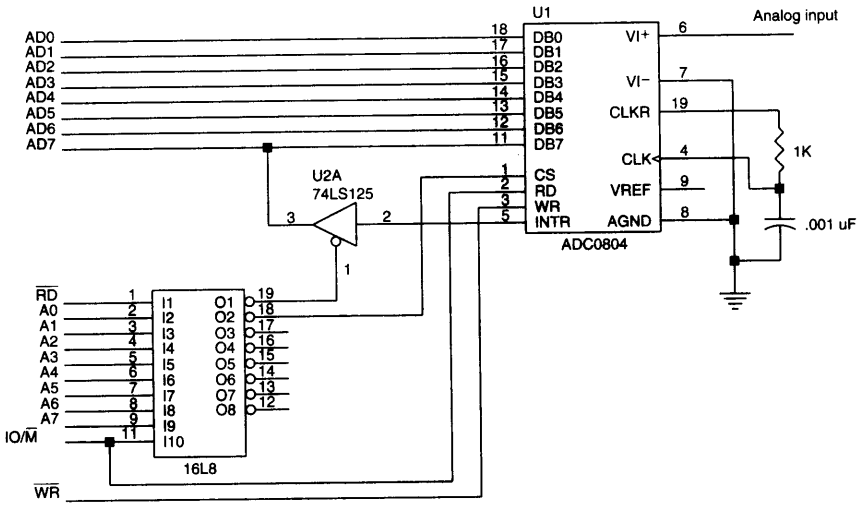


FIGURE 10–60 The ADC0804 interfaced to the microprocessor.

that the INTR bit is polled and if it becomes a logic 0, the procedure ends with AL, containing the converted digital code.

EXAMPLE 10–31

```
                        ;A procedure that reads data from the ADC and returns
                        ;it in AL.
                        ;
0000                    ADCX   PROC   NEAR

0000  E6 40                    OUT    40H,AL       ;start conversion
0002                    ADCX1:
0002  E4 42                    IN     AL,42H       ;read INTR
0004  A8 80                    TEST   AL,80H       ;test INTR
0006  75 FA                    JNZ    ADCX1        ;repeat until INTR = 0
0008  E4 40                    IN     AL,40H       ;get ADC data
000A  C3                       RET

000B                    ADCX   ENDP
```

Using the ADC0804 and the DAC0830

This section of the text illustrates an example that uses both the ADC0804 and the DAC0830 to capture and replay audio signals or speech. In the past, we often used a speech synthesizer to generate speech, but the quality of the speech was poor. For human quality speech, we can use the ADC0804 to capture an audio signal and store it in memory for later playback through the DAC0830.

Figure 10–61 illustrates the circuitry required to connect the ADC0804 at I/O ports 0700H and 0702H. The DAC0830 is interfaced at I/O port 704H. These I/O ports are in the low bank of a 16-bit microprocessor such as the 8086 or 80386SX. The software used to run these converters appears in Example 10–32. This software reads a 1-second burst of speech and then plays it back 10 times. This process repeats until the system is turned off. In this example, speech is sampled and stored in a section of memory called WORDS. The sample rate is chosen at 2048 samples per second, which renders acceptable-sounding speech.

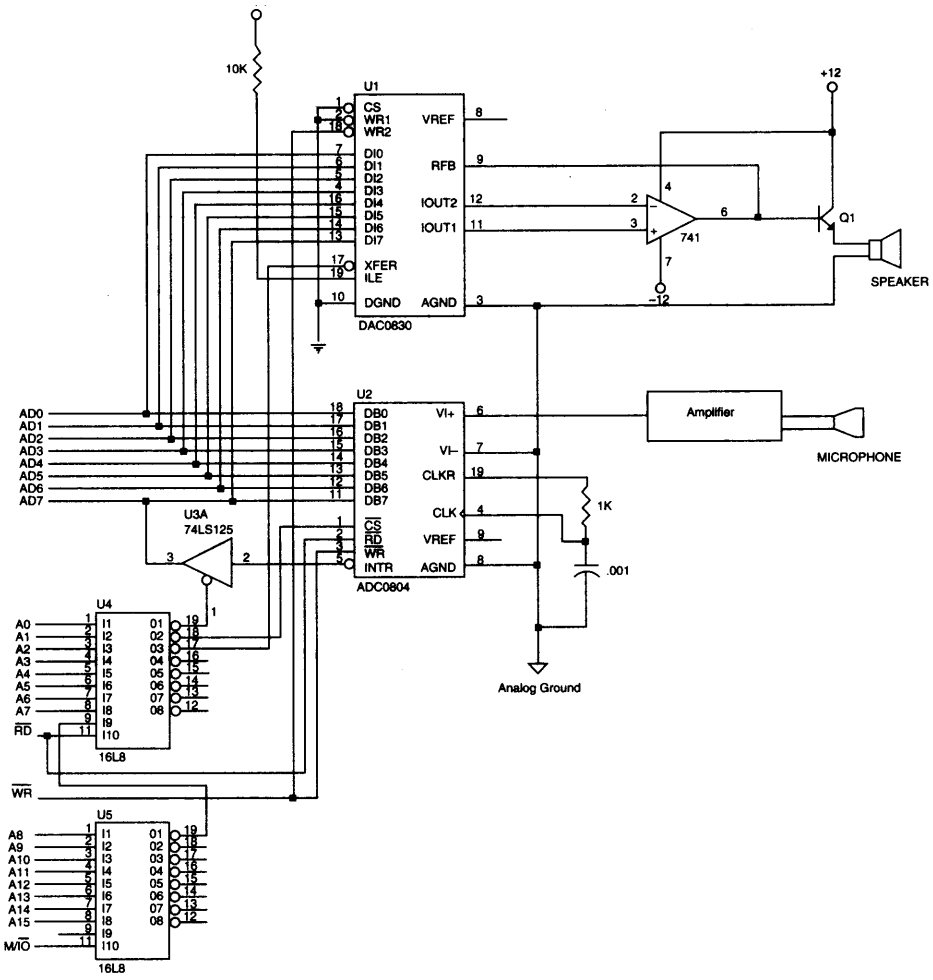


FIGURE 10–61 A circuit that stores speech and plays it back through the speaker.

EXAMPLE 10–32

```
                        ;Software that records a 1-second passage of speech
                        ;and plays it back 10 times before recording the
                        ;next 1-second passage of speech.
                        ;
                        ;Assumes a clock of 8 MHz (8086) for the time delay.
                        ;
                        .MODEL SMALL
                        .DATA
0000 0500 [             WORDS DB 2048 DUP (?)     ;space for speech
          0000
              ]
                        .CODE
                        .STARTUP
0018                AGAIN:
0018  E8 000A           CALL  READ                ;read speech
001B  B9 000A           MOV   CX,10               ;set count to 10
001E                LOOP1:
001E  E8 0023           CALL  WRITE               ;playback speech
```

```
0021  E2 FB          LOOP  LOOP1              ;repeat 10 times
0023  EB EA          JMP   AGAIN              ;repeat forever
0025           READ  PROC  NEAR
0025  BF 0000 R      MOV   DI,OFFSET WORDS    ;address data area
0028  B9 0500        MOV   CX,2048            ;load count
002B  BA 0700        MOV   DX,0700H           ;address port
002E           READ1:
002E  EE             OUT   DX,AL              ;start converter
002F  83 C2 C0       ADD   DX,2               ;address status port
0032           READ2:
0032  EC             IN    AL,DX              ;get INTR
0033  A8 80          TEST  AL,80H             ;test INTR
0035  75 FB          JNZ   READ2              ;wait for INTR = 0
0037  83 EA 02       SUB   DX,2               ;address data port
003A  EC             IN    AL,DX              ;get data from ADC
003B  88 05          MOV   [DI],AL            ;store data in array
003D  47             INC   DI                 ;address next element
003E  E8 0018        CALL  DELAY              ;wait for 1/2048 seconds
0041  E2 EB          LOOP  READ1              ;repeat 2,048 times
0043  C3             RET

0044           READ  ENDP

0044           WRITE PROC  NEAR

0044  51             PUSH  CX
0045  BF 0000 R      MOV   DI,OFFSET WORDS    ;address data
0048  B9 0500        MOV   CX,2048            ;load count
004B  BA 0704        MOV   DX,0704H           ;address DAC
004E           WRITE1:
004E  8A 05          MOV   AL,[DI]            ;get data from array
0050  EE             OUT   DX,AL              ;send data to DAC
0051  47             INC   DI                 ;address next element
0052  E8 0004        CALL  DELAY              ;wait 1/2048 second
0055  E2 F7          LOOP  WRITE1             ;repeat 2,048 times
0057  59             POP   CX
0058  C3             RET

0059           WRITE ENDP

0059           DELAY PROC  NEAR

0059  51             PUSH  CX
005A  B9 00E1        MOV   CX,225             ;approximately 1/2048 sec.
005D           DELAY1:
005D  E2 FE          LOOP   DELAY1
005F  59             POP   CX
0060  C3             RET

0061           DELAY ENDP
               .END
```

10–8 SUMMARY

1. The 8086–Pentium 4 microprocessors have two basic types of I/O instructions: IN and OUT. The IN instruction inputs data from an external I/O device into either the AL (8-bit) or AX (16-bit) register. The IN instruction is available as a fixed port instruction, a variable port instruction. The OUT instruction outputs data from AL or AX to an external I/O device and is available as a fixed variable. The fixed port instruction uses an 8-bit I/O port address, while the variable and string I/O instructions use a 16-bit port number found in the DX register.

2. Isolated I/O, sometimes called direct I/O, uses a separate map for the I/O space, freeing the entire memory for use by the program. Isolated I/O uses the IN and OUT instructions to transfer data between the I/O device and the microprocessor. The control structure of the I/O map uses IORC (I/O read control) and IOWC (I/O write control), plus the bank selectionsignals BHE and BLE (A0 on the 8086 and 80286), to affect the I/O transfer. The early 8086/8088 uses the M/IO (IO/M) signal with RD and WR to generate the I/O control signals.
3. Memory-mapped I/O uses a portion of the memory space for I/O transfers. This reduces the amount of memory available, but it negates the need to use the IORC and IOWC signals for I/O transfers. In addition, any instruction that addresses a memory location using any addressing mode can be used to transfer data between the microprocessor and the I/O device using memory-mapped I/O.
4. All input devices are buffered so that the I/O data are connected only to the data bus during the execution of the IN instruction. The buffer is either built into a programmable peripheral or located separately.
5. All output devices use a latch to capture output data during the execution of the OUT instruction. This is necessary because data appear on the data bus for less than 100 ns for an OUT instruction, and most output devices require the data for a longer time. In many cases, the latch is built into the peripheral.
6. Handshaking or polling is the act of two independent devices synchronizing with a few control lines. For example, the computer asks a printer if it is busy by inputting the BUSY signal from the printer. If it isn't busy, the computer outputs data to the printer and informs the printer that data are available with a data strobe (DS) signal. This communication between the computer and the printer is a handshake or a poll.
7. Interfaces are required for most switch-based input devices and for most output devices that are not TTL-compatible.
8. The I/O port number appears on address bus connections A7–A0 for a fixed port I/O instruction and on A15–A0 for a variable port I/O instruction (note that A15–A8 contain zeros for an 8-bit port). In both cases, address bits above A15 are undefined.
9. Because the 8086/80286/80386SX microprocessors contain a 16-bit data bus and the I/O addresses reference byte-sized I/O locations, the I/O space is also organized in banks, as is the memory system. In order to interface an 8-bit I/O device to the 16-bit data bus, we often require separate write strobes (an upper and a lower) for I/O write operations. Likewise, the 80486 and Pentium–Pentium 4 also have I/O arranged in banks.
10. The I/O port decoder is much like the memory address decoder, except instead of decoding the entire address, the I/O port decoder decodes only a 16-bit address for variable port instructions and often an 8-bit port number for fixed I/O instructions.
11. The 82C55 is a programmable peripheral interface (PIA) that has 24 I/O pins that are programmable in two groups of 12 pins each (group A and group B). The 82C55 operates in three modes: simple I/O (mode 0), strobed I/O (mode 1), and bi-directional I/O (mode 2). When the 82C55 is interfaced to the 8086 operating at 8 MHz, we insert two wait states because the speed of the microprocessor is faster than the 82C55 can handle.
12. The 8279 is a programmable keyboard/display controller that can control a 64-key keyboard and a 16-digit numeric display.
13. The LCD display device requires a fair amount of software, but it displays ASCII-coded information.
14. The 8254 is a programmable interval timer that contains three 16-bit counters that count in binary or binary-coded decimal (BCD). Each counter is independent of each other, and operates in six different modes. The six modes of the counter are (1) events counter, (2) retriggerable monostable multivibrator, (3) pulse generator, (4) square-wave generator, (5) software-triggered pulse generator, and (6) hardware triggered pulse generator.
15. The 16550 is a programmable communications interface, capable of receiving and transmitting asynchronous serial data.
16. The DAC0830 is an 8-bit digital-to-analog converter that converts a digital signal to an analog voltage within 1.0 μs.
17. The ADC0804 is an 8-bit analog-to-digital converter that converts an analog signal into a digital signal within 100 μs.

10–9 QUESTIONS AND PROBLEMS

1. Explain which way the data flow for an IN and an OUT instruction.
2. Where is the I/O port number stored for a fixed I/O instruction?
3. Where is the I/O prot number stored for a string I/O instruction?
4. To which register are data input by the IN instruction?
5. Which signal of 80X86 is asserted when an I/O instruction is execerted?
6. Describe the operation of the OUTSB instruction.
7. What size of I/O space can the 8086 address?
8. Contrast a memory-mapped I/O system with an isolated I/O system.
9. What is the basic input interface?
10. What is the basic output interface?
11. Explain the term *handshaking* as it applies to computer I/O systems.
12. An even-number I/O port address is found in the _____ I/O bank in the 8086 microprocessor.
13. Show the circuitry that generates the upper and lower I/O write strobes.
14. What is the purpose of a contact bounce eliminator?
15. Develop an interface to correctly drive a relay. The relay is 12 V and requires a coil current of 150 mA.
16. Develop an I/O port decoder, using a 74ALS138, which generates low-bank I/O strobes for the following 8-bit I/O port addresses: 10H, 12H, 14H, 16H, 18H, 1AH, 1CH, and 1EH.
17. Develop an I/O port decoder, using a 74ALS138, which generates high-bank I/O strobes for the following 8-bit I/O port addresses: 11H, 13H, 15H, 17H, 19H, 1BH, 1DH, and 1FH.
18. Develop an I/O port decoder, using a PAL16L8, which generates 16-bit I/O strobes for the following 16-bit I/O port addresses: 1000H–1001H, 1002H–1003H, 1004H–1005H, 1006H–1007H, 1008H–1009H, 100AH–100BH, 100CH–100DH, and 100EH–100FH.
19. Develop an I/O port decoder, using the PAL16L8, which generates the following low-bank I/O strobes: 00A8H, 00B6H, and 00EEH.
20. Develop an I/O port decoder, using the PAL16L8, which generates the following high-bank I/O strobes: 300DH, 300BH, 1005H, and 1007H.
21. Why are both BHE and BLE (A0) ignored in a 16-bit port address decoder?
22. An 8-bit I/O device, located at I/O port address 0010H, is connected to which data bus connections?
23. An 8-bit I/O device, located at I/O port address 100DH, is connected to which data bus connections?
24. The 82C55 has how many programmable I/O pin connections?
25. List the pins that belong to group A and to group B in the 82C55.
26. Which two 82C55 pins accomplish internal I/O port address selection?
27. The RD connection on the 82C55 is attached to which 8086 system control bus connection?
28. Using a PAL16L8, interface an 82C55 to the 8086 microprocessor so that it functions at I/O locations 0380H, 0382H, 0384H, and 0386H.
29. When the 82C55 is reset, its I/O ports are all initialized as _____.
30. What three modes of operation are available to the 82C55?
31. What is the purpose of the STB signal in strobed input operation of the 82C55?
32. Explain the operation of a simple four-coil stepper motor.
33. What sets the IBF pin in strobed input operation of the 82C55?
34. Write the software required to place a logic 1 on the PC7 pin of the 82C55 during strobed input operation.
35. How is the interrupt request pin (INTR) enabled in the strobed input mode of operation of the 82C55?
36. In strobed output operation of the 82C55, what is the purpose of the ACK signal?
37. What clears the OBF signal in strobed output operation of the 82C55?
38. Write the software required to decide whether PC4 is a logic 1 when the 82C55 is operated in the strobed output mode.

39. Which group of pins are used during bi-directional operation of the 82C55?
40. Which pins are general-purpose I/O pins during mode 2 operation of the 82C55?
41. Describe how the display is cleared by using the LCD display.
42. How is a display position selected in the LCD display?
43. Write a short procedure that places an ASCII-Z in display position 6 on the LCD display.
44. How is the busy flag tested in the LCD display?
45. What changes must be made to Figure 10–24 so that it functions with a keyboard matrix that contains three rows and five columns?
46. What time is usually used to de-bounce a keyboard?
47. What is normally connected to the CLK pin of the 8279?
48. How many wait states are required to interface the 8279 to the 8086 microprocessor operating with an 8 MHz clock?
49. If the 8279 CLK pin is connected to a 3.0 MHz clock, program the internal clock.
50. What is an overrun error in the 8279?
51. What is the difference between encoded and decoded, as defined for the 8279?
52. Interface the 8279 so that it functions at 8-bit I/O ports 40H–7FH. Use the 74ALS138 as a decoder, and use either the upper or lower data bus.
53. Interface a 16-key keyboard and an 8-digit numeric display to the 8279.
54. The 8254 interval timer functions from DC to _____ Hz.
55. Each counter in the 8254 functions in how many different modes?
56. Interface an 8254 to function at I/O port addresses XX10H, XX12H, XX14H, and XX16H. Write the software that programs counter 2 to generate an 80 KHz square-wave if the CLK input to counter 2 is 8 MHz.
57. What number is programmed in an 8254 counter to count 300 events?
58. If a 16-bit count is programmed into the 8254, which byte of the count is programmed first?
59. Explain how the read-back control word functions in the 8254.
60. Program counter 1 of the 8254 so that it generates a continuous series of pulses that have a high time of 100 μs and a low time of 1 μs. Make sure to indicate the CLK frequency required for this task.
61. Why does a 50 percent duty cycle cause the motor to stand still in the motor speed and direction control circuit presented in this chapter?
62. What is asynchronous serial data?
63. What is Baud rate?
64. Program the 16550 for operation using six data bits, even parity, one stop bit, and a Baud rate of 19,200 using a 18.432 MHz clock. (Assume that the I/O ports are numbered 20H and 22H.)
65. If the 16550 is to generate a serial signal at a Baud rate of 2400 Baud and the Baud rate divisor is programmed for 16, what is the frequency of the signal?
66. Describe the following terms: *simplex, half-duplex,* and *full-duplex.*
67. How is the 16550 reset?
68. Write a procedure for the 16550 that transmits 16 bytes from a small buffer in the data segment address (DS is loaded externally) by SI (SI is loaded externally).
69. The DAC0830 converts an 8-bit digital input to an analog output in approximately _____.
70. What is the step voltage at the output of the DAC0830 if the reference voltage is –2.55 V?
71. Interface a DAC0830 to the 8086 so that it operates at I/O port 400H.
72. Develop a program for the interface of Question number 74 so the DAC0830 generates a triangular voltage wave-form. The frequency of this wave-form must be approximately 100 Hz
73. The ADC080X requires approximately _____ to convert an analog voltage into a digital code.
74. What is the purpose of the INTR pin on the ADC080X?
75. The WR pin on the ADC080X is used for what purpose?
76. Interface an ADC080X at I/O port 0260H for data and 0270H to test the INTR pin.
77. Develop a program for the ADC080X in Question 76 so that it reads an input voltage once per 100 ms and stores the results in a memory array that is 100H bytes long.

CHAPTER 11

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-initiated 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.

11–1 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 10, for instance, we showed a keyboard example using strobed input operation of the

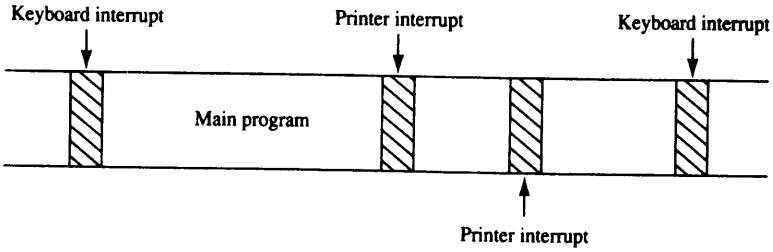


FIGURE 11–1 A time line that indicates interrupt usage in a typical system.

82C55. In that example, software polled the 82C55 and its 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 11–1 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 4).

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 000000H–0003FFH. It contains 256 different 4-byte interrupt vectors. An **interrupt vector** contains the address (segment and offset) of the interrupt service procedure.

Figure 11–2 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 4, 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.

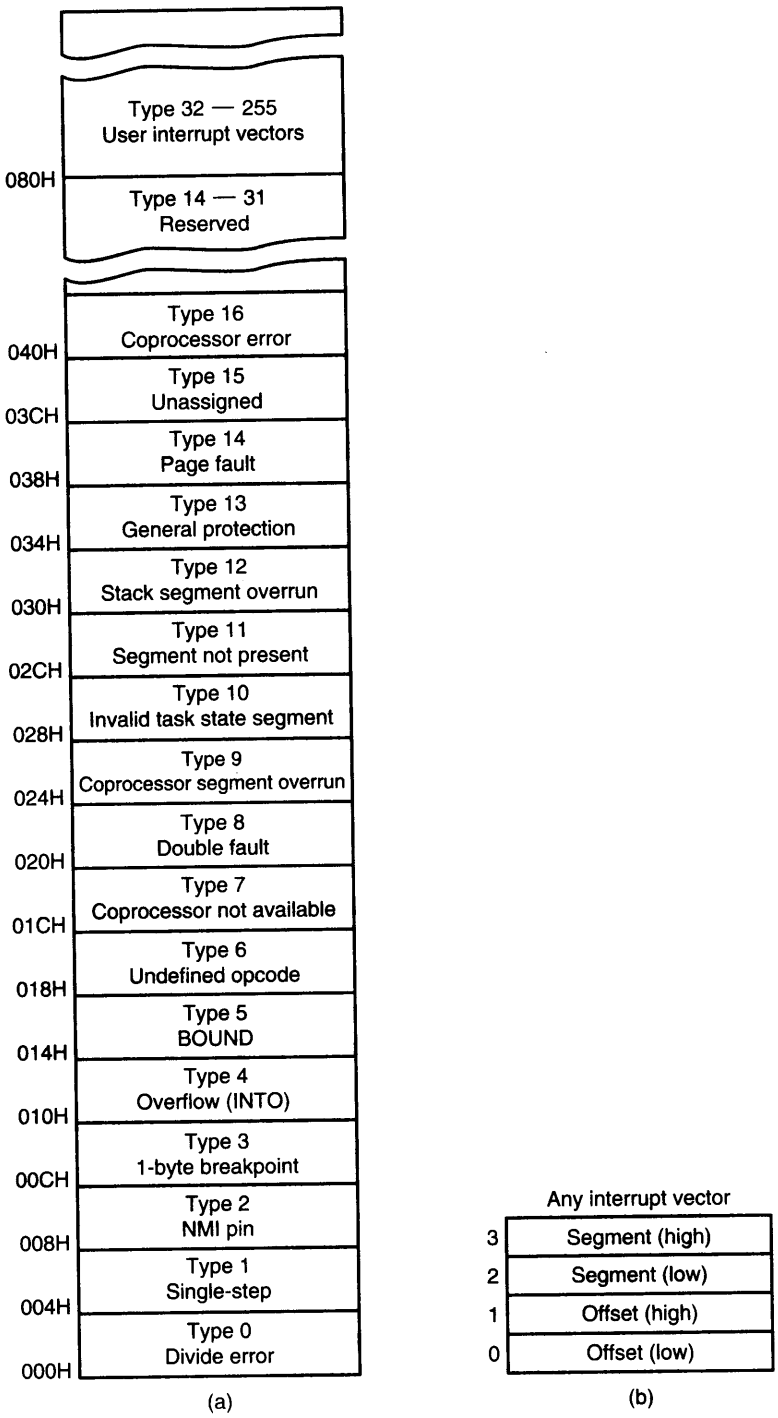


FIGURE 11–2 (a) The interrupt vector table for the microprocessor and (b) the contents of an interrupt vector.

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.

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. Applicable for 80286 unwards. Type 5 to Type 18.

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 4 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 4 microprocessors.

Type 16 Coprocessor Error—Takes effect whenever a coprocessor error (ERROR = 0) occurs for the ESCape or WAIT instructions for the 80386, 80486, and Pentium–Pentium 4 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 4 microprocessors.

Type 18 Machine Check—Activates a system memory management mode interrupt in the Pentium–Pentium 4 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 × 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 4, there is also an IRETD instruction because these microprocessors can push the EFLAG 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 4 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 × 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 11–3 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.

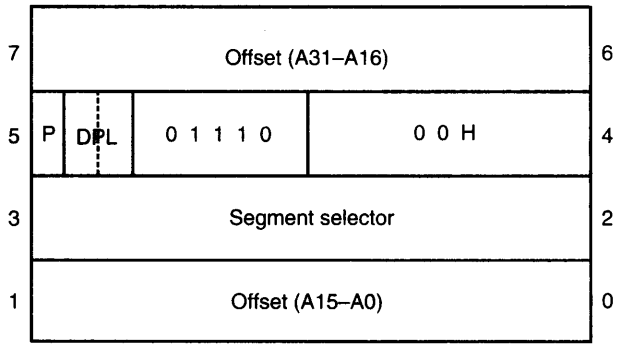


FIGURE 11–3 The protected mode interrupt descriptor.

FIGURE 11–4 The flag register. (Courtesy of Intel Corporation.)

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 11–4 illustrates the contents of the flag register and 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 15–17, which detail the 80386–Pentium 4.

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 11–1 shows an interrupt service procedure that turns tracing on by setting the trap flag bit on the stack from inside the procedure. Example 11–2 shows an interrupt service procedure that turns tracing off by clearing the trap flag on the stack from within the procedure.

EXAMPLE 11–1

```
                        ;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 11–2

```
                        ;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 11–3 traces through a program immediately following the INT 40H instruction. The interrupt service procedure illustrated in Example 11–3 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 11–3

```
                        .MODEL TINY
0000                    .CODE
0000  41 58 20 3D 20 42 RNAME DB 'AX = ','BX = ','CX = ','DX = '
      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 0F                       AND   AL,0FH        ;convert to ASCII
                                  .IF AL > 9
00D4  04 07                          ADD AL,7
                                  .ENDIF
00D6  04 30                       ADD   AL,30H
                                  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 11–4 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.

EXAMPLE 11–4

```
                        .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  D1 EA                  SHR   DX,1
012B  D1 EA                  SHR   DX,1
012D  42                     INC   DX
012E  B8 3100                MOV   AX,3100H
0131  CD 21                  INT   21H
                             END
```

11–2 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 00H through 1FH for internal and future expansion. The 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–D0. Figure 11–5 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 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 11–6 shows a power failure detection circuit that provides a logic 1 to the NMI input whenever AC power is interrupted.

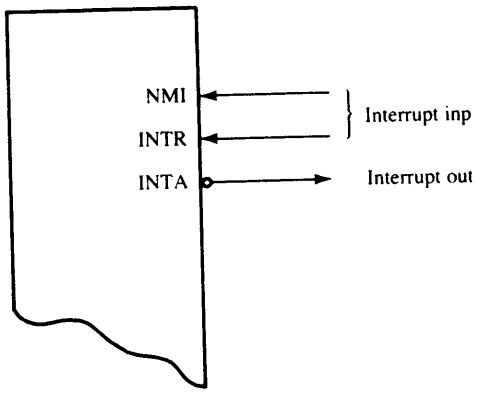


FIGURE 11–5 The interrupt pins on all versions of the Intel microprocessor.

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 11–7 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.

When DC power fails, the battery provides a reduced voltage to the V_{CC} connection on the memory device. Most memory devices will retain data with V_{CC} 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 V_{CC} during a power outage, so no data will be written to the memory.

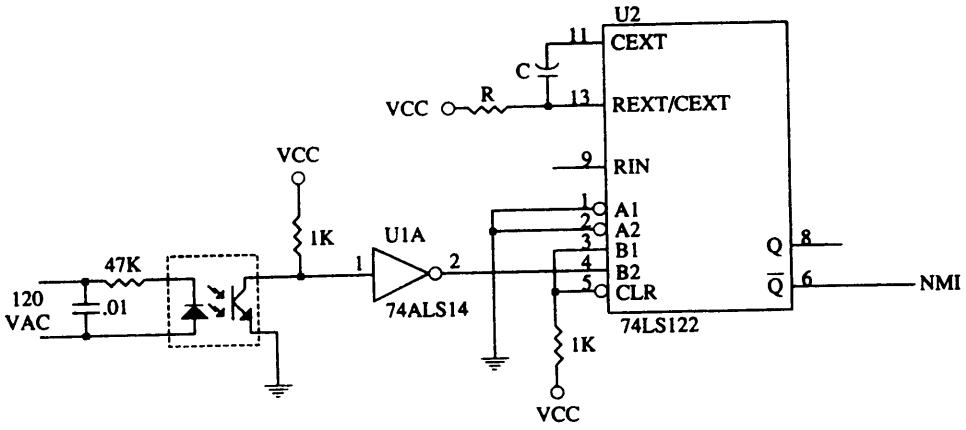


FIGURE 11–6 A power failure detection circuit.

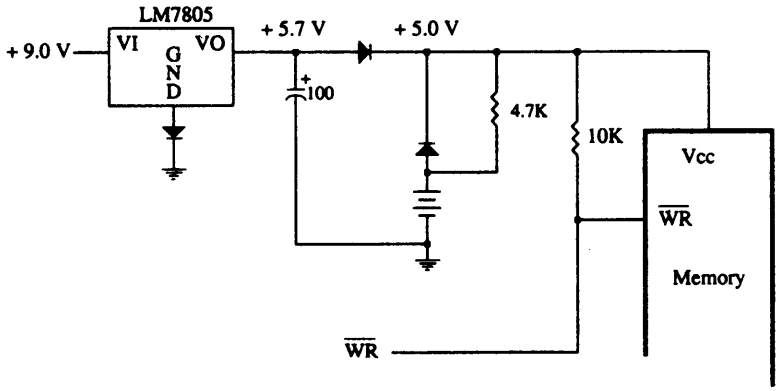


FIGURE 11–7 A battery-backed-up memory system using a NiCad, lithium, or gel cell.

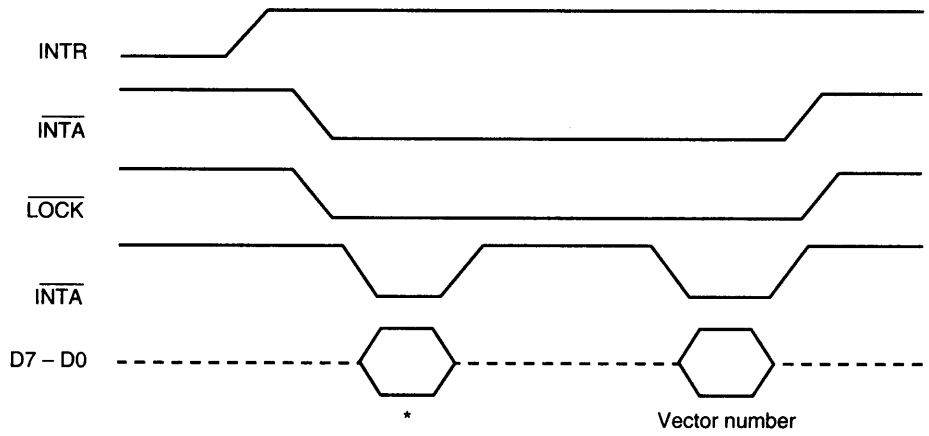


FIGURE 11–8 The timing of the INTR input and INTA output. **Note:* This portion of the data bus is ignored and usually contains the vector number.

INTR and 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 4 use the IRETD instruction in the protected mode of operation.

The microprocessor responds to the INTR input by pulsing the INTA output in anticipation of receiving an interrupt vector type number on data bus connection D7–D0. Figure 11–8 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.

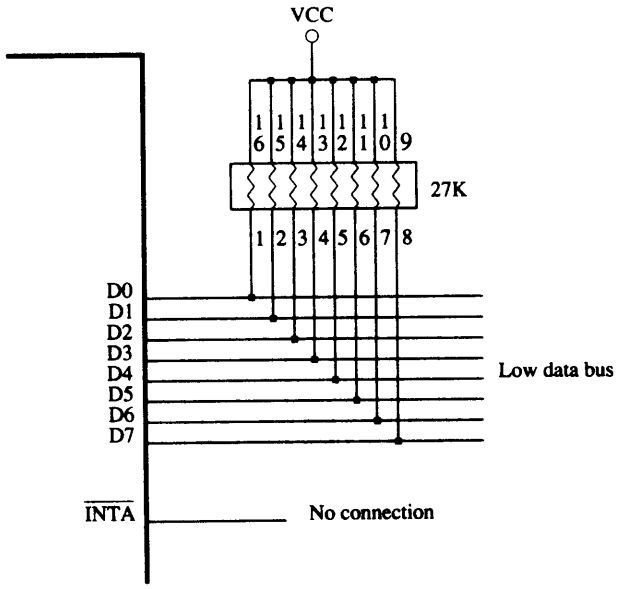


FIGURE 11–9 A simple method for generating interrupt vector type number FFH in response to INTR.

Figure 11–9 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 INTA. Figure 11–10 shows how interrupt vector type number 80H is applied to the data bus (D0–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 11–11. 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.

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 11–12 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

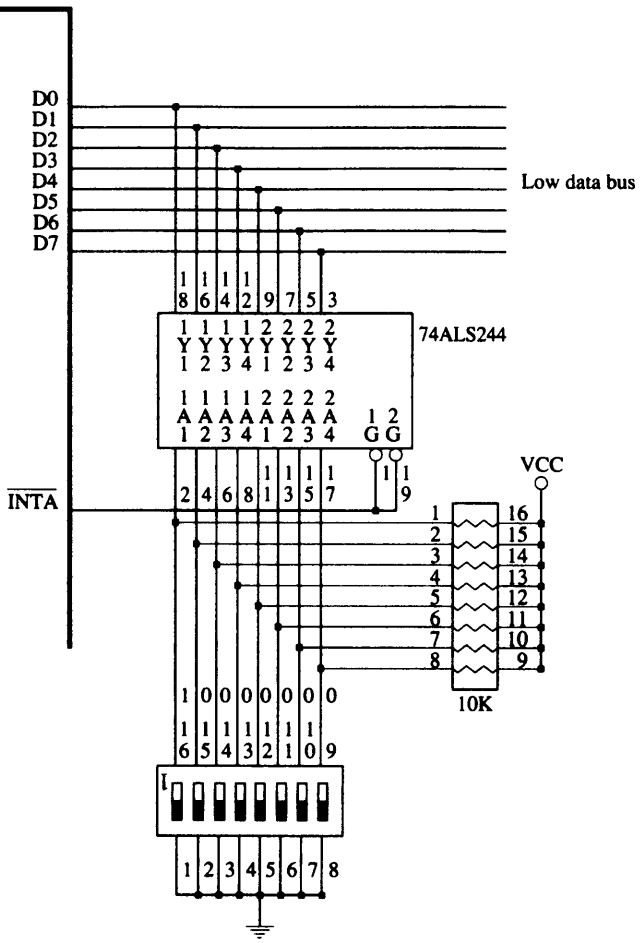


FIGURE 11–10 A circuit that applies any interrupt vector type number in response to INTA. Here the circuit is applying type number 80H.

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 11–5 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 11–5

```
                        ;An interrupt service procedure that reads a key
                        ;from the keyboard in Figure 11-12.
                        ;
= 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

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.

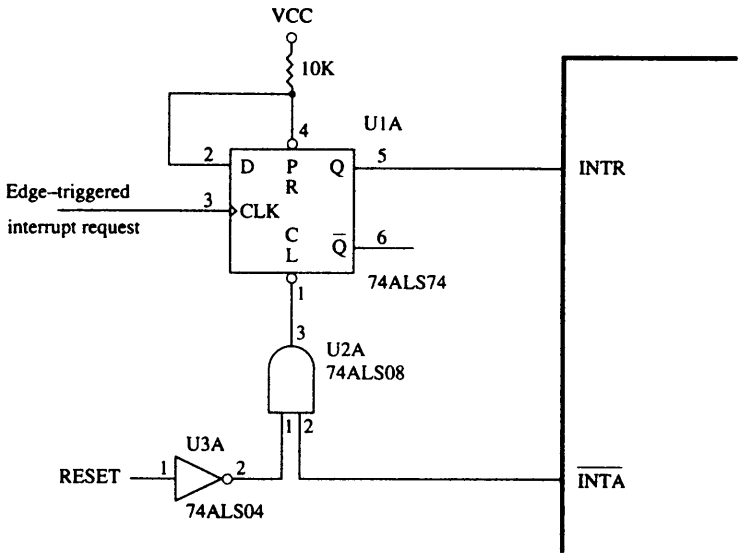


FIGURE 11–11 Converting INTR into an edge-triggered interrupt request input.

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 11–6 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.

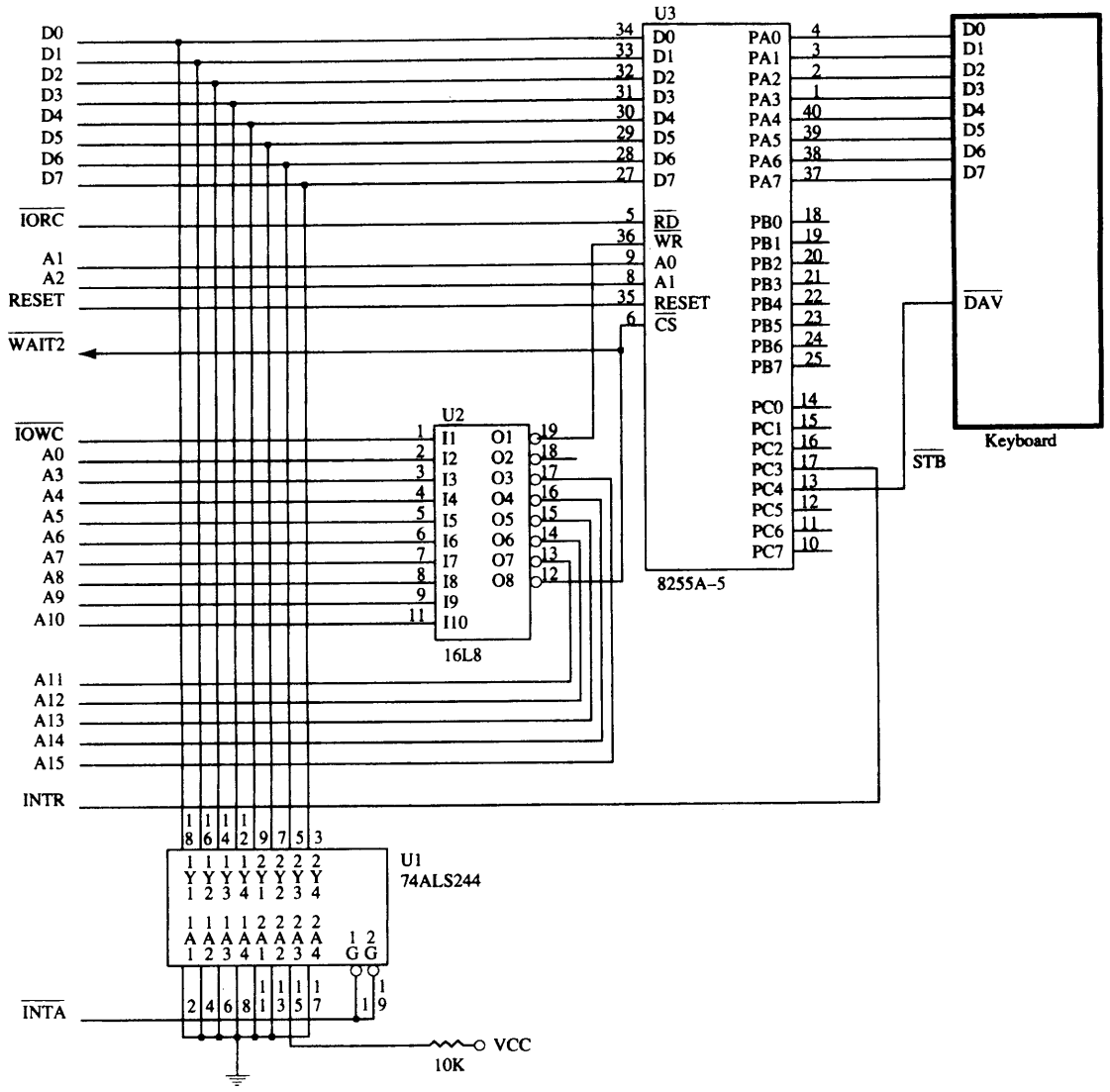


FIGURE 11–12 An 82C55 interfaced to a keyboard from the microprocessor system using interrupt vector 40H.

EXAMPLE 11-6

```
                        ;A procedure that reads data from the queue of
                        ;Example 11-5 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
```

11-3 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 11–10, 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 11–13 allows the circuit of Figure 11–10 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.

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 11–1 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.

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. Set-

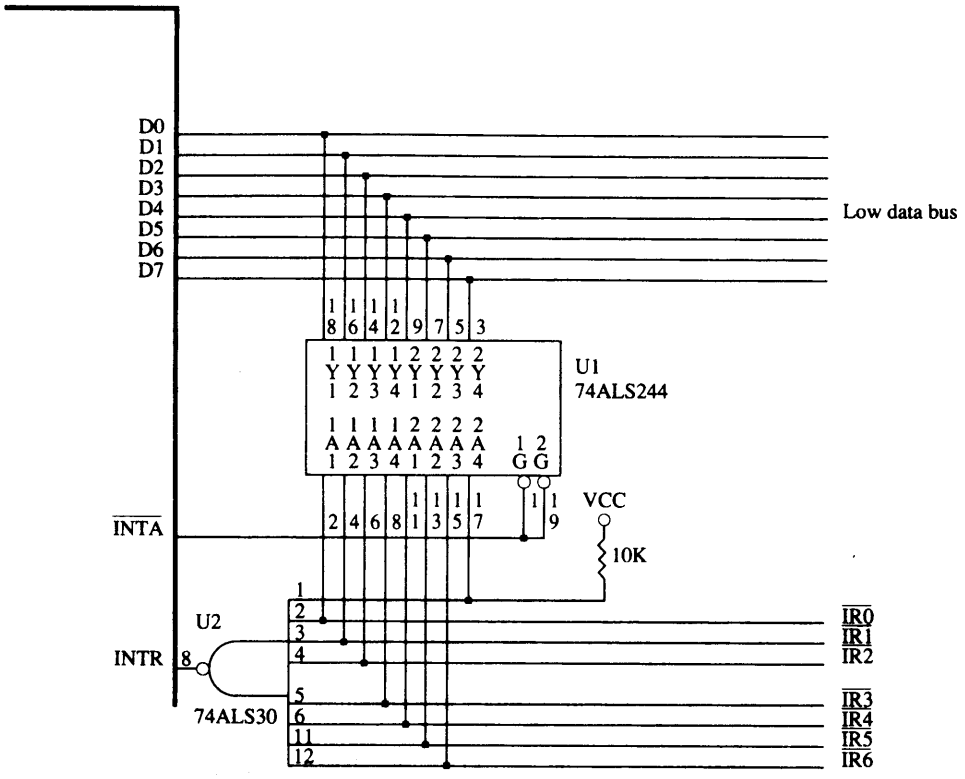


FIGURE 11–13 Expanding the INTR input from one to seven interrupt request lines.

TABLE 11–1 Single interrupt request for Figure 11–13.

IR6	*IR5*	*IR4*	*IR3*	*IR2*	*IR1*	*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.

ting 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 11–14 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 (D0–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.

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 11–7 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.

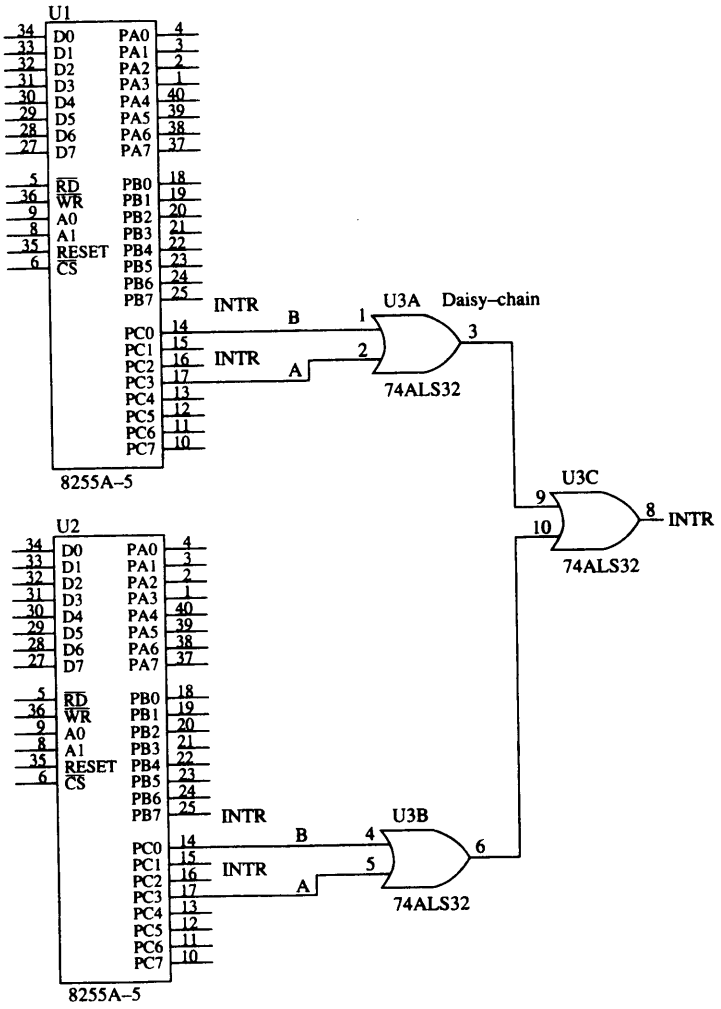


FIGURE 11–14 Two 82C55 PPI connected to the INTR outputs are daisy-chained to produce an INTR signal.

EXAMPLE 11–7

```
                        ;A procedure that services the daisy-chain interrupt
                        ;of Figure 11-14.
                        ;
= 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 0F                     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
```

11–4 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 11–15 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–D0 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 4 is used, then they connect to any 8-bit bank.

IR7–IR0 **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.

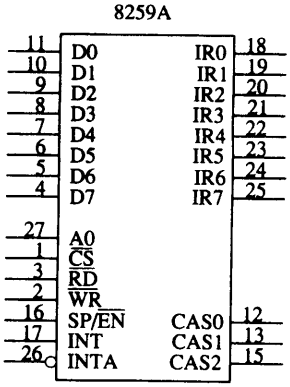


FIGURE 11–15 The pin-out of the 8259A programmable interrupt controller (PIC).

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.
INTA	**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.
A0	The A0 address input selects different command words within the 8259A.
CS	**Chip select** enables the 8259A for programming and control.
SP/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–CAS0	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 11–16 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 11–17 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–0FH. The AT-style computer uses interrupt vector 0AH 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–0FH 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.

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 A0 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 11–18 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 4 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. This ICW selects single or cascade operation by programming the SNGL bit.

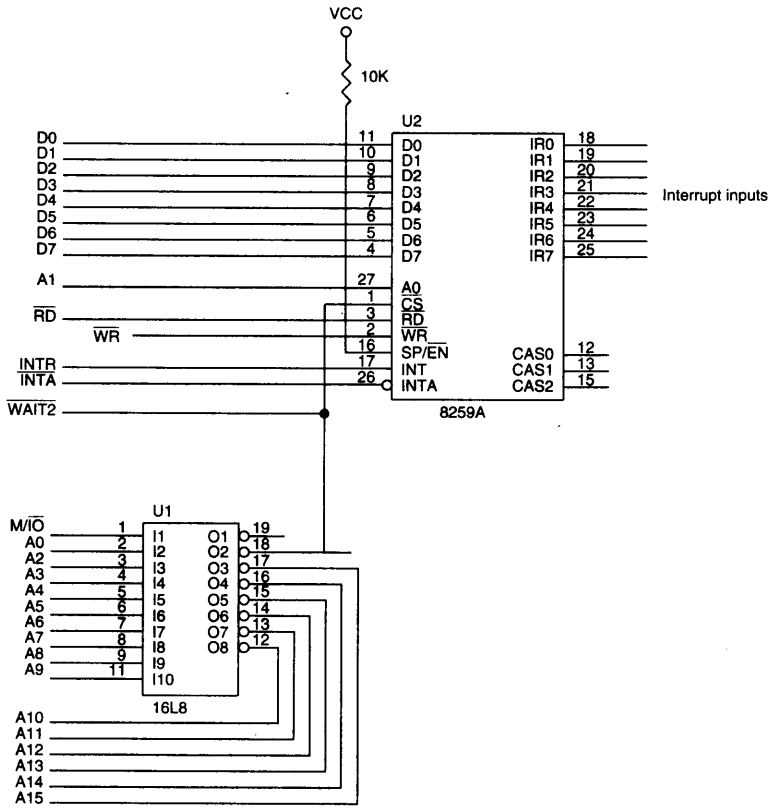


FIGURE 11–16 An 8259A interfaced to the 8086 microprocessor.

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–0FH, 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.

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 11–18 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 4 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 4 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

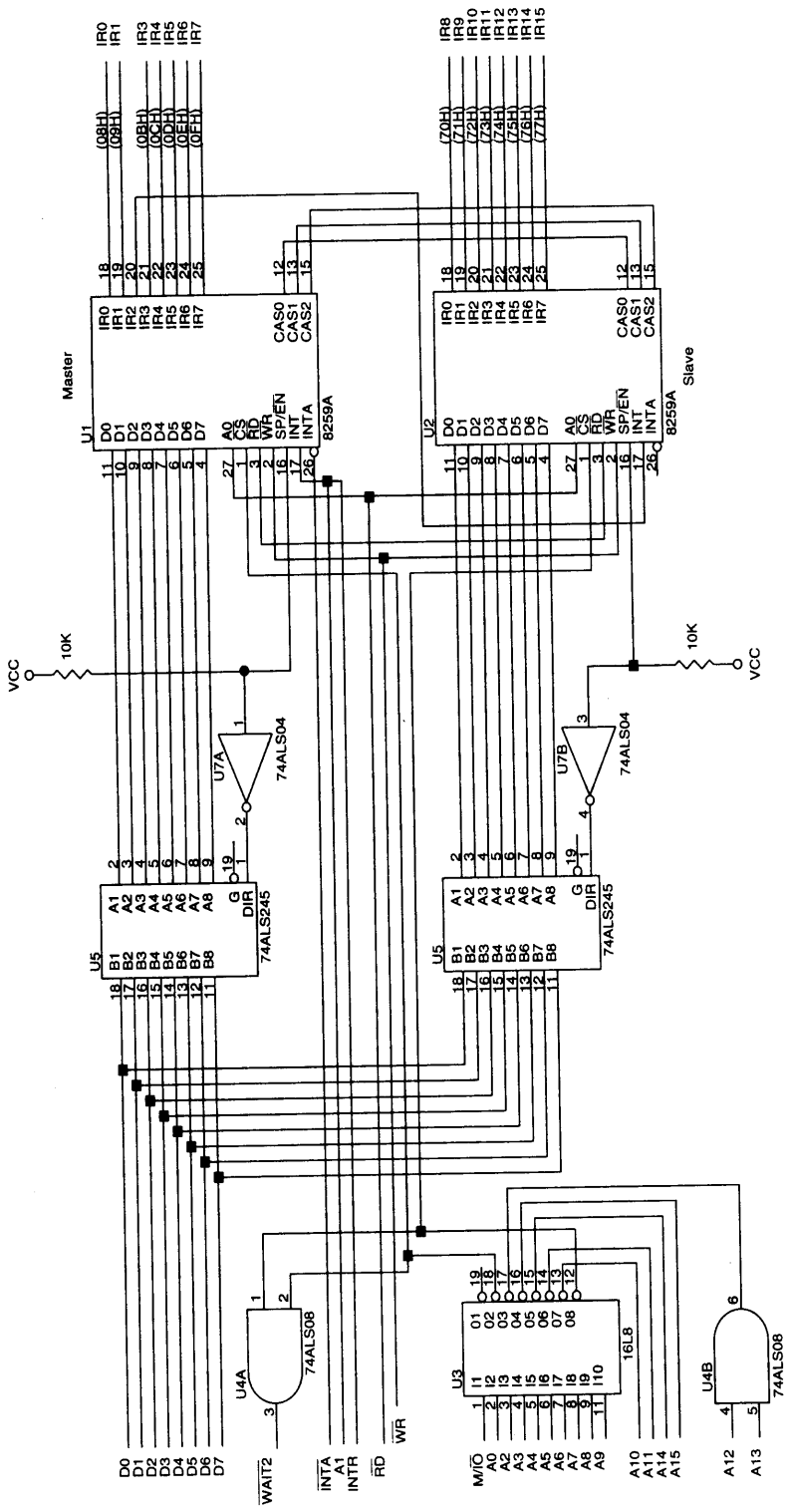


FIGURE 11–17 Two 8259As interfaced to the 8259A at I/O ports 0300H and 0302H for the master and 0304H and 0306H for the slave.

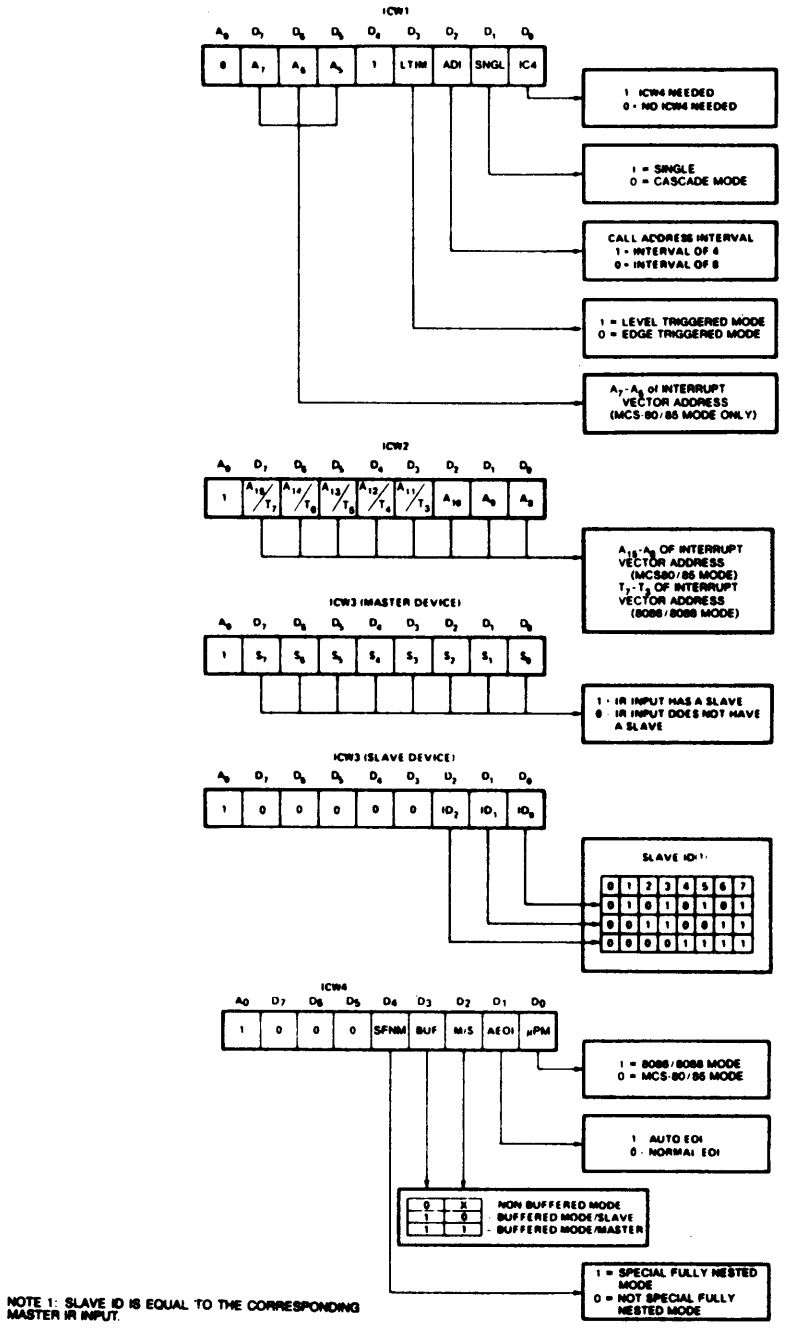


FIGURE 11–18 The 8259A initialization command words (ICWs) (Courtesy of Intel Corporation.)

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 A0 pin is at a logic 0 level, except for OCW1, which is selected when A0 is a logic 1. Figure 11–19 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–L0 of OCW2.

Rotate-on-Nonspecific EOI—A command that functions exactly like the Nonspecific 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.

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–L0 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 11–20 for all three status registers; they all have the

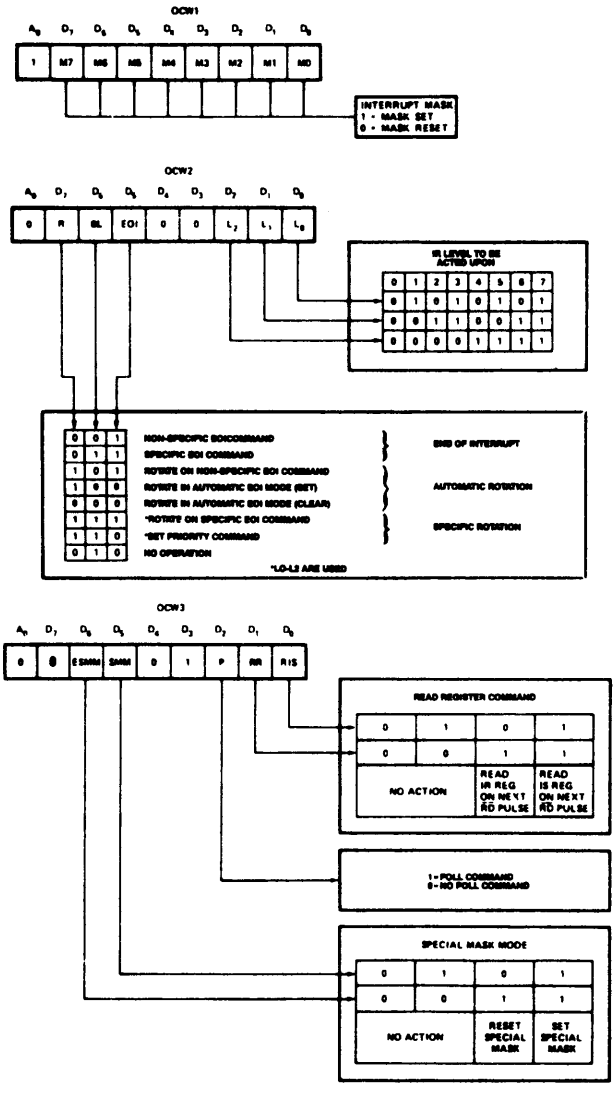


FIGURE 11–19 The 8259A operation command words (OCWs). (Courtesy of Intel Corporation.)

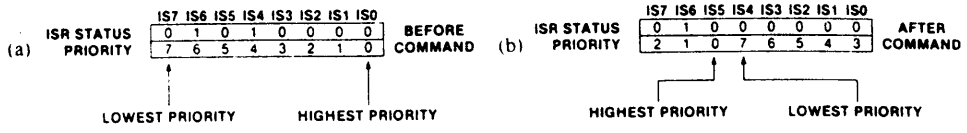


FIGURE 11–20 The 8259A in-service register (ISR). (a) Before IR4 is accepted and (b) after IR4 is accepted. (Courtesy of Intel Corporation.)

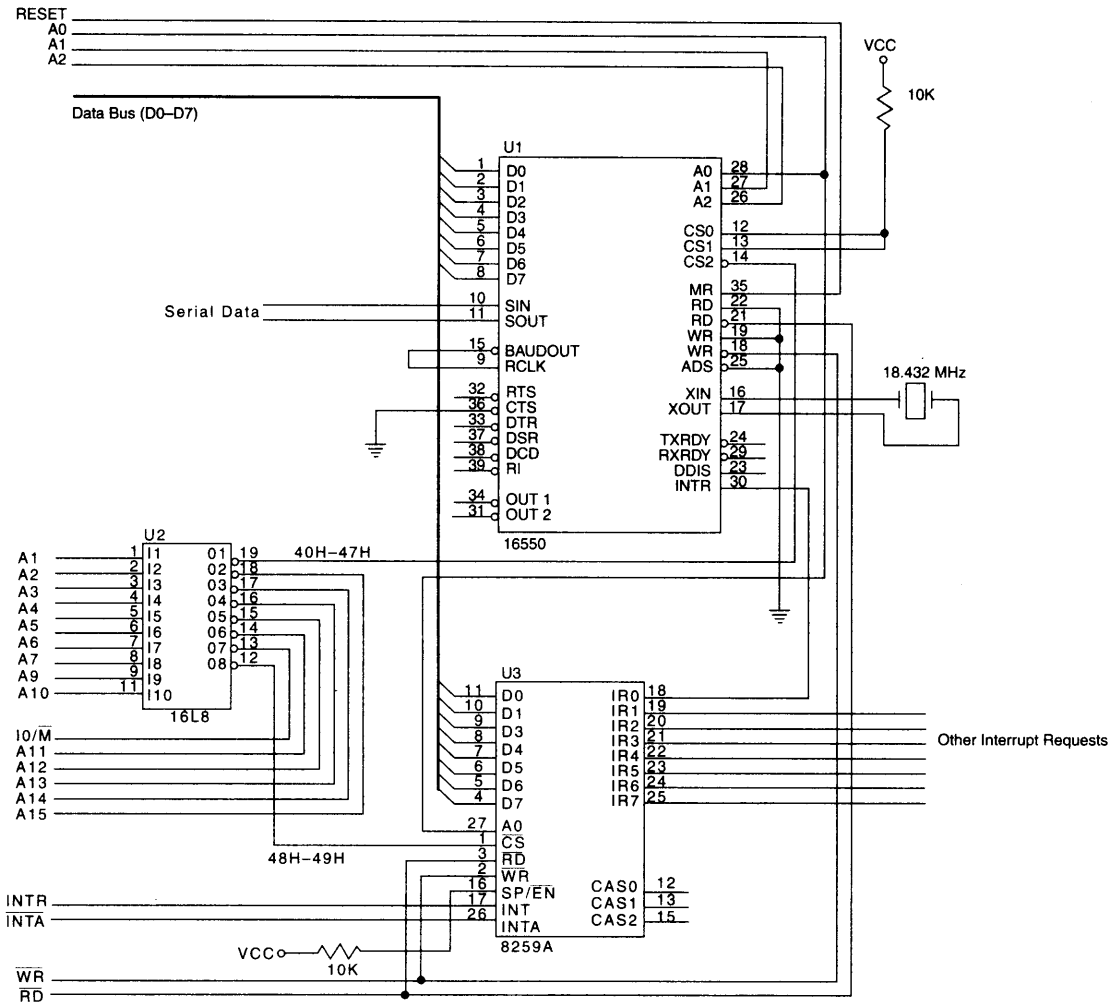


FIGURE 11–21 The 16550 UART interfaced to the 8088 microprocessor through the 8259A.

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, A0 = 1; to read either IRR or ISR, A0 = 0. Bit positions D0 and D1 of OCW3 select which register (IRR or ISR) is read when A0 = 0.

8259A Programming Example

Figure 11–21 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 char-

acter, (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 11–8 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 11–8

```
                        ;Initialization software for the 16550 and 8259A
                        ;of the circuit in Figure 11-21.
                        ;
= 0048                  PIC1  EQU   48H       ;8259A control A0 = 0
= 0049                  PIC1  EQU   49H       ;8259A control A0 = 1
= 001B                  ICW1  EQU   1bH       ;8259A ICW1
= 0080                  ICW2  EQU   80H       ;8259A ICW2
= 0003                  ICW4  EQU   3         ;8259A ICW4
= 00FE                  OCW1  EQU   0FEH      ;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  B0 8A                   MOV   AL,10001010B   ;enable Baud divisor
0002  E6 43                  OUT   LINE,AL

0004  B0 78                   MOV   AL,120         ;program Baud rate
0006  E6 40                   OUT   LSB,AL         ;9600 Baud rate
0008  B0 00                   MOV   AL,0
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. This program example for 8259 A is to introduce the instialization of the registers of 8259 A. Readers may find this example useful for developing their own applications.

11–5 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 handler 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 00000H–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 D0–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).

11–6 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 (D0–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 D0–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 IR0–IR7 pins on the 8259A?
39. When are the CAS2–CAS0 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?

CHAPTER 12

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 various video interface standards that are found in the personal computer.

12–1 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

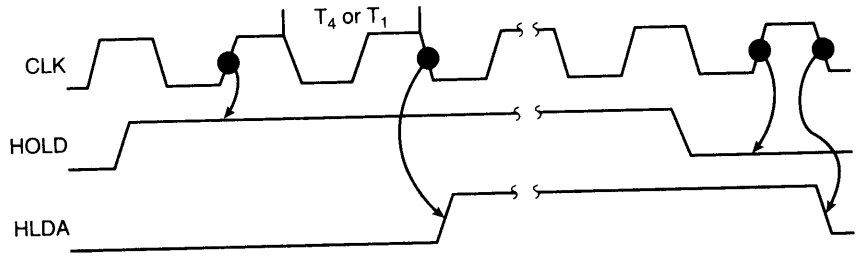


FIGURE 12–1 HOLD and HLDA timing for the microprocessor.

output that acknowledges the DMA action. Figure 12–1 shows the timing that is typically found on these two DMA control pins.

Whenever the HOLD input is placed at a logic 1 level, a DMA action (hold) is requested. The microprocessor responds, within a few clocks, by suspending the execution of the program and by placing its address, data, and control bus at their high-impedance states. The high-impedance state causes the microprocessor to appear as if it has been removed from its socket. This state allows external I/O devices or other microprocessors to gain access to the system buses so that memory can be accessed directly.

As the timing diagram indicates, HOLD is sampled in the middle of any clocking cycle. Thus, the hold can take effect any time during the operation of any instruction in the microprocessor's instruction set. As soon as the microprocessor recognizes the hold, it stops executing software and enters hold cycles. Note that the HOLD input has a higher priority than the INTR or NMI interrupt inputs. Interrupts take effect at the end of an instruction, while a HOLD takes effect in the middle of an instruction. The only microprocessor pin that has a higher priority than a HOLD is the RESET pin. Note that the HOLD input may not be active during a RESET or the reset is not guaranteed.

The HLDA signal becomes active to indicate that the microprocessor has indeed placed its buses at their high-impedance state, as can be seen in the timing diagram. Note that there are a few clock cycles between the time that HOLD changes and until HLDA changes. The HLDA output is a signal to the external requesting device that the microprocessor has relinquished control of its memory and I/O space. You could call the HOLD input a DMA request input and the HLDA output a DMA grant signal.

Basic DMA Definitions

Direct memory accesses normally occur between an I/O device and memory without the use of the microprocessor. A **DMA read** transfers data from the memory to the I/O device. A **DMA write** transfers data from an I/O device to memory. In both operations, the memory and I/O are controlled simultaneously, which is why the system contains separate memory and I/O control signals. This special control bus structure of the microprocessor allows DMA transfers. A DMA read causes both the MRDC and IOWC signals to simultaneously activate, transferring data from the memory to the I/O device. A DMA write causes the MWTC and IORC signals to both activate. These control bus signals are available to all microprocessors in the Intel family except the 8086/8088 system. The 8086/8088 require their generation with either a system controller or a circuit such as the one illustrated in Figure 12–2. The DMA controller provides the memory with its address and a signal from the controller (DACK) selects the I/O device during the DMA transfer.

The data transfer speed is determined by the speed of the memory device or a DMA controller that often controls DMA transfers. If the memory speed is 100 ns, DMA transfers occur at rates of up to 1/100 ns or 10 Mbytes per second. If the DMA controller in a system functions at a maximum rate of 5 MHz and we still use 100 ns memory, the maximum transfer rate is 5 MHz because the DMA controller is slower than the memory. In many cases, the DMA controller slows the speed of the system when DMA transfers occur.

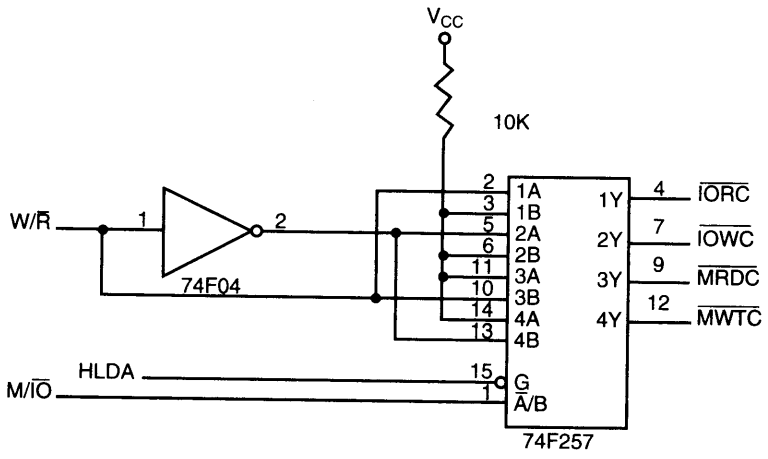


FIGURE 12–2 A circuit that generates system control signals in a DMA environment.

12–2 THE 8237 DMA CONTROLLER

The 8237 DMA controller supplies the memory and I/O with control signals and memory address information during the DMA transfer. The 8237 is actually a special-purpose microprocessor whose job is high-speed data transfer between memory and the I/O. Figure 12–3 shows the pin-out and block diagram of the 8237 programmable DMA controller. Although this device may not appear as a discrete component in modern microprocessor-based systems, it does appear within system controller chip-sets found in most systems. Although not described because of its complexity, the chip set (82357 ISP or integrated system peripheral controller) and its integral set of two DMA controllers are programmed exactly as the 8237. The ISP also provides a pair of 8259A programmable interrupt controllers for the system.

The 8237 is a four-channel device that is compatible with the 8086/8088 microprocessors. The 8237 can be expanded to include any number of DMA channel inputs, although four channels seem to be adequate for many small systems. The 8237 is capable of DMA transfers at rates of up to 1.6M bytes per second. Each channel is capable of addressing a full 64K-byte section of memory and can transfer up to 64K bytes with a single programming.

Pin Definitions

CLK The **clock** input is connected to the system clock signal as long as that signal is 5 MHz or less. In the 8086/8088 system, the clock must be inverted for the proper operation of the 8237.

CS **Chip select** enables the 8237 for programming. The CS pin is normally connected to the output of a decoder. The decoder does not use the 8086/8088 control signal IO/M (M/IO) because it contains the new memory and I/O control signals (MEMR, MEMW, IOR, and IOW).

RESET The **reset** pin clears the command, status, request, and temporary registers. It also clears the first/last flip-flop and sets the mask register. This input primes the 8237 so it is disabled until programmed otherwise.

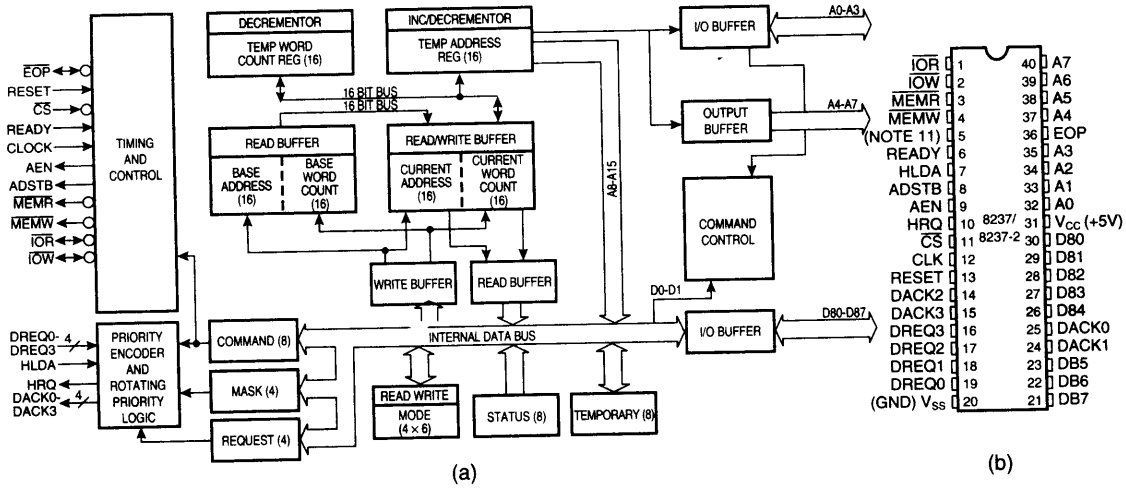


FIGURE 12–3 The 8237A-5 programmable DMA controller. (a) Block diagram and (b) pin-out. (Courtesy of Intel Corporation.)

READY A logic 0 on the **ready** input causes the 8237 to enter wait states for slower memory and/or I/O components.

HLDA A **hold acknowledge** signals the 8237 that the microprocessor has relinquished control of the address, data, and control buses.

DREQ3–DREQ0 The **DMA request** inputs are used to request a DMA transfer for each of the four DMA channels. Because the polarity of these inputs is programmable, they are either active-high or active-low inputs.

DB7–DB0 The **data bus** pins are connected to the microprocessor data bus connections and are used during the programming of the DMA controller.

IOR **I/O read** is a bi-directional pin used during programming and during a DMA write cycle.

IOW **I/O write** is a bi-directional pin used during programming and during a DMA read cycle.

EOP **End-of-process** is a bi-directional signal that is used as an input to terminate a DMA process or as an output to signal the end of the DMA transfer. This input is often used to interrupt a DMA transfer at the end of a DMA cycle.

A3–A0 These **address pins** select an internal register during programming and also provide part of the DMA transfer address during a DMA action.

A7–A4 These **address pins** are outputs that provide part of the DMA transfer address during a DMA action.

HRQ **Hold request** is an output that connects to the HOLD input of the microprocessor in order to request a DMA transfer.

DACK3–DACK0 **DMA channel acknowledge** outputs acknowledge a channel DMA request. These outputs are programmable as either active-high or active-low signals. The DACK outputs are often used to select the DMA controlled I/O device during the DMA transfer.

AEN The **address enable** signal enables the DMA address latch connected to the DB7–DB0 pins on the 8237. It is also used to disable any buffers in the system connected to the microprocessor.

ADSTB **Address strobe** functions as ALE, except that it is used by the DMA controller to latch address bits A15–A8 during the DMA transfer.

MEMR **Memory read** is an output that causes memory to read data during a DMA read cycle.

MEMW **Memory write** is an output that causes memory to write data during a DMA write cycle.

Internal Registers

CAR The **current address register** is used to hold the 16-bit memory address used for the DMA transfer. Each channel has its own current address register for this purpose. When a byte of data is transferred during a DMA operation, the CAR is either incremented or decremented, depending on how it is programmed.

CWCR The **current word count register** programs a channel for the number of bytes (up to 64K) transferred during a DMA action. The number loaded into this register is one less than the number of bytes transferred. For example, if a 10 is loaded into the CWCR, then 11 bytes are transferred during the DMA action.

BA and BWC The **base address** (BA) and **base word count** (BWC) registers are used when auto-initialization is selected for a channel. In the auto-initialization mode, these registers are used to reload both the CAR and CWCR after the DMA action is completed. This allows the same count and address to be used to transfer data from the same memory area.

CR The **command register** programs the operation of the 8237 DMA controller. Figure 12–4 depicts the function of the command register.

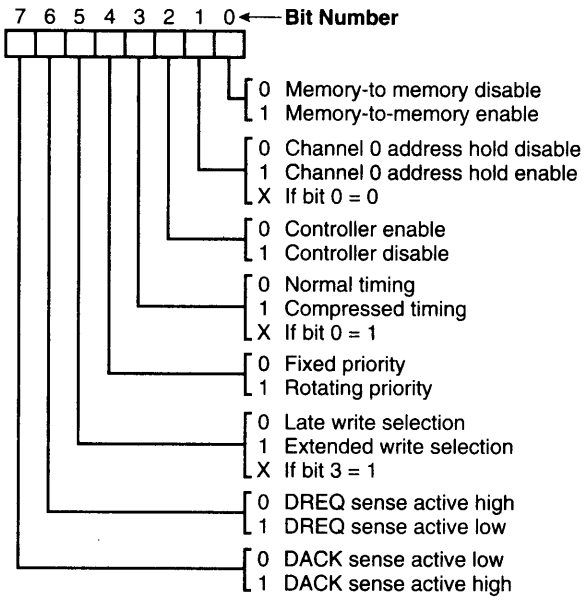


FIGURE 12–4 8237A-5 command register. (Courtesy of Intel Corporation.)

The command register uses bit position 0 to select the memory-to-memory DMA transfer mode. Memory-to-memory DMA transfers use DMA channel 0 to hold the source address and DMA channel 1 to hold the destination address. (This is similar to the operation of a MOVSB instruction.) A byte is read from the address accessed by channel 0 and saved within the 8237 in a temporary holding register. Next, the 8237 initiates a memory write cycle, in which the contents of the temporary holding register are written into the address selected by DMA channel 1. The number of bytes transferred is determined by the channel 1 count register.

The channel 0 address hold enable bit (bit position 1) programs channel 0 for memory-to-memory transfers. For example, if you must fill an area of memory with data, channel 0 can be held at the same address while channel 1 changes for memory-to-memory transfer. This copies the contents of the address accessed by channel 0 into a block of memory accessed by channel 1.

The controller enable/disable bit (bit position 2) turns the entire controller on and off. The normal and compressed bit (bit position 3) determine whether a DMA cycle contains 2 (compressed) or 4 (normal) clocking periods. Bit position 5 is used in normal timing to extend the write pulse so it appears one clock earlier in the timing for I/O devices that require a wider write pulse.

Bit position 4 selects priority for the four DMA channel DREQ inputs. In the fixed priority scheme, channel 0 has the highest priority and channel 3 has the lowest. In the rotating priority scheme, the most recently serviced channel assumes the lowest priority. For example, if channel 2 just had access to a DMA transfer, it assumes the lowest priority and channel 3 assumes the highest priority position. Rotating priority is an attempt to give all channels equal priority.

The remaining two bits (bit positions 6 and 7) program the polarities of the DREQ inputs and the DACK outputs.

MR The **mode register** programs the mode of operation for a channel. Note that each channel has its own mode register (see Figure 12–5), as selected by bit positions 1 and 0. The remaining bits of the mode register select the operation, auto-initialization, increment/decrement, and mode for the channel. Verification operations generate the DMA addresses without generating the DMA memory and I/O control signals.

The modes of operation include demand mode, single mode, block mode, and cascade mode. Demand mode transfers data until an external EOP is input or until the DREQ input becomes inactive. Single mode releases the HOLD after each byte of data is transferred. If the DREQ pin is held active, the 8237 again requests a DMA transfer through the DRQ line to the microprocessor's HOLD input. Block mode automatically transfers the number of bytes indicated by the count register for the channel. DREQ need not be held active through the block mode transfer. Cascade mode is used when more than one 8237 is present in a system.

RR The **request register** is used to request a DMA transfer via software (see Figure 12–6). This is very useful in memory-to-memory transfers, where an external signal is not available to begin the DMA transfer.

MRSR The **mask register set/reset** sets or clears the channel mask, as illustrated in Figure 12–7. If the mask is set, the channel is disabled. Recall that the RESET signal sets all channel masks to disable them.

MSR The **mask register** (see Figure 12–8) clears or sets all of the masks with one command instead of individual channels, as with the MRSR.

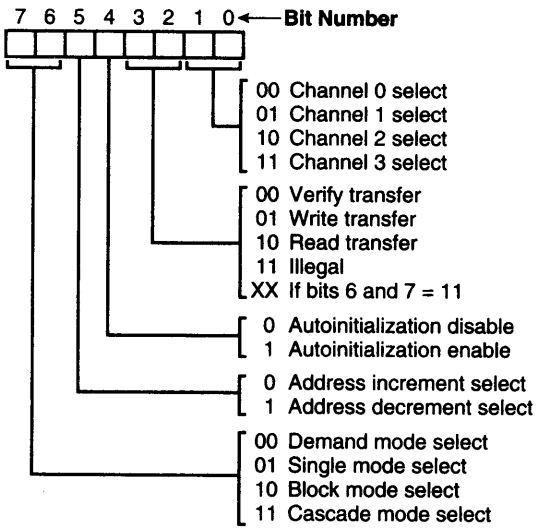


FIGURE 12–5 8237A-5 mode register (Courtesy of Intel Corporation).

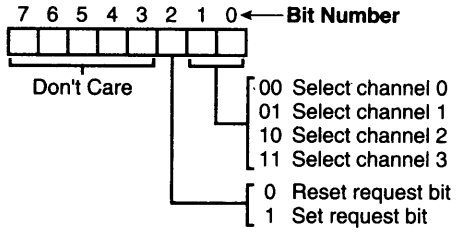


FIGURE 12–6 8237A-5 request register. (Courtesy of Intel Corporation.)

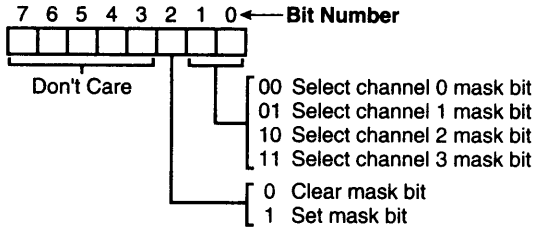


FIGURE 12–7 8237A-5 mask register set/reset mode. (Courtesy of Intel Corporation.)

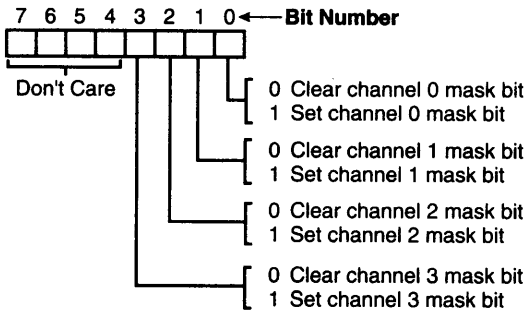


FIGURE 12–8 8237A-5 mask register. (Courtesy of Intel Corporation.)

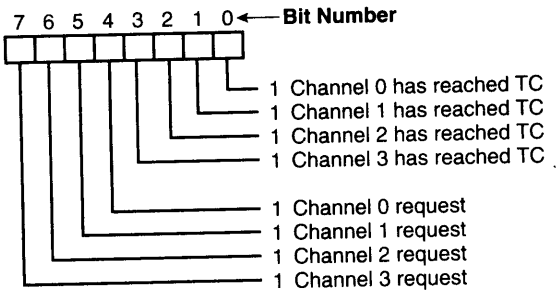


FIGURE 12–9 8237A-5 status register. (Courtesy of Intel Corporation.)

Signals						Operation
A3	A2	A1	A0	$\overline{IOR}$	$\overline{IOW}$	
1	0	0	0	0	1	Read Status Register
1	0	0	0	1	0	Write Command Register
1	0	0	1	0	1	Illegal
1	0	0	1	1	0	Write Request Register
1	0	1	0	0	1	Illegal
1	0	1	0	1	0	Write Single Mask Register Bit
1	0	1	1	0	1	Illegal
1	0	1	1	1	0	Write Mode Register
1	1	0	0	0	1	Illegal
1	1	0	0	1	0	Clear Byte Pointer Flip/Flop
1	1	0	1	0	1	Read Temporary Register
1	1	0	1	1	0	Master Clear
1	1	1	0	0	1	Illegal
1	1	1	0	1	0	Clear Mask Register
1	1	1	1	0	1	Illegal
1	1	1	1	1	0	Write All Mask Register Bits

FIGURE 12–10 8237A-5 command and control port assignments. (Courtesy of Intel Corporation.)

SR The **status register** shows the status of each DMA channel (see Figure 12–9). The TC bits indicate whether the channel has reached its terminal count (transferred all its bytes). Whenever the terminal count is reached, the DMA transfer is terminated for most modes of operation. The request bits indicate whether the DREQ input for a given channel is active.

Software Command

Three software commands are used to control the operation of the 8237. These commands do not have a binary bit pattern, as do the various control registers within the 8237. A simple output to the correct port number enables the software command. Figure 12–10 shows the I/O port assignments that access all registers and the software commands.

The function of the software commands are explained in the following list:

1. **Clear the first/last flip-flop**—Clears the first/last (F/L) flip-flop within the 8237. The F/L flip-flop selects which byte (low or high order) is read/written in the current address and current count registers. If F/L = 0, the low order byte is selected; if F/L = 1, the high order byte is selected. Any read or write to the address or count register automatically toggles the F/L flip-flop.
2. **Master clear**—Acts exactly the same as the RESET signal to the 8237. As with the RESET signal, this command disables all channels.
3. **Clear mask register**—Enables all four DMA channels.

Programming the Address and Count Registers

Figure 12–11 illustrates the I/O port locations for programming the count and address registers for each channel. Notice that the state of the F/L flip-flop determines whether the LSB or MSB is programmed. If the state of the F/L flip-flop is unknown, the count and address could be programmed incorrectly. It is also important that the DMA channel be disabled before its address and count are programmed.

There are four steps required to program the 8237: (1) the F/L flip-flop is cleared using a clear F/L command, (2) the channel is disabled, (3) the LSB and then MSB of the address are programmed, and (4) the LSB and MSB of the count are programmed. Once these four operations are performed, the channel is programmed and

Channel	Register	Operation	Signals							Internal Flip-Flop	Data Bus DB0-DB7
			$\overline{CS}$	$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0		
0	Base and Current Address	Write	0	1	0	0	0	0	0	0	A0-A7
			0	1	0	0	0	0	0	1	A8-A15
	Current Address	Read	0	0	1	0	0	0	0	0	A0-A7
			0	0	1	0	0	0	0	1	A8-A15
	Base and Current Word Count	Write	0	1	0	0	0	0	1	0	W0-W7
			0	1	0	0	0	0	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	0	0	1	0	W0-W7
			0	0	1	0	0	0	1	1	W8-W15
1	Base and Current Address	Write	0	1	0	0	0	1	0	0	A0-A7
			0	1	0	0	0	1	0	1	A8-A15
	Current Address	Read	0	0	1	0	0	1	0	0	A0-A7
			0	0	1	0	0	1	0	1	A8-A15
	Base and Current Word Count	Write	0	1	0	0	0	1	1	0	W0-W7
			0	1	0	0	0	1	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	0	1	1	0	W0-W7
			0	0	1	0	0	1	1	1	W8-W15
2	Base and Current Address	Write	0	1	0	0	1	0	0	0	A0-A7
			0	1	0	0	1	0	0	1	A8-A15
	Current Address	Read	0	0	1	0	1	0	0	0	A0-A7
			0	0	1	0	1	0	0	1	A8-A15
	Base and Current Word Count	Write	0	1	0	0	1	0	1	0	W0-W7
			0	1	0	0	1	0	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	1	0	1	0	W0-W7
			0	0	1	0	1	0	1	1	W8-W15
3	Base and Current Address	Write	0	1	0	0	1	1	0	0	A0-A7
			0	1	0	0	1	1	0	1	A8-A15
	Current Address	Read	0	0	1	0	1	1	0	0	A0-A7
			0	0	1	0	1	1	0	1	A8-A15
	Base and Current Word Count	Write	0	1	0	0	1	1	1	0	W0-W7
			0	1	0	0	1	1	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	1	1	1	0	W0-W7
			0	0	1	0	1	1	1	1	W8-W15

FIGURE 12–11 8237A-5 DMA channel I/O port addresses. (Courtesy of Intel Corporation.)

ready to use. Additional programming is required to select the mode of operation before the channel is enabled and started.

The 8237 Connected to the 80X86 Microprocessor

Figure 12–12 shows an 80X86-based system that contains the 8237 DMA controller.

The address enable (AEN) output of the 8237 controls the output pins of the latches and the outputs of the 74LS257 (E). During normal 80X86 operation (AEN = 0), latches A and C and the multiplexer (E) provide address bus bits A19–A16 and A7–A0. The multiplexer provides the system control signals as long as the 80X86 is in control of the system. During a DMA action (AEN = 1), latches A and C are disabled along with the multiplexer (E). Latches D and B now provide address bits A19–A16 and A15–A8. Address bus bits A7–A0 are provided directly by the 8237 and contain a part of the DMA transfer address. The control signals MEMR, MEMW, IOR, and IOW are provided by the DMA controller.

The address strobe output (ADSTB) of the 8237 clocks the address (A15–A8) into latch D during the DMA action so that the entire DMA transfer address becomes available on the address bus. Address bus bits A19–A16 are provided by latch B, which must be programmed with these four address bits before the controller is enabled for the DMA transfer. The DMA operation of the 8237 is limited to a transfer of not more than 64K bytes within the same 64K-byte section of the memory.

TABLE 12-1 DMA page register ports.

Channel	*Port number (A16–A23)*	*Port number (A24–A31)*
0	87H	487H
1	83H	483H
2	81H	481H
3	82H	482H
4	8FH	48FH
5	8BH	48BH
6	89H	489H
7	8AH	48AH

The decoder (F) selects the 8237 for programming and the 4-bit latch (B) for the uppermost four address bits. The latch in a PC is called the DMA page register (8-bits) that holds address bits A16–A23 for a DMA transfer. A high page register also exists, but its address is chip-dependent. The port numbers for the DMA page registers are listed in Table 12–1.The decoder in this system enables the 8237 for I/O port addresses XX60H–XX7FH, and the I/O latch (B) for ports XX00H–XX1FH. Notice that the decoder output is combined with the IOW signal to generate an active-high clock for the latch (B).

During normal 80X86 operation, the DMA controller and integrated circuits B and D are disabled. During a DMA action, integrated circuits A, C, and E are disabled so that the 8237 can take control of the system through the address, data, and control buses.

In the personal computer, the two DMA controllers are programmed at I/O ports 0000H–000FH for DMA channels 0–3, and at ports 00C0H–00DFH for DMA channels 4–7. Note that the second controller is programmed at even addresses only, so the channel 4 base and current address is programmed at I/O port 00C0H and the channel 4 base and current count is programmed at port 00C2H. The page register, which holds address bits A23–A16 of the DMA address, are located at I/O ports 0087H (CH–0), 0083H (CH–1), 0081H (CH–2), 0082H (CH–3), (no channel 4), 008BH (CH–5), 0089H (CH–6) and 008AH (CH–7). The page register functions as the address latch described with the examples in this text.

Memory-to-Memory Transfer with the 8237

The memory-to-memory transfer is much more powerful than even the automatically repeated MOVSB instruction. While the repeated MOVSB instruction tables the 8088 4.2 μs per byte, the 8237 requires only 2.0 μs per byte, which is over twice as fast as a software data transfer. This is not true if an 80386, 80846, or Pentium through Pentium 4 is in use in the system.

Sample Memory-to-Memory DMA Transfer. Suppose that the contents of memory locations 10000H–13FFFH are to be transferred into memory locations 14000H–17FFFH. This is accomplished with a repeated string move instruction, or, at a much faster rate, with the DMA controller.

Example 12–1 illustrates the software required to initialize the 8237 and program latch B in Figure 12–12 for this DMA transfer. This software is written for an embedded application. For it to function in the PC, you must use the port addresses listed in Table 12–1 for the page registers.

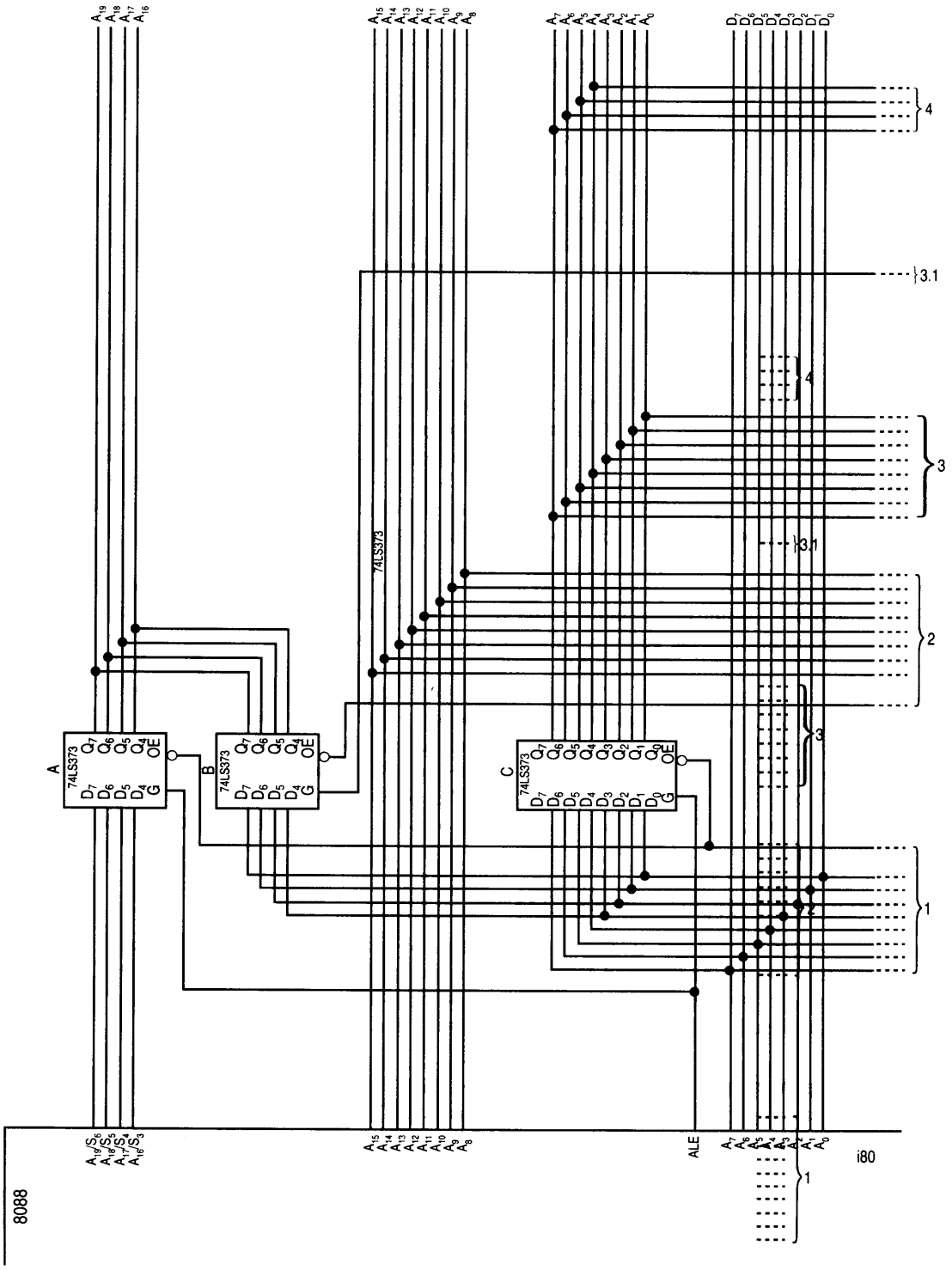
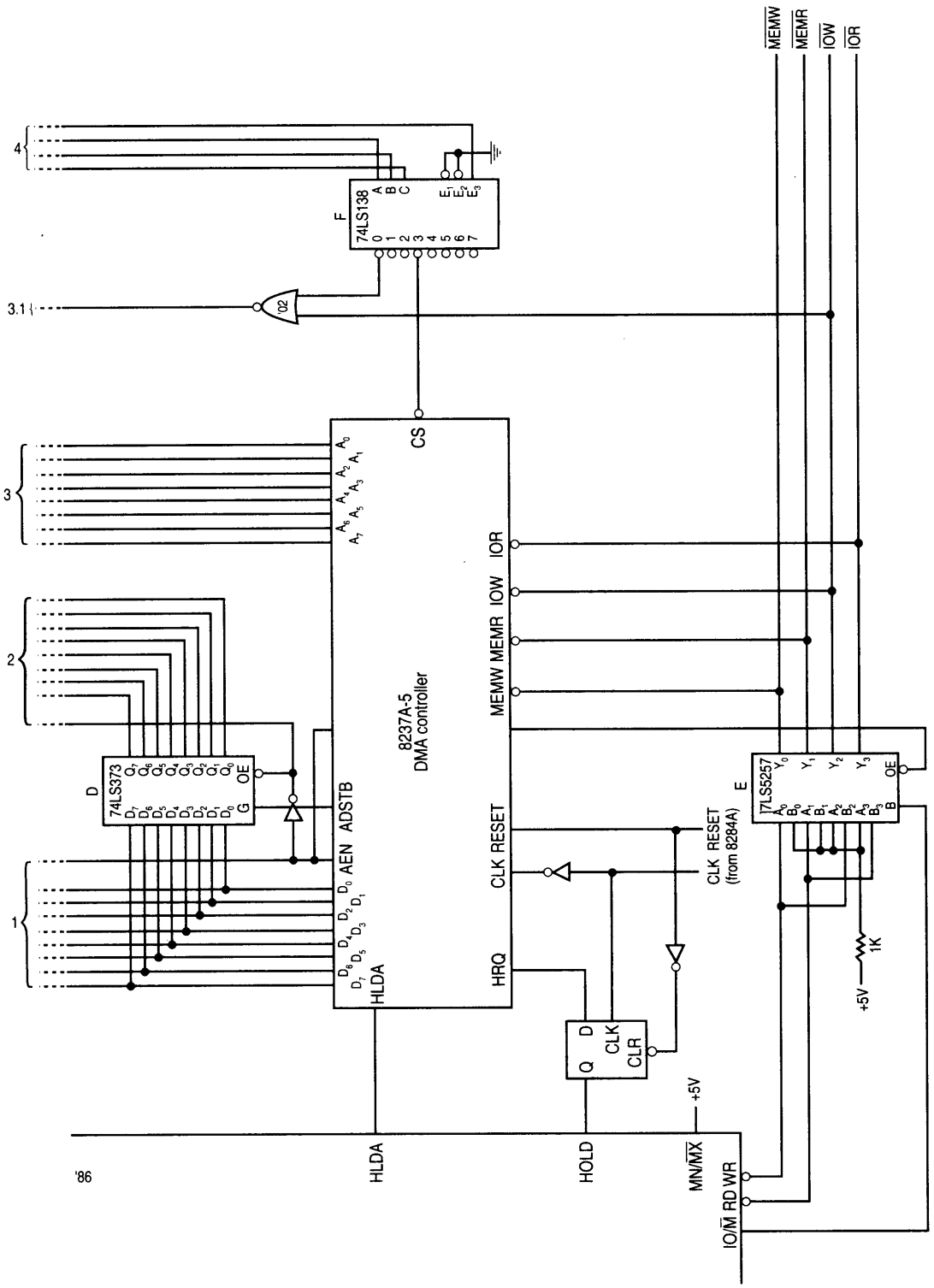


FIGURE 12-12 Complete 8088 minimum mode DMA system.


MEMW
MEMR
IOW
IOR
4
F
74LS138
A
B
C
E1
E2
E3
0
1
2
3
4
5
6
7
3.1
'02
CS
3
A0
A1
A2
A3
A4
A5
A6
A7
IOR
IOW
MEMW MEMR
8237A-5
DMA controller
2
D
74LS373
OE
G
ADSTB
AEN
1
D0
D1
D2
D3
D4
D5
D6
D7
HLDA
CLK RESET
HRQ
E
74LS257
OE
CLK RESET
(from 8284A)
1K
+5V
Q
D
CLK
CLR
+5V
'86
HLDA
HOLD
MN/MX
IO/M RD WR

EXAMPLE 12–1

```
                ;A procedure that transfers a block of data using the
                ;8237A DMA controller in Figure 12-12. This is a
                ;memory-to-memory block transfer.
                ;
                ;Calling parameters:
                ;    SI = source address
                ;    DI = destination address
                ;    CX = count
                ;    ES = segment of source and destination
;
= 0010              LATCHB  EQU    10H         ;latch B
= 007C              CLEAR_F EQU    7CH         ;F/L flip flop
= 0070              CH0_A   EQU    70H         ;channel 0 address
= 0072              CH1_A   EQU    72H         ;channel 1 address
= 0073              CH1_C   EQU    73H         ;channel 1 count
= 007B              MODE    EQU    7BH         ;mode
= 0078              CMMD    EQU    78H         ;command
= 007F              MASKS   EQU    7FH         ;masks
= 0079              REQ     EQU    79H         ;request register
= 0078              STATUS  EQU    78H         ;status register

0000                TRANS   PROC   FAR USES AX

0001  8C C0                MOV    AX,ES       ;program latch B
0003  8A C4                MOV    AL,AH
0005  C0 E8 04             SHR    AL,4
0008  E6 10                OUT    LATCHB,AL
000A  E6 7C                OUT    CLEAR_F,AL ;clear F/L flip-flop

000C  8C C0                MOV    AX,ES       ;program source address
000E  C1 E0 04             SHL    AX,4
0011  03 C6                ADD    AX,SI       ;form source offset
0013  E6 70                OUT    CH0_A,AL
0015  8A C4                MOV    AL,AH
0017  E6 70                OUT    CH0_A,AL

0019  8C C0                MOV    AX,ES       ;program destination address
001B  C1 E0 04             SHL    AX,4
001E  03 C7                ADD    AX,DI       ;form destination offset
0020  E6 72                OUT    CH1_A,AL
0022  8A C4                MOV    AL,AH
0024  E6 72                OUT    CH1_A,AL

0026  8B C1                MOV    AX,CX       ;program count
0028  48                   DEC    AX          ;adjust count
0029  E6 73                OUT    CH1_C,AL
002B  8A C4                MOV    AL,AH
002D  E6 73                OUT    CH1_C,AL

002F  B0 88                MOV    AL,88H      ;program mode
0031  E6 7B                OUT    MODE,AL
0033  B0 85                MOV    AL,85H
0035  E6 7B                OUT    MODE,AL

0037  B0 01                MOV    AL,1        ;enable block transfer
0039  E6 78                OUT    CMMD,AL

003B  B0 0E                MOV    AL,0EH      ;unmask channel 0
003D  E6 7F                OUT    MASKS,AL

003F  B0 04                MOV    AL,4        ;start DMA transfer
0041  E6 79                OUT    REQ,AL
```