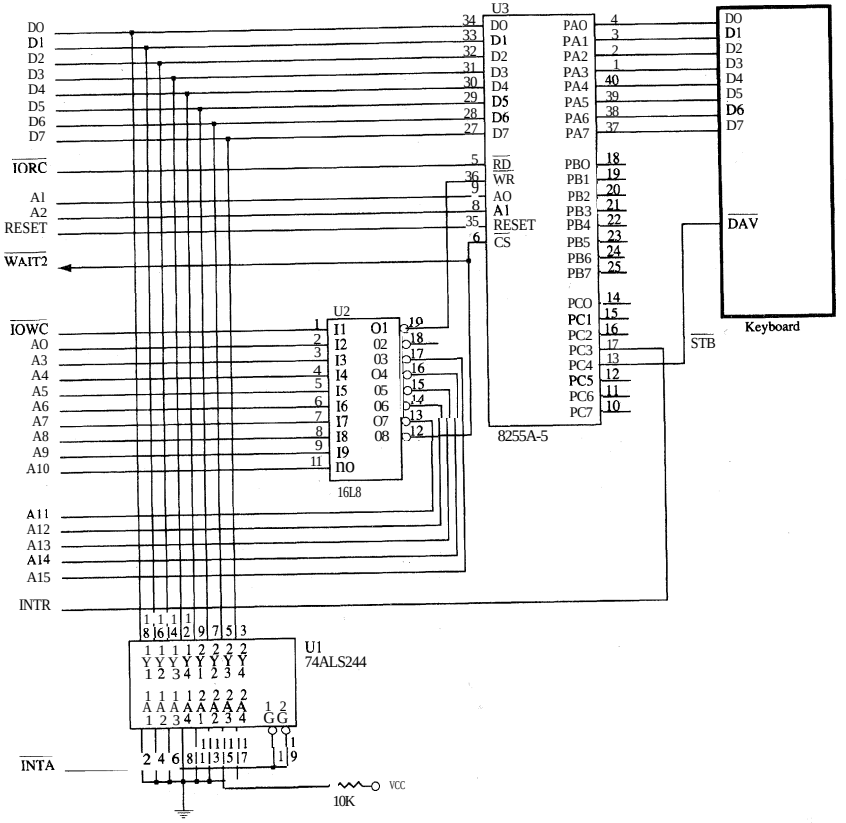


FIGURE 12–12 An 82C55 interfaced to a keyboard from the microprocessor system using interrupt vector 40H.

```
0108  2E: 8B 1E 0100 R        MOV   BX,CS:INP     ;load input pointer
010D  2E: 8B 3E 0102 R        MOV   DI,CS:OUTP    ;load output pointer

0112  FE C3                   INC   BL            ;test for queue = full
0114  3B DF                   CMP   BX,DI
0116  74 11                   JE    FULL          ;if queue is full

0118  FE CB                   DEC   BL
011A  BA 0500                 MOV   DX,PORTA
011D  EC                      IN    AL,DX         ;get data from 82C55
011E  2E: 88 07               MOV   CS:[BX],AL    ; save data in queue
0121  2E: FE 06 0100 R        INC   BYTE PTR INP
0126  EB 07 90                JMP   DONE
0129                    FULL:
0129  B0 08                   MOV   AL,8          ;disable 82C55 interrupt
012B  BA 0506                 MOV   DX,CNTR
012E  EE                      OUT   DX,AL
012F                    DONE:
012F                          IRET

0134                    KEY   ENDP
```


The procedure is short because the 80386SX already knows that keyboard data are available when the procedure is called. Data are input from the keyboard and then stored in the FIFO (first-in, first-out) buffer. Most keyboard interfaces contain a FIFO that is at least 16 bytes in depth. The FIFO in this example is 256 bytes, which is more than adequate for a keyboard interface. Take note at how the INC BYTE PTR INP is used to add one to the input pointer and also make sure that it always addressed data in the queue.

This procedure first checks to see whether the FIFO is full. A full condition is indicated when the input pointer (INP) is one byte below the output pointer (OUTP). If the FIFO is full, the interrupt is disabled with a bit set/reset command to the 82C55, and a return from the interrupt occurs. If the FIFO is not full, the data are input from port A, and the input pointer is incremented before a return occurs.

Example 12-6 shows the procedure that removes data from the FIFO. This procedure first determines whether the FIFO is empty by comparing the two pointers. If the pointers are equal, the FIFO is empty, and the software waits at the EMPTY loop where it continuously tests the pointers. The EMPTY loop is interrupted by the keyboard interrupt, which stores data into the FIFO so that it is no longer empty. This procedure returns with the character in register AH.

EXAMPLE 12-6

```
                              ;A procedure that reads data from the queue of
                              ;Example 12-5 and returns with it in AH.
                              ;
0134                    READ  PROC  FAR USES BX DI DX

0137                    EMPTY:
0137  2E: 8B 1E 0100 R       MOV   BX,CS:INP        ; load input pointer
013D  2E: 8B 3E 0102 R       MOV   DI,CS:OUTP       ; load output pointer
0142  3B DF                  CMP   BX,DI
0144  74 F2                  JE    EMPTY            ;if queue is empty

0146  2E: 8A 25              MOV   AH,CS:[DI]       ;get data
0149  B0 09                  MOV   AL, 9            ;enable 82C55 interrupt
014B  BA 0506                MOV   DX,CNTR
014E  EE                     OUT   DX,AL
014F  2E: FE 06 0102 R       INC   BYTE PTR CS:OUTP
                             RET

0157                    READ  ENDP
```

12–3 EXPANDING THE INTERRUPT STRUCTURE

This text covers three of the more common methods of expanding the interrupt structure of the microprocessor. In this section, we explain how, with software and some hardware modification of the circuit shown in Figure 12-10, it is possible to expand the INTR input so that it accepts seven interrupt inputs. We also explain how to "daisy-chain" interrupts by software polling. In the next section, we describe a third technique, in which up to 63 interrupting inputs can be added by means of the 8259A programmable interrupt controller.

Using the 74ALS244 to Expand

The modification shown in Figure 12-13 allows the circuit of Figure 12-10 to accommodate up to seven additional interrupt inputs. The only hardware change is the addition of an 8-input NAND gate, which provides the INTR signal to the microprocessor when any of the IR inputs becomes active.

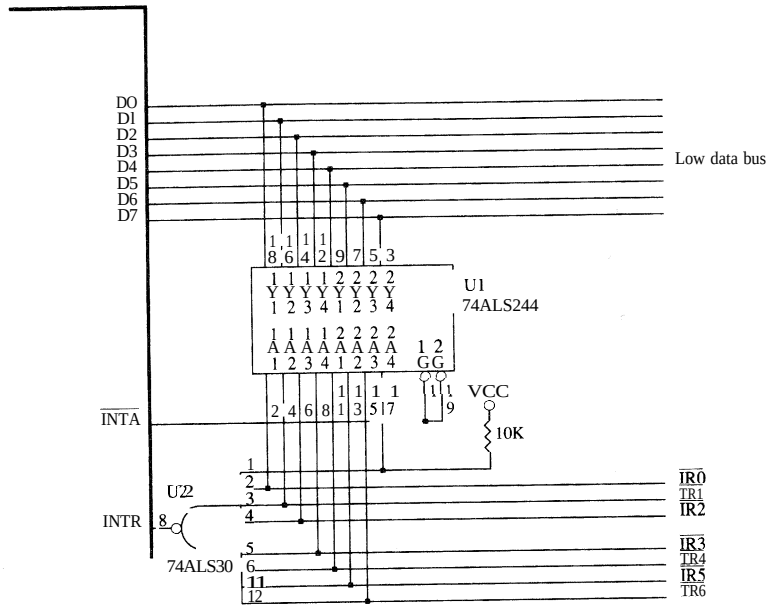


FIGURE 12–13 Expanding the INTR input from one to seven interrupt request lines.

Operation. If any of the IR inputs becomes a logic 0, then the output of the NAND gate goes to a logic 1 and requests an interrupt through the INTR input. The interrupt vector that is fetched during the INTA pulse depends on which interrupt request line becomes active. Table 12-1 shows the interrupt vectors used by a single interrupt request input.

If two or more interrupt request inputs are simultaneously active, a new interrupt vector is generated. For example, if IR1 and IR0 are both active, the interrupt vector generated is FCH (252). Priority is resolved at this location. If the IR0 input is to have the higher priority, the vector address for IR0 is stored at vector location FCH. The entire top half of the vector table and its 128 interrupt vectors must be used to accommodate all possible conditions of these seven interrupt request inputs. This seems wasteful, but in many dedicated applications it is a cost-effective approach to interrupt expansion.

TABLE 12-1 Single interrupt request for Figure 12-13.

$\overline{IR6}$	$\overline{IR5}$	$\overline{IR4}$	$\overline{IR3}$	$\overline{IR2}$	$\overline{IR1}$	$\overline{IR0}$	Vector
1	1	1	1	1	1	0	FEH
1	1	1	1	1	0	1	FDH
1	1	1	1	0	1	1	FBH
1	1	1	0	1	1	1	F7H
1	1	0	1	1	1	1	EFH
1	0	1	1	1	1	1	DFH
0	1	1	1	1	1	1	BFH

Note: Although not illustrated, the IR inputs are all active low.

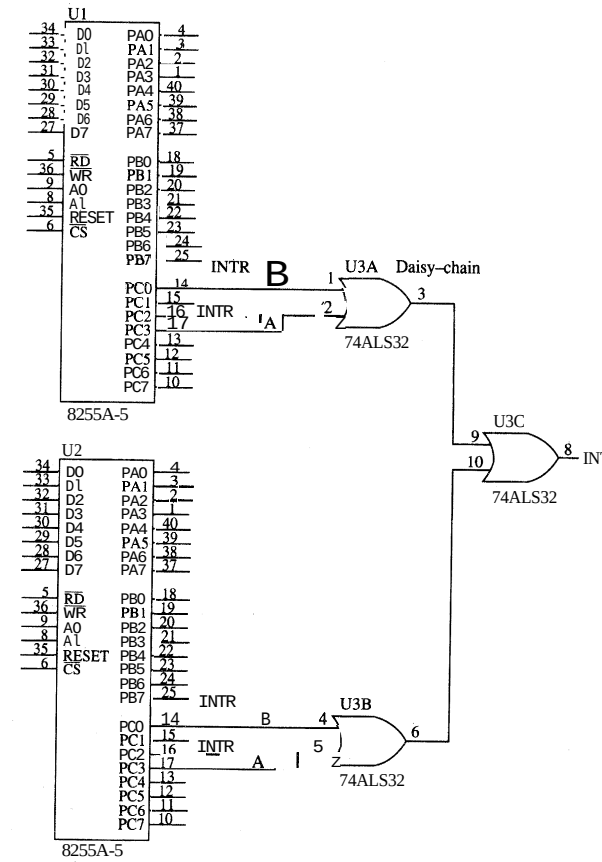
Daisy-Chained Interrupt

Expansion by means of a daisy-chained interrupt is in many ways better than using the 74ALS244 interrupt expansion because it requires only one interrupt vector. The task of determining priority is left to the interrupt service procedure. Setting priority for a daisy-chain does require additional software execution time, but in general this is a much better approach to expanding the interrupt structure of the microprocessor.

Figure 12-14 illustrates a set of two 82C55 peripheral interfaces with their four INTR outputs daisy-chained and connected to the single INTR input of the microprocessor. If any interrupt output becomes a logic 1, so does the INTR input to the microprocessor causing an interrupt.

When a daisy-chain is used to request an interrupt, it is better to pull the data bus connections (DO-D7) high by using pull-up resistors so interrupt vector FFH is used for the chain. Any interrupt vector can be used to respond to a daisy-chain. In the circuit, any of the four INTR outputs from the two 82C55s will cause the INTR pin on the microprocessor to go high, requesting an interrupt.

FIGURE 12–14 Two 82C55 PIAs connected to the INTR outputs are daisy-chained to produce an INTR signal.



When the INTR pin does go high with a daisy-chain, the hardware gives no direct indication as to which 82C55 or which INTR output caused the interrupt. The task of locating which INTR output became active is up to the interrupt service procedure, which must poll the 82C55s to determine which output caused the interrupt.

Example 12-7 illustrates the interrupt service procedure that responds to the daisy-chain interrupt request. The procedure polls each 82C55 and each INTR output to decide which interrupt service procedure to utilize.

EXAMPLE 12-7

```
                        ;A procedure that services the daisy-chain interrupt
                        ;of Figure 12-14.
                        ;
= 0504                  C1      EQU     504H        ;first 82C55
= 0604                  C2      EQU     604H        ;second 82C55
= 0001                  MASK1   EQU     1           ;INTRB
= 0008                  MASK2   EQU     8           ;INTRA

0000                    POLL    PROC    FAR USES AX DX

0002  BA 0504                   MOV     DX,C1       ;address first 82C55
0005  EC                       IN      AL,DX       ;get port C
0006  A8 01                    TEST    AL,MASK1
0008  75 OF                    JNZ     LEVEL_0     ;if INTRB is set
000A  A8 08                    TEST    AL,MASK2
000C  75 13                    JNZ     LEVEL_1     ;if INTRA is set

000E  BA 0604                  MOV     DX,C2       ;address second 82C55
0011  EC                       IN      AL,DX       ;get port C
0012  A8 01                    TEST    AL,MASK1
0014  75 1B                    JNZ     LEVEL_2     ;if INTRB is set
0016  EB 29 00                 JMP     LEVEL_3     ;for INTRA

0019                    POLL    ENDP
```

12–4 8259A PROGRAMMABLE INTERRUPT CONTROLLER

The 8259A programmable interrupt controller (PIC) adds eight vectored priority encoded interrupts to the microprocessor. This controller can be expanded, without additional hardware, to accept up to 64 interrupt requests. This expansion requires a master 8259A and eight 8259A slaves.

General Description of the 8259A

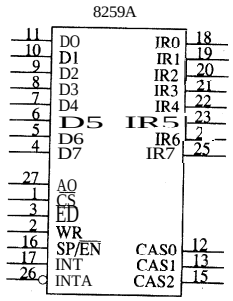
Figure 12-15 shows the pin-out of the 8259A. The 8259A is easy to connect to the microprocessor because all of its pins are direct connections except the CS pin, which must be decoded, and the WR pin, which must have an I/O bank write pulse. Following is a description of each pin on the 8259A:

D7-DO **The bi-directional data connections** are normally connected to either the upper or lower data bus on the 80386SX microprocessor or the data bus on the 8088. If an 80486 or Pentium-Pentium II is used, then they connect to any 8-bit bank.

IR7-IRO **Interrupt request** inputs are used to request an interrupt and to connect to a slave in a system with multiple 8259As.

WR **The write** input connects to either the lower or upper write strobe signal in a 16-bit system, or to any other bus write strobe in any size system.

FIGURE 12-15 The pin-out of the 8259A programmable interrupt controller (PIC).



$\overline{RD}$ **The read** input connects to the IORC signal.

INT The **interrupt** output connects to the INTR pin on the microprocessor from the master, and is connected to a master IR pin on a slave.

$\overline{IOTA}$ **Interrupt acknowledge** is an input that connects to the INTA signal on the system. In a system with a master and slaves, only the master INTA signal is connected.

AO The AO address input selects different command words within the 8259A.

$\overline{CS}$ **Chip select** enables the 8259A for programming and control.

SP/$\overline{EN}$ **Slave program/enable buffer** is a dual-function pin. When the 8259A is in buffered mode, this is an output that controls the data bus transceivers in a large microprocessor-based system. When the 8259A is not in the buffered mode, this pin programs the device as a master (1) or a slave (0).

CAS2-CASO The **cascade** lines are used as outputs from the master to the slaves for cascading multiple 8259As in a system.

Connecting a Single 8259A

Figure 12-16 shows a single 8259A connected to the 8086 microprocessor. Here the SP/EN pin is pulled high to indicate that it is a master. The 8259A is decoded at I/O ports 0400H and 0402H by the PAL16L8 (no program shown). Like other peripherals discussed in Chapter 11, the 8259A requires four wait states for it to function properly with a 16 MHz 80386SX and more for some other versions of the Intel microprocessor family.

Cascading Multiple 8259As

Figure 12-17 shows two 8259As connected to the 80386SX microprocessor in a way that is often found in the AT-style computer, which has two 8259As for interrupts. The XT- or PC-style computer uses an 8259A controller at interrupt vectors 08H-OFH. The AT-style computer uses interrupt vector OAH as a cascade input from a second 8259A located at vectors 70H through 77H. Appendix A contains a table that lists the functions of all the interrupt vectors used in the PC-, XT-, and AT-style computers.

This circuit uses vectors 08H-OFH and I/O ports 0300H and 0302H for U1, the master; and vectors 70H-77H and I/O ports 0304H and 0306H for U2, the slave. Notice that we also include data bus buffers to illustrate the use of the SP/EN pin on the 8259A. These buffers are used only in very large systems that have many devices connected to their data bus connections. In practice, we seldom find these buffers.

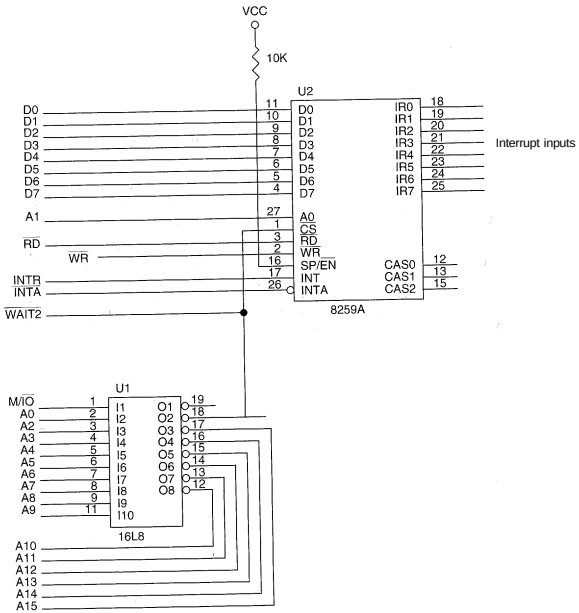


FIGURE 12–16 An 8259A interfaced to the 8086 microprocessor.

Programming the 8259A

The 8259A is programmed by initialization and operation command words. **Initialization command words** (ICWs) are programmed before the 8259A is able to function in the system and dictate the basic operation of the 8259A. **Operation command words** (OCWs) are programmed during the normal course of operation. The OCWs control the operation of the 8259A.

Initialization Command Words. There are four initialization command words (ICWs) for the 8259A that are selected when the AO pin is a logic one. When the 8259A is first powered up, it must be sent ICW1, ICW2, and ICW4. If the 8259A is programmed in cascade mode by ICW1, then we also must program ICW3. So if a single 8259A is used in a system, ICW1, ICW2, and ICW4 must be programmed. If cascade mode is used in a system, then all four ICWs must be programmed. Refer to Figure 12-18 for the format of all four ICWs. The following is a description of each ICW:

ICW1 Programs the basic operation of the 8259A. To program this ICW for 8086–Pentium II operation, we place a logic 1 in bit IC4. Bits ADI, A7, A6, and A5 are don't cares for microprocessor operation and only apply to the 8259A when used with an 8-bit 8085 microprocessor (not covered in this textbook). This ICW selects single or cascade operation by programming the SNGL bit. If cascade operation is selected, we must also program ICW3. The LTIM bit determines whether the interrupt request inputs are positive edge-triggered or level-triggered.

ICW2 Selects the vector number used with the interrupt request inputs. For example, if we decide to program the 8259A so it functions at vector locations 08H-OFH, we place a 08H into this command word. Likewise, if we decide to program the 8259A for vectors 70H-77H, we place a 70H in this ICW.

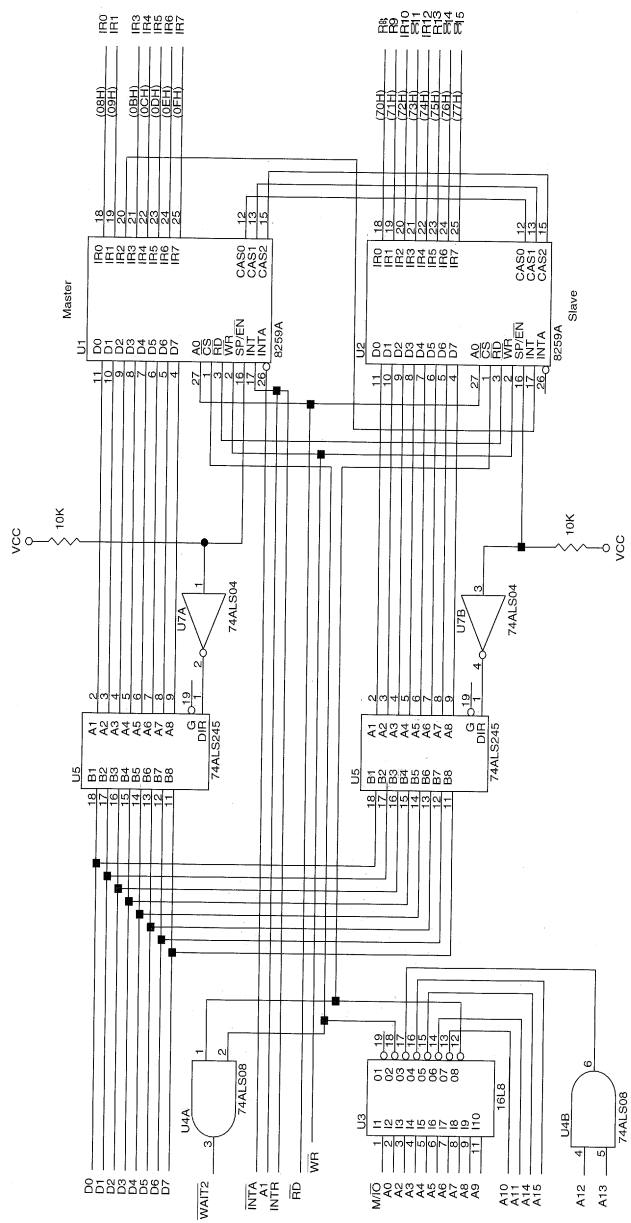
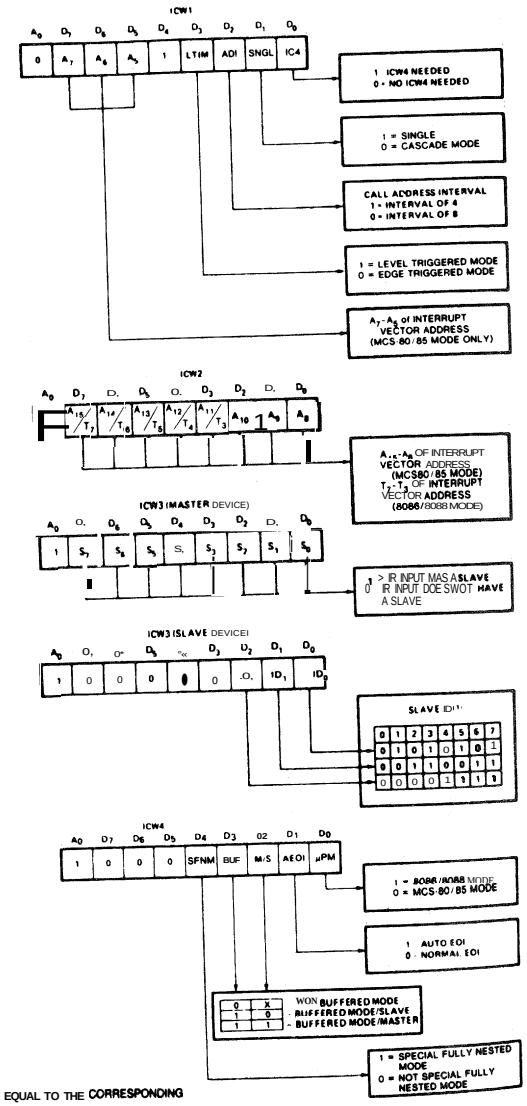


FIGURE 12–17 Two 8259As interfaced to the 8259A at I/O ports 0300H and 0302H for the master and 0304H and 0306H for the slave.



NOTE 1: SLAVE ID IS EQUAL TO THE CORRESPONDING MASTER IR INPUT.

FIGURE 12-18 The 8259A initialization command words (ICWs) (Courtesy of Intel Corporation.)

ICW3 Is only used when ICW1 indicates that the system is operated in cascade mode. This ICW indicates where the slave is connected to the master. For example, in Figure 12–18 we connected a slave to IR2. To program ICW3 for this connection, in both master and slave, we place a 04H in ICW3. Suppose we have two slaves

connected to a master using IR0 and IR1. The master is programmed with an ICW3 of 03H; one slave is programmed with an ICW3 of 01H and the other with an ICW3 of 02H.

ICW4 Is programmed for use with the 8086–Pentium II microprocessors, but is not programmed in a system that functions with the 8085 microprocessor. The rightmost bit must be a logic 1 to select operation with the 8086-Pentium II microprocessors, and the remaining bits are programmed as follows:

SFNM—Selects the special fully-nested mode of operation for the 8259A if a logic 1 is placed in this bit. This allows the highest-priority interrupt request from a slave to be recognized by the master while it is processing another interrupt from a slave. Normally, only one interrupt request is processed at a time and others are ignored until the process is complete.

BUF and M/S—Buffer and master slave are used together to select buffered operation or nonbuffered operation for the 8259A as a master or a slave.

AEOI—Selects automatic or normal end of interrupt (discussed more fully under operation command words). The EOI commands of OCW2 are used only if the AEOI mode is not selected by ICW4. If AEOI is selected, the interrupt automatically resets the interrupt request bit and does not modify priority. This is the preferred mode of operation for the 8259A and reduces the length of the interrupt service procedure.

Operation Command Words. The operation command words (OCWs) are used to direct the operation of the 8259A once it is programmed with the ICW. The OCWs are selected when the AO pin is at a logic 0 level, except for OCW1, which is selected when AO is a logic 1. Figure 12-19 lists the binary bit patterns for all three operation command words of the 8259A. Following is a list describing the function of each OCW:

OCW1 Is used to set and read the interrupt mask register. When a mask bit is set, it will turn off (mask) the corresponding interrupt input. The mask register is read when OCW1 is read. Because the state of the mask bits are unknown when the 8259A is first initialized, OCW1 must be programmed after programming the ICW upon initialization.

OCW2 Is programmed only when the AEOI mode is not selected for the 8259A. In this case, this OCW selects the way that the 8259A responds to an interrupt. The modes are listed as follows:

Nonspecific End-of-Interrupt—A command sent by the interrupt service procedure to signal the end of the interrupt. The 8259A automatically determines which interrupt level was active and resets the correct bit of the interrupt status register. Resetting the status bit allows the interrupt to take action again or a lower priority interrupt to take effect.

Specific End-of-Interrupt—A command that allows a specific interrupt request to be reset. The exact position is determined with bits L2-LO of OCW2.

Rotate-on-Nonspecific EOI—A command that functions exactly like the Non-specific End-of-Interrupt command, except that it rotates interrupt priorities after resetting the interrupt status register bit. The level reset by this command becomes the lowest-priority interrupt. For example, if IR4 was just serviced by this command, it becomes the lowest-priority interrupt input and IR5 becomes the highest priority.

Rotate-on-Automatic EOI—A command that selects automatic EOI with rotating priority. This command must only be sent to the 8259A once if this mode is desired. If this mode must be turned off, use the clear command.

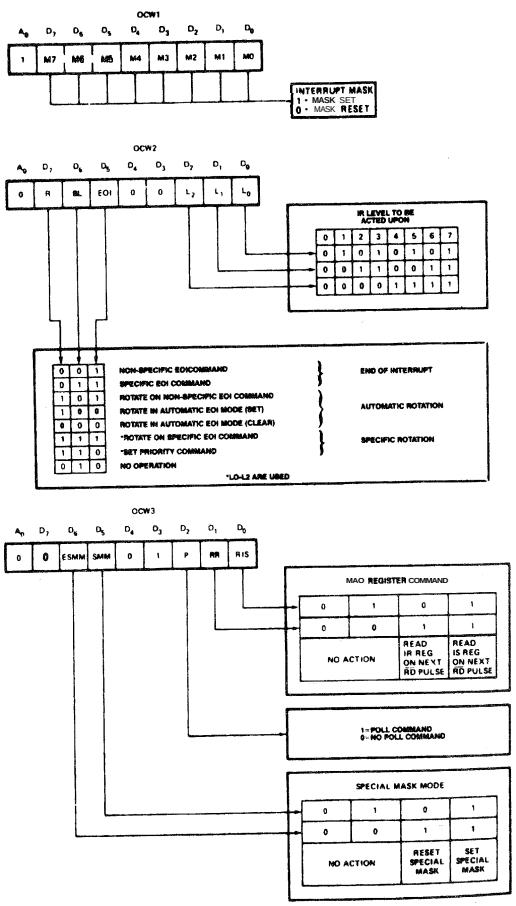


FIGURE 12-19 The 8259A operation command words (OCWs). (Courtesy of Intel Corporation.)

Rotate-on-Specific EOI—Functions as the specific EOI, except that it selects rotating priority.

Set priority—Allows the programmer to set the lowest priority interrupt input using the L2-LO bits.

OCW3 Selects the register to be read, the operation of the special mask register, and the poll command. If polling is selected, the P bit must be set and then output to the 8259A. The next read operation will read the poll word. The rightmost three bits of the poll word indicate the active interrupt request with the highest priority. The leftmost bit indicates whether there is an interrupt and must be checked to determine whether the rightmost three bits contain valid information.

Status Register. Three status registers are readable in the 8259A: interrupt request register (IRR), in-service register (ISR), and interrupt mask register (IMR). (See Figure 12-20 for all three

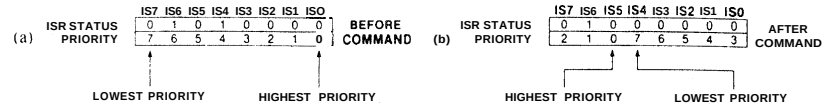


FIGURE 12-20 The 8259A in-service register (ISR). (a) Before IR4 is accepted, and (b) after IR4 is accepted. (Courtesy of Intel Corporation.)

status registers; they all have the same bit configuration.) The IRR is an 8-bit register that indicates which interrupt request inputs are active. The ISR is an 8-bit register that contains the level of the interrupt being serviced. The IMR is an 8-bit register that holds the interrupt mask bits and indicates which interrupts are masked off.

Both the IRR and ISR are read by programming OCW3 and IMR is read through OCW1. To read the IMR, AO = 1; to read either IRR or ISR, AO = 0. Bit positions DO and D1 of OCW3 select which register (IRR or ISR) is read when AO = 0.

8259A Programming Example

Figure 12-21 illustrates the 8259A programmable interrupt controller connected to a 16550 programmable communications controller. In this circuit, the INTR pin from the 16550 is connected to the programmable interrupt controller's interrupt request input IR0. An IR0 occurs whenever (1) the transmitter is ready to send another character, (2) the receiver has received a character, (3) an error is detected while receiving data, and (4) a modem interrupt occurs. Notice that the 16550 is decoded at I/O ports 40H and 47H, and the 8259A is decoded at 8-bit I/O ports 48H and 49H. Both devices are interfaced to data bus of an 8088 microprocessor.

Initialization Software. The first portion of the software for this system must program both the 16550 and the 8259A, and then enable the INTR pin on the 8088 so that interrupts can take effect. Example 12-8 lists the software required to program both devices and enable INTR. This software uses two memory FIFOs that hold data for the transmitter and for the receiver. Each memory FIFO is 16K bytes long and is addressed by a pair of pointers (input and output).

EXAMPLE 12-8

```
                        ;Initialization software for the 16550 and 8259A
                        ;of the circuit in Figure 12-21.
                        ;
= 0048                  PIC1   EQU   48H        ;8259A control AO = 0
= 0049                  PIC1   EQU   49H        ;8259A control AO = 1
= 001B                  ICW1   EQU   1bH        ;8259A ICW1
= 0080                  ICW2   EQU   80H        ;8259A ICW2
= 0003                  ICW4   EQU   3          ;8259A ICW4
= 00FE                  OCW1   EQU   OFEH       ;8259A OCW1
= 0043                  LINE   EQU   43H        ;16550 line register
= 0040                  LSB    EQU   40H        ;16550 Baud divisor LSB
= 0041                  MSB    EQU   41H        ;16550 Baud divisor MSB
= 0042                  FIFO   EQU   42H        ;16550 FIFO register
= 0041                  ITR    EQU   41H        ;16550 interrupt register

0000                    START  PROC  NEAR
                        ;
                        ;Program 16550, but do not enable interrupts yet
                        ;
0000  BO 8A                    MOV   AL,10001010B   ;enable Baud divisor
0002  E6 43                   OUT   LINE,AL

0004  BO 78                    MOV   AL,120         ;program Baud rate
0006  E6 40                    OUT   LSB,AL         ;9600 Baud rate
0008  BO 00                    MOV   AL,0
```

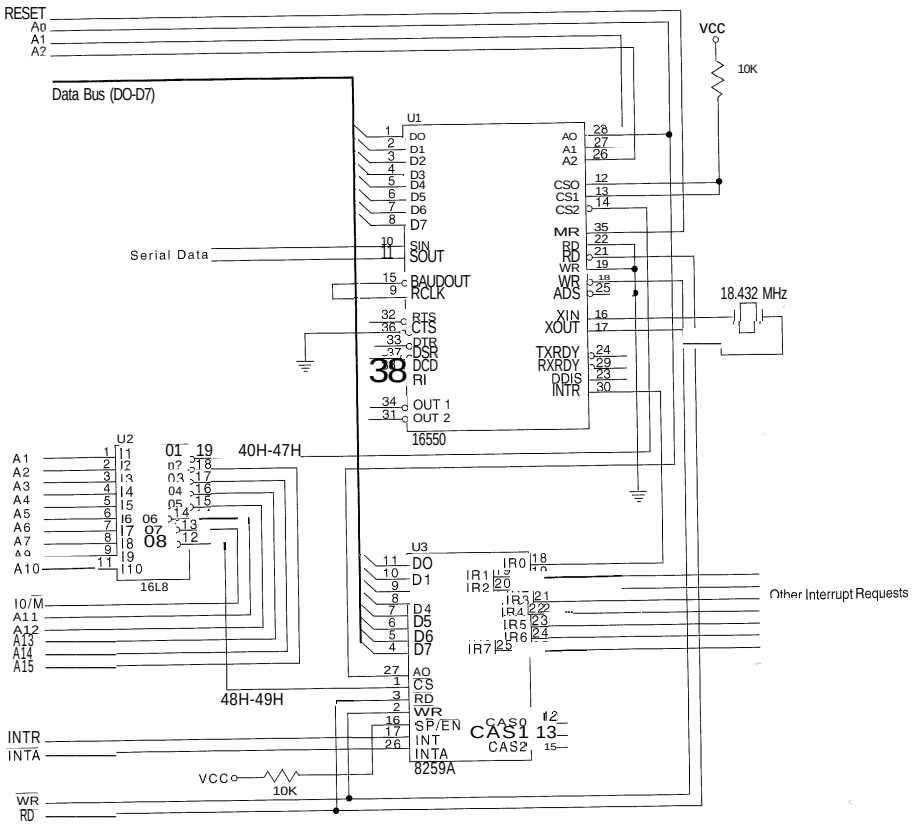


FIGURE 12-21 The 16550 UART interfaced to the 8088 microprocessor through the 8259A.

```
000A  E6 41              OUT   MSB,AL

000C  B0 0A              MOV   AL,00001010B   ;program 7-data, odd
000E  E6 43              OUT   LINE,AL        ;parity, one stop
0010  B0 07              MOV   AL,00000111B   ;enable transmitter and
0012  E6 42              OUT   FIFO,AL        ;and receiver
                    ;
                    ;Program 8259A
                    ;
0014  B0 1B              MOV   AL,ICW1        ;program ICW1
0016  E6 48              OUT   PIC1,AL

0018  B0 80              MOV   AL,ICW2        ;program ICW2
001A  E6 49              OUT   PIC2,AL

001C  B0 03              MOV   AL,ICW4        ;program ICW4
001E  E6 49              OUT   PIC2,AL
```



```
0020  B0 FE              MOV   AL,OCW1        ;program OCW1
0022  E6 49              OUT   PIC2,AL
0024  FB                 STI                  ;enable system INTR pin
                    ;
                    ;enable 16550 interrupts
                    ;
0025  B0 07              MOV   AL,5           ;enable receiver and
0027  E6 41              OUT   ITR,AL         ;error interrupts
0029  C3                 RET

002A                START ENDP
```

The first portion of the procedure (START) programs the 16550 UART for operation with seven data bits, odd parity, one stop bit, and a Baud rate clock of 9600. The FIFO control register also enables both the transmitter and receiver.

The second part of the procedure programs the 8259A, with its three ICWs and its one OCW. The 8259A is set up so that it functions at interrupt vectors 80H-87H and operates with automatic EOI. The ICW enables the interrupt for the 16550 UART. The INTR pin of the microprocessor is also enabled by using the STI instruction.

The final part of the software enables the receiver and error interrupts of the 16550 UART through the interrupt control register. The transmitter interrupt is not enabled until data are available for transmission. See Figure 12-22 for the contents of the interrupt control register of the 16550 UART. Notice that the control register can enable or disable the receiver, transmitter, line status (error), and modem interrupts.

Handling the 16550 UART Interrupt Request. Because the 16550 generates only one interrupt request for various interrupts, the interrupt handler must poll the 16550 to determine what type of interrupt has occurred. This is accomplished by examining the interrupt identification register (see Figure 12-23). Note that the interrupt identification register (read-only) shares the same I/O port as the FIFO control register (write-only).

FIGURE 12–22 The 16550 interrupt control register.

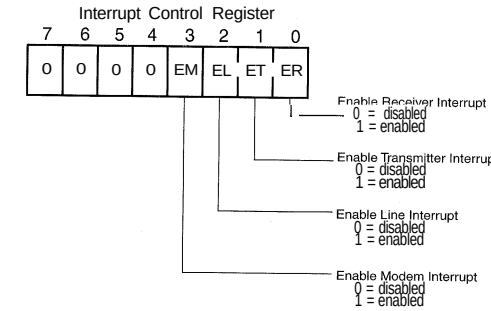


FIGURE 12-23 The 16550 interrupt identification register.

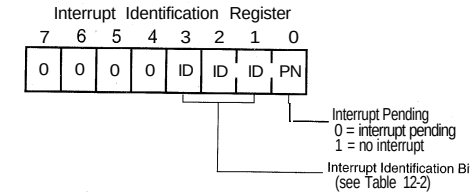


TABLE 12–2 The interrupt control bits of the 16550.

Bit3	Bit2	Bit1	BitO	Priority	Type	Reset Control
0	0	0	1	—	No interrupt	—
0	1	1	0	1	Receiver error (parity, framing, overrun, or break)	Reset by reading the line register
0	1	0	0	2	Receiver data available	Reset by reading the data
1	1	0	0	2	Character time-out, nothing has been removed from the receiver FIFO for at least four character times	Reset by reading the data
0	0	1	0	3	Transmitter empty	Reset by writing to the transmitter
0	0	0	0	4	Modem status	Reset by reading the modem status

Note: 1 is the highest priority and 4 the lowest.

The interrupt identification register indicates whether an interrupt is pending, the type of interrupt, and whether the transmitter and receiver FIFO memories are enabled. See Table 12–2 for the contents of the interrupt control bits.

The interrupt service procedure must examine the contents of the interrupt identification register to determine what event caused the interrupt and pass control to the appropriate procedure for the event. Example 12-9 shows the first part of an interrupt handler that passes control to RECV for a receiver data interrupt, TRANS for a transmitter data interrupt, and ERR for a line status error interrupt. Note that the modem status is not tested in this example.

EXAMPLE 12-9

```
                        ;Interrupt handler for the 16550 UART of
                        ;Figure 12-21.
                        ;
0000                    INT80 PROC  FAR

0000  50                      PUSH  AX
0001  E4 42                  IN    AL,42H      ;input interrupt ID reg
0003  3C 06                  CMP   AL,6        ;test for error
0005  74 20                  JE    ERR         ;for receiver error

0007  3C 02                  CMP   AL,2        ;test for transmitter
0009  74 55                  JE    TRANS       ; for transmitter ready

000B  3C 04                  CMP   AL,4        ;test for receiver
000D  74 11                  JE    RECV        ;for receiver ready
```

Receiving Data from the 16550. The data received by the 16550 are stored, not only in the FIFO within the UART, but also in a FIFO memory until the software in the main program can use them. The FIFO memory used for received data is 16K bytes long, so many characters can easily be stored and received before any intervention from the microprocessor is required to empty the receiver's memory FIFO. The receiver memory FIFO is stored in the extra segment so string instructions that use the DI register can be used to access it.

Receiving data from the 16550 requires two procedures. One procedure reads the data register of the 16550 each time that the INTR pin requests an interrupt, and stores it into the memory FIFO. The other procedure reads data from the memory FIFO from the main program.

Example 12-10 lists the procedure used to read data from the memory FIFO from the main program. This procedure assumes that the pointers (IIN and IOUT) are initialized in the initialization dialog for the system (not shown). The READ procedure returns with AL containing a character read from the memory FIFO. If the memory FIFO is empty, the procedure returns with the carry flag bit set to a logic one. If AL contains a valid character, the carry flag bit is cleared upon return from READ.

Notice how the FIFO is reused by changing the address from the top of the FIFO to the bottom whenever it exceeds the start of the FIFO plus 16K. This is located at the CMP instruction at offset address 0015. Also notice that interrupts are enabled at the end of this procedure, in case they are disabled by a full memory FIFO condition by the RECV interrupt procedure.

EXAMPLE 12–10

```
                        ;A procedure that reads one character from the memory
                        ;FIFO and returns with it in AL.
                        ;If the FIFO is empty the return occurs with Carry = 1.
                        ;
0000                    READ  PROC  NEAR USES BX DI

0002  26 : 8B 3E 4002 R      MOV   DI,IOUT          ;get output pointer
0007  26 : 8B 1E 4000 R      MOV   BX, UN           ; get input pointer

000C  3B DF                  CMP   BX,DI            ; compare pointers
000E  F9                     STC                    ;set carry flag
000F  74 16                  JE    DONE1            ;if empty

0011  26 : BA 06             MOV   AL,ES: [DI]      ;get data from FIFO
0014  47                     INC   DI               ;address next byte
0015  81 FF 4000 R           CMP   DI,OFFSET FIFO+16*1024
0019  26 : 89 3E 4002 R      MOV   IOUT,DI          ;save pointer
001E  76 07                  JBE   DONE             ;if within bounds
0020  26 : C7 06 4002 R      MOV   IOUT,OFFSET FIFO
      0000  R
0027                    DONE:
0027  F8                     CLC                    ;clear carry flag
0028                    DONE1:
0028  9C                     PUSHF                  ;save carry flag
0029  E4 41                  IN    AL,41H           ;read interrupt control
002B  06 05                  OR    AL,5             ;enable receiver interrupts
002D  E6 41                  OUT   41H,AL
002F  9D                     POPF
                             RET

0033                    READ  ENDP
```

Example 12–11 lists the RECV interrupt service procedure that is called each time the 16550 receives a character for the microprocessor. In this example, the interrupt uses vector type number 80H, which must address the interrupt handler of Example 12-9. Each time that this interrupt occurs, the REVC procedure is accessed by the interrupt handler reading a character from the 16550. The RECV procedure stores the character into the memory FIFO. If the memory FIFO is full, the receiver interrupt is disabled by the interrupt control register within the 16550. This may result in lost data, but at least it will not cause the interrupt to overrun valid data already stored in the memory FIFO. Any error conditions detected by the 8251A store a ? (3FH) in the memory FIFO. Note that errors are detected by the ERR portion of the interrupt handler (not shown).

EXAMPLE 12–11

```
                        ;RECV portion of the interrupt handler in Example
                        ;12-9.
                        ;
0020                    RECV:                  ;continues from Example 12-9
0020  53                      PUSH  BX         ;save registers
```

```
0021  57                      PUSH  DI
0022  56                      PUSH  SI
0023  26: 8B 1E 4002 R       MOV   BX,IOUT      ;load output pointer
0028  26: 8B 36 4000 R       MOV   SI, UN       ; load input pointer
002D  8B FE                  MOV   DI,SI
002F  46                     INC   SI
0030  81 FE 4000 R           CMP   SI,OFFSET FIFO+16*1024
0034  76 03                  JBE   NEXT
0036  BE 0000 R              MOV   SI,OFFSET FIFO
0039                  NEXT:
0039  3B DE                  CMP   BX,SI        ;is FIFO full?
003B  74 0B                  JE    FULL         ;if it is full
003D  E4 40                  IN    AL,40H       ;read 16550 receiver
003F  AA                     STOSB              ;save it in FIFO
0040  26: 89 36 4000 R       MOV   UN, SI       ;save input pointer
0045  EB 06 90               JMP   DONE         ;end up
0048                  FULL
0048  E4 41                  IN    AL,41H       ;read interrupt control
004A  24 FA                  AND   AL,0FAH      ;disable receiver
004C  E6 41                  OUT   41H,AL
004E                  DONE:
004E  B0 20                  MOV   AL,20H       ;signal 8259A EOI
0050  E6 49                  OUT   49H,AL
0052  5E                    POP   SI           ;restore registers
0053  5F                    POP   DI
0054  5B                    POP   BX
0055  58                    POP   AX
0056  CF                    IRET
```

Transmitting Data to the 16550. Data are transmitted to the 16550 in much the same manner as they are received, except that the interrupt service procedure removes transmit data from a second 16K-byte memory FIFO.

Example 12-12 lists the procedure that fills the output FIFO. It is similar to the procedure listed in Example 12-10, except it determines whether the FIFO is full instead of empty.

EXAMPLE 12–12

```
                      ;A procedure that places data into the memory FIFO for
                      ;transmission by the transmitter interrupt.
                      ;AL = character to be transmitted.
                      ;
0000                  SAVE  PROC  NEAR USES BX DI SI

0003  26:: 8B 36 8004 R      MOV   SI,OIN   ;get input pointer
0008  26:: 8B 1E 8006 R      MOV   BX,OOUT  ;get output pointer
000D  8B FE                  MOV   DI,SI
000F  46                     INC   SI
0010  81 FE 8004 R           CMP   SI,OFFSET OFIFO+16*1024
0014  76 03                  JBE   NEXT
0016  BE 4004 R              MOV   SI,OFFSET OFIFO
0019                  NEXT:
0019  3B DE                  CMP   BX,SI
001B  74 06                  JE    DONE     ;if full
001D  AA                     STOSB          ;save data in OFIFO
001E  26:: 89 36 8004 R      MOV   OIN,SI
0023                  DONE:
0023  E4 41                  IN    AL,41H   ;read interrupt control
0025  06 01                  OR    AL,1     ;enable transmitter
0027  E6 41                  OUT   41H,AL
                      RET
002D                  SAVE  ENDP
```

Example 12-13 lists the interrupt service subroutine for the 16550 UART transmitter. This procedure is a continuation of the interrupt handler presented in Example 12–9 and is similar to the RECV procedure of Example 12–11, except that it determines whether the FIFO is empty rather than full. Note that we do not include an interrupt service procedure for the break interrupt or any errors.

EXAMPLE 12–13

```
                      ;Interrupt service procedure for the 16550
                      ;transmitter.
                      ;
0060                  TRANS:
0060  53                    PUSH  BX           ;save registers
0061  57                    PUSH  DI
0062  26 : 8B 1E 8004 R     MOV   BX,OIN       ;load input pointer
0068  26 : 8B 3E 8006 R     MOV   DI,OOUT      ;load output pointer
006D  3B DF                  CMP   BX,DI
006F  74 17                  JE    EMPTY        ;if empty
0071  26 : 8A 05             MOV   AL,ES: [DI]  ;get character
0074  E6 40                  OUT   40H,AL       ;send it to UART
0076  47                     INC   DI
0077  81 FF 8004 R           CMP   DI,OFFSET OFIFO+16*1024
007B  76 03                  JBE   NEXT1
007D  BF 4004 R              MOV   DI,OFFSET OFIFO
0080                  NEXT1:
0080  26: 89 3E 8006 R       MOV   OOUT,DI
0085  EB 07 90               JMP   DONES
0088                  EMPTY:
0088  E4 41                  IN    AL,41H       ;read interrupt control
008A  24 FD                  AND   AL,0FDH      ;disable transmitter
008C  E6 41                  OUT   41H,AL
008E                  DONES:
008E  B0 20                  MOV   AL,20H       ;signal 8259A EOI
0090  E6 49                  OUT   49H,AL
0092  5F                    POP   DI
0093  5B                    POP   BX
0094  58                    POP   AX
0095  CF                    IRET
```

The 16550 also contains a scratch register, which is a general-purpose register that can be used in any way deemed necessary by the programmer. Also contained within the 16550 are a modem control register and a modem status register. These registers allow the modem to cause interrupt and control the operation of the 16550 with a modem. See Figure 12-24 for the contents of both the modem status register and the modem control register.

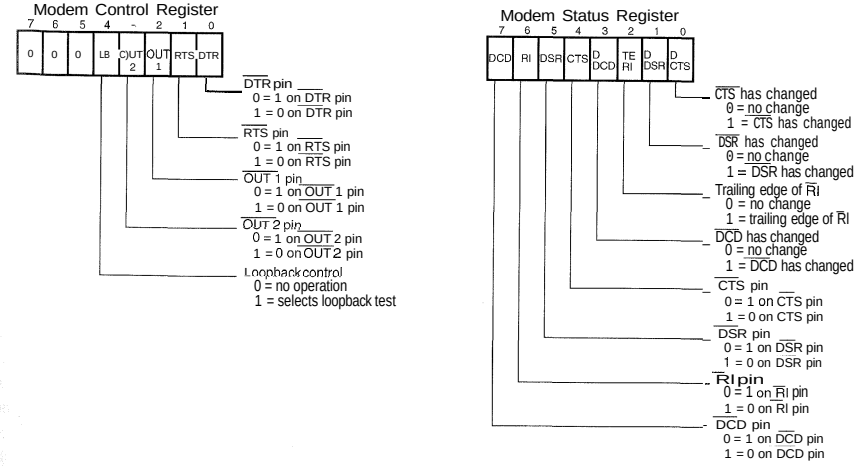


FIGURE 12–24 The 16550 modem control and modem status registers.

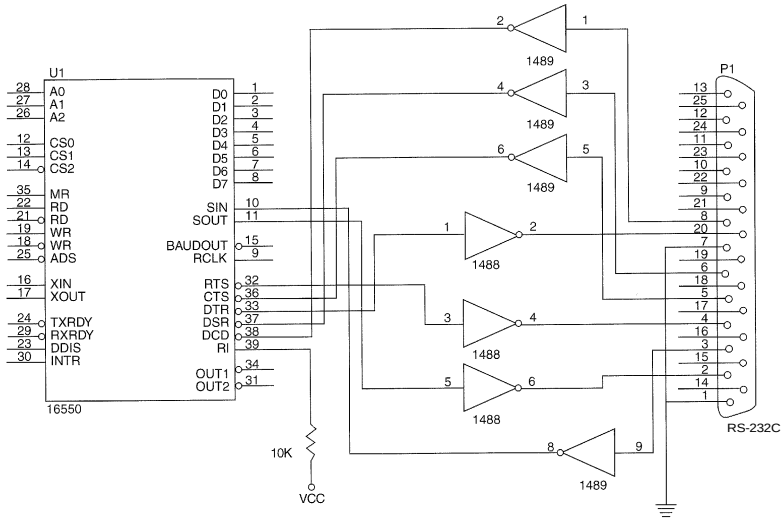


FIGURE 12-25 The 16550 interfaced to RS-2332C using 1488 line drivers and 1489 line receivers.

The modem control register uses bit positions 0-3 to control various pins on the 16550. Bit position 4 enables the internal loop-back test for testing purposes. The modem status register allows the status of the modem pins to be tested; it also allows the modem pins to be checked for a change or, in the case of $\overline{RI}$, a trailing edge.

Figure 12-25 illustrates the 16550 UART, connected to an RS-232C interface that is often used to control a modem. Included in this interface are line driver and receiver circuits used to convert between TTL levels on the 16550 to RS-232C levels found on the interface. Note that RS-232C levels are usually +12 V for a logic 0 and -12 V for a logic 1 level.

In order to transmit or receive data through the modem, the DTR pin is activated (logic 0) and the UART then waits for the DSR pin to become a logic 0 from the modem, indicating that the modem is ready. Once this handshake is complete, the UART sends the modem a logic 0 on the RTS pin. When the modem is ready, it returns the CTS signal (logic 0) to the UART. Communications can now commence. The DCD signal from the modem is an indication that the modem has detected a carrier. This signal must also be tested before communications can begin.

12–5 INTERRUPT EXAMPLES

This section of the text presents a real-time clock and an interrupt processed keyboard as examples of interrupt applications. A **real-time clock** keeps time in real time—that is, in hours and minutes. The example illustrated here keeps time in hours, minutes, seconds, and 1/60 seconds, using four memory locations to hold the BCD time of day. The interrupt processed keyboard uses a periodic interrupt to scan through the keys of the keyboard.

Real-Time Clock

Figure 12-26 illustrates a simple circuit that uses the 60 Hz AC power line to generate a periodic interrupt request signal for the NMI interrupt input pin. Although we are using a signal from the AC power line, which varies slightly in frequency from time to time, it is accurate over a period of time.

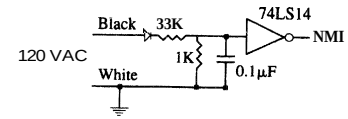
The circuit uses a signal from the 120 VAC power line that is conditioned by a Schmitt trigger inverter before it is applied to the NMI interrupt input. Note that you must make certain that the power line ground is connected to the system ground in this schematic. The power line ground (neutral) connection is the large flat pin on the power line. The narrow flat pin is the hot side or 120 VAC side of the line.

The software for the real-time clock contains an interrupt service procedure that is called 60 times per second and a procedure that updates the count located in four memory locations. Example 12-14 lists both procedures, along with the four bytes of memory used to hold the BCD time of day.

EXAMPLE 12–14

```
                .MODEL TINY
0000            .CODE
                .STARTUP
0100 EB 04          JMP   TIMES
0102 00         TIME  DB    ?             ;1/60 second counter
0103 00               DB    ?             ;seconds counter
0104 00               DB    ?             ;minutes counter
0105 00               DB    ?             ;hours counter
                ;
                ;Interrupt handler for NMI
                ;
0106            TIMES PROC  FAR

0106  50              PUSH  AX            ;save registers
0107  56              PUSH  SI

0108  B4 60           MOV   AH,60H        ;load modulus 60
010A  BE 0102 R        MOV   SI,OFFSET TIME ;address time
010D  E8 0014          CALL  UP            ;increment 1/60 counter
0110  75 0F            JNZ   DONE
0112  E8 000F          CALL  UP            ;increment seconds
0115  75 0A            JNZ   DONE
0117  E8 000A          CALL  UP            ;increment minutes
011A  75 05            JNZ   DONE
011C  B4 24            MOV   AH,24H        ;load modulus 24
011E  E8 0003          CALL  UP            ;increment minutes
0121                DONE:
0121  5E               POP   SI            ;reload registers
0122  58               POP   AX
0123  CF               IRET

0124            TIMES ENDP

0124            UP    PROC  NEAR
```

FIGURE 12-26 Converting the AC power line to a 60 Hz TTL signal for the NMI input.



```
0124  2E:: 8A 04      MOV   AL,CS:[SI]      ;get count
0127  46              INC   SI              ;address next counter
0128  04 01           ADD   AL,1            ;increment count
012A  27              DAA                   ;make it BCD
012B  2E: 88 44 FF    MOV   CS:[SI-1],AL    ;save count
012F  2A C4           SUB   AL,AH           ;test modulus
0131  75 04           JNZ   UP1
0133  2E: 88 44 FF    MOV   CS:[SI-1],AL    ;clear count
0137            UP1:
0137  C3              RET

0138            UP    ENDP

                      END
```

Interrupt-Processed Keyboard

The interrupt-processed keyboard scans through the keys on a keyboard through a periodic interrupt. Each time the interrupt occurs, the interrupt-service procedure tests for a key or debounces the key. Once a valid key is detected, the interrupt-service procedure stores the key-code into a keyboard queue for later reading by the system. The basis for this system is a periodic interrupt that can be caused by a timer or other device in the system. Note that most systems already have a periodic interrupt for the real-time clock. In this example, we assume that the interrupt calls the interrupt-service procedure every 10 ms.

Figure 12-27 shows the keyboard interfaced to an 8255. It does not show the timer or other circuitry required to call the interrupt, once in every 10 ms. (Not shown in the software is programming of the 82C55.) The 82C55 must be programmed so that port A is an input port, port B is an output port, and the initialization software must store a OOH at port B. This interfaces uses memory that is stored in the code segment for a queue and a few bytes that keep track of the keyboard scanning. Example 12-15 lists the interrupt service procedure for the keyboard.

EXAMPLE 12–15

```
;interrupt procedure for the keyboard in
;figure 12-27
;
.MODEL TINY
```

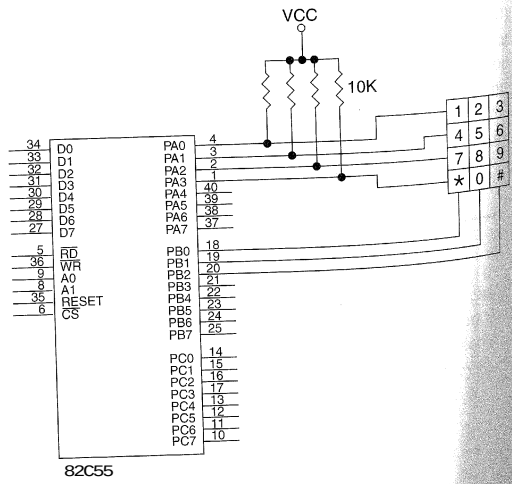


FIGURE 12-27 A telephone style keypad interfaced to the 82C55.


```
                    .386
0000                .CODE

= 1000              PORTA EQU   1000H              ;define port A address
= 1001              PORTB EQU   1001H              ;define Port B address

                    .STARTUP
0100                INTKEY PROC FAR USES AX DX ;keyboard interrupt
0102 BA 1000                MOV   DX,PORTA
0105 EC                     IN AL,DX               ; test for any key
0106 OC FO                  OR AL,0F0H
                            .IF AL != OFFH         ;if key found
010C FE 06 0192 R              INC DBCNT           ;increment bounce count
                               .IF DBCNT==3        ;if > 20 ms
0117 C6 06 0192 R 02              MOV DBCNT,2
                                  .IF DBF==0
0123 53                              PUSH BX
0124 C6 06 0193 R 01                 MOV  DBF,1
0129 BB OOFB                         MOV  BX,0FBH
                                     .WHILE 1     ;find key
012C BA 1001                            MOV DX,PORTB
012F 8A C3                              MOV AL,BL
0131 EE                                 OUT DX,AL
0132 DO CB                              ROR BL,1
0134 BA 1000                            MOV DX,PORTA
0137 EC                                 IN  AL,DX
0138 OC FO                              OR  AL,0F0H
                                        .BREAK .IF AL != OFFH
013E 80 C7 04                           ADD BH,4
                                     .ENDW
0143 8A D8                           MOV BL,AL
0145 BA 1001                         MOV DX,PORTB
0148 BO 00                           MOV AL,0
014A EF                              OUT DX,AX    ;clear port B pins
014B FE CF                           DEC BH
                                     .REPEAT      ;find key code
014D DO EB                              SHR BL,1
014F FE C7                              INC BH
                                     .UNTIL !CARRY?
0153 8A C7                           MOV AL,BH
0155 8B 1E 0194 R                    MOV BX,PNTR
0159 2E: 88 07                       MOV CS:[BX],AL ;key code to queue
015C FF 06 0194 R                    INC PNTR
                                     .IF BX == OFFSET DBCNT-1
0166 C7 06 0194 R 0182 R                MOV PNTR, OFFSET QUEUE
                                     .ENDIF
016C 5B                              POP BX
                                  .ENDIF
                               .ENDIF

                            .ELSE                  ;no key found
016F FE OE 0192 R              DEC DBCNT           ;decrement bounce count
                               .IF SIGN?
0175 C6 06 0192 R 00              MOV   DBCNT,0    ;clear count and flag
017A C6 06 0193 R 00              MOV   DBF,0
                               .ENDIF
                            .ENDIF
                            IRET
0182                INTKEY ENDP
0182 0010 [         QUEUE DB 16 DUP(?)             ;keyboard queue
             00
               ]
0192 00             DBCNT DB 0                     ;debounce count
0193 00             DBF   DB 0                     ;debounce flag
0194 0182 R         PNTR  DW QUEUE                 ;pointer to queue
                    END
```

The keyboard-interrupt finds the key and stores the key-code in the queue. The code stored in the queue is a raw code that does not indicate the key number. For example, the key code for the 1-key is a OOH, the key code for a 4-key is a 01H, etc. There is no provision for a queue overflow in this software. It could be added, but in almost all cases it is difficult to out-type a 16 bytes queue.

Example 12-16 illustrates a procedure that removes data from the keyboard queue. This procedure is not interrupt-driven and is called only when information from the keyboard is needed in a program. Example 12-17 shows the caller software for the key procedure.

EXAMPLE 12–16

```
0198 01 04 07 0A        LOOK  DB  1,4,7,10          ;look up table
019C 02 05 08 00              DB  2,5,8,0
01A0 03 06 09 0B              DB  3,6,9,11

01A4                    KEY   PROC  NEAR  USES BX
01A5 8B 1E 0196 R             MOV   BX,OPNTR
                              .IF BX == PNTR        ;queue empty
01AF F9                         STC
                              .ELSE
01B2 2E: 8A 07                  MOV AL,CS:[BX]      ;get code from queue
01B5 43                         INC BX
                                .IF BX == OFFSET DBCNT-1
01BC BB 0182 R                     MOV BX,OFFSET QUEUE
                                .ENDIF
01BF 89 1E 0196 R               MOV OPNTR,BX
01C3 BB 0198 R                  MOV BX,OFFSET LOOK
01C6 2E: D7                     XLAT CS:LOOK
01C8 F8                         CLC
                              .ENDIF
                              RET
01CB                    KEY   ENDP
```

EXAMPLE 12–17

```
                        .REPEAT
01CB E8 FFD6              CALL    KEY
                        .UNTIL !CARRY?
```

12-6 SUMMARY

1. An interrupt is a hardware- or software-initiated call that interrupts the currently executing program at any point and calls a procedure. The procedure is called by the interrupt or an interrupt service procedure.
2. Interrupts are useful when an I/O device needs to be serviced only occasionally at low data-transfer rates.
3. The microprocessor has five instructions that apply to interrupts: BOUND, INT, INT 3, INTO, and IRET. The INT and INT 3 instructions call procedures with addresses stored in the interrupt vector whose type is indicated by the instruction. The BOUND instruction is a conditional interrupt that uses interrupt vector type number 5. The INTO instruction is a conditional interrupt that interrupts a program only if the overflow flag is set. Finally, the IRET instruction is used to return from interrupt service procedures.
4. The microprocessor has three pins that apply to its hardware interrupt structure: INTR, NMI, and INTA. The interrupt inputs are INTR and NMI, which are used to request interrupts and INTA is an output used to acknowledge the INTR interrupt request.

5. Real mode interrupts are referenced through a vector table that occupies memory locations OOOOOH-003FFH. Each interrupt vector is four bytes long and contains the offset and segment addresses of the interrupt service procedure. In protected mode, the interrupts reference the interrupt descriptor table (IDT) that contains 256 interrupt descriptors. Each interrupt descriptor contains a segment selector and a 32-bit offset address.
6. Two flag bits are used with the interrupt structure of the microprocessor: trap (TF) and interrupt enable (IF). The IF flag bit enables the INTR interrupt input, and the TF flag bit causes interrupts to occur after the execution of each instruction, as long as TF is active.
7. The first 32 interrupt vector locations are reserved for Intel use, with many predefined in the microprocessor. The last 224 interrupt vectors are for user use and can perform any function desired.
8. Whenever an interrupt is detected, the following events occur: (1) the flags are pushed onto the stack, (2) the IF and TF flag bits are both cleared, (3) the IP and CS registers are both pushed onto the stack, and (4) the interrupt vector is fetched from the interrupt vector table and the interrupt service subroutine is accessed through the vector address.
9. Tracing or single-stepping is accomplished by setting the TF flag bit. This causes an interrupt to occur after the execution of each instruction for debugging.
10. The non-maskable interrupt input (NMI) calls the procedure whose address is stored at interrupt vector type number 2. This input is positive-edge triggered.
11. The INTR pin is not internally decoded, as is the NMI pin. Instead, INTA is used to apply the interrupt vector type number to data bus connections DO-D7 during the INTA pulse.
12. Methods of applying the interrupt vector type number to the data bus during INTA vary widely. One method uses resisters to apply interrupt type number FFH to the data bus, while another uses a three-state buffer to apply any vector type number.
13. The 8259A programmable interrupt controller (PIC) adds at least eight interrupt inputs to the microprocessor. If more interrupts are needed, this device can be cascaded to provide up to 64 interrupt inputs.
14. Programming the 8259A is a two-step process. First, a series of initialization command words (ICWs) are sent to the 8259A, then a series of operation command words (OCWs) are sent.
15. The 8259A contains three status registers: IMR (interrupt mask register), ISR (in-service register), and IRR (interrupt request register).
16. A real-time clock is used to keep time in real-time. In most cases, time is stored in either binary or BCD form in several memory locations.

12–7 QUESTIONS AND PROBLEMS

1. What is interrupted by an interrupt?
2. Define the term *interrupt.*
3. What is called by an interrupt?
4. Why do interrupts free up time for the microprocessor?
5. List the interrupt pins found on the microprocessor.
6. List the five interrupt instructions for the microprocessor.
7. What is an interrupt vector?
8. Where are the interrupt vectors located in the microprocessor's memory?
9. How many different interrupt vectors are found in the interrupt vector table?
10. Which interrupt vectors are reserved by Intel?

11. Explain how a type 0 interrupt occurs.
12. Where is the interrupt descriptor table located for protected mode operation?
13. Each protected mode interrupt descriptor contains what information?
14. Describe the differences between a protected and real mode interrupt.
15. Describe the operation of the BOUND instruction.
16. Describe the operation of the INTO instruction.
17. What memory locations contain the vector for an INT 44H instruction?
18. Explain the operation of the IRET instruction.
19. What is the purpose of interrupt vector type number 7?
20. List the events that occur when an interrupt becomes active.
21. Explain the purpose of the interrupt flag (IF).
22. Explain the purpose of the trap flag (TF).
23. How is IF cleared and set?
24. How is TF cleared and set?
25. The NMI interrupt input automatically vectors through which vector type number?
26. Does the INTA signal activate for the NMI pin?
27. The INTR input is _____ -sensitive.
28. The NMI input is _____ -sensitive.
29. When the INTA signal becomes a logic 0, it indicates that the microprocessor is waiting for an interrupt _____ number to be placed on the data bus (DO-D7).
30. What is a FIFO?
31. Develop a circuit that places interrupt type number 86H on the data bus in response to the INTR input.
32. Develop a circuit that places interrupt type number CCH on the data bus in response to the INTR input.
33. Explain why pull-up resistors on DQ-D7 cause the microprocessor to respond with interrupt vector type number FFH for the INTA pulse.
34. What is a daisy-chain?
35. Why must interrupting devices be polled in a daisy-chained interrupt system?
36. What is the 8259A?
37. How many 8259As are required to have 64 interrupt inputs?
38. What is the purpose of the IRO-IR7 pins on the 8259A?
39. When are the CAS2-CASO pins used on the 8259A?
40. Where is a slave INT pin connected on the master 8259A in a cascaded system?
41. What is an ICW?
42. What is an OCW?
43. How many ICWs are needed to program the 8259A when operated as a single master in a system?
44. Where is the vector type number stored in the 8259A?
45. Where is the sensitivity of the IR pins programmed in the 8259A?
46. What is the purpose of ICW1?
47. What is a non-specific EOI?
48. Explain priority rotation in the 8259A.
49. What is the purpose of IRR in the 8259A?
50. At which I/O ports is the master 8259A PIC found in the personal computer?
51. At which I/O ports is the slave 8259A found in the personal computer?

CHAPTER 13

Direct Memory Access and DMA-Controlled I/O

INTRODUCTION

In previous chapters, we discussed basic and interrupt-processed I/O. Now we turn to the final form of I/O called **direct memory access** (DMA). The DMA I/O technique provides direct access to the memory while the microprocessor is temporarily disabled. This allows data to be transferred between memory and the I/O device at a rate that is limited only by the speed of the memory components in the system or the DMA controller. The DMA transfer speed can approach 32-40 M-byte transfer rates with today's high-speed RAM memory components.

DMA transfers are used for many purposes, but more common are DRAM refresh, video displays for refreshing the screen, and disk memory system reads and writes. The DMA transfer is also used to do high-speed memory-to-memory transfers.

This chapter also explains the operation of disk memory systems and video systems that are often DMA-processed. Disk memory includes floppy, fixed, and optical disk storage. Video systems include digital and analog monitors.

CHAPTER OBJECTIVES

Upon completion of this chapter, you will be able to:

1. Describe a DMA transfer.
2. Explain the operation of the HOLD and HLDA direct memory access control signals.
3. Explain the function of the 8237 DMA controller when used for DMA transfers.
4. Program the 8237 to accomplish DMA transfers.
5. Describe the disk standards found in personal computer systems.
6. Describe the various video interface standards that are found in the personal computer.

13–1

BASIC DMA OPERATION

Two control signals are used to request and acknowledge a direct memory access (DMA) transfer in the microprocessor-based system. The HOLD pin is an input that is used to request a DMA action and the HLDA pin is an output that acknowledges the DMA action. Figure 13–1 shows the timing that is typically found on these two DMA control pins.