# Week 9

# Interrupt Interface of 8088 and 8086 processors,
# 8259 Interrupt controller

# Interrupt Mechanisms, Types, and Priority

INTERRUPT TYPES SHOWN WITH DECREASING PRIORITY ORDER

- Reset
- Internal interrupts and exceptions
- Software interrupt
- Nonmaskable interrupt
- Hardware interrupt

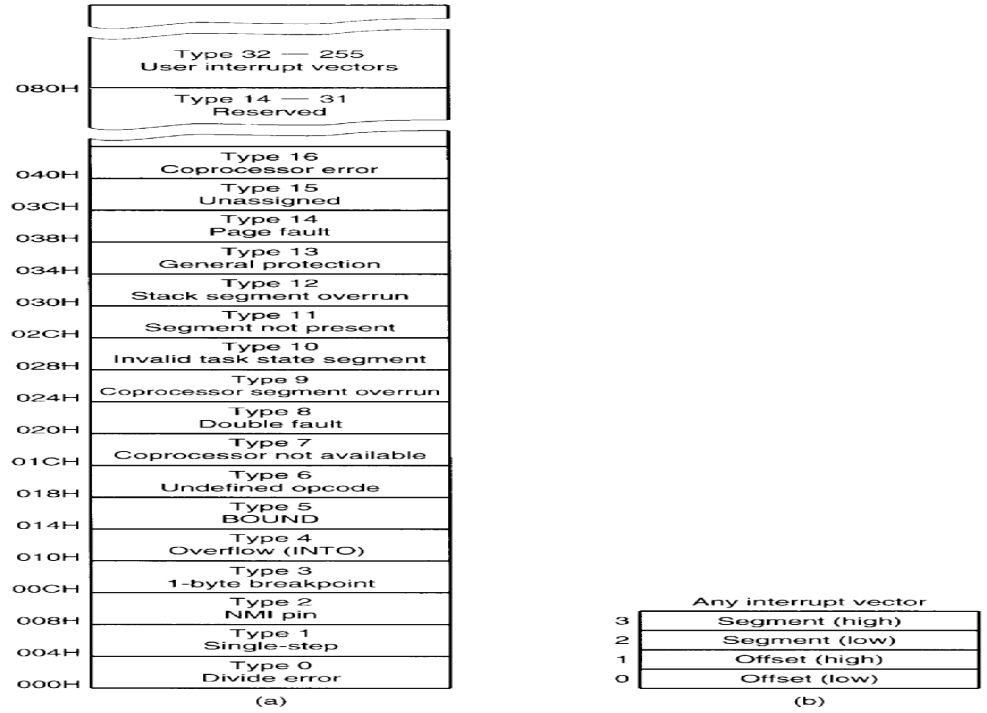
All the interrupts are serviced on priority basis. The higher priority interrupt is served first and an active lower priority interrupt service is interrupted by a higher priority one. Lower priority interrupts will have to wait until their turns come.

The section of program to which the control is passed called **Interrupt-service routine** (e.g. printer driver)

# Interrupt Vector Table

460 CHAPTER 12 INTERRUPTS

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




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

# Interrupt Vector Table

| | |
|---|---|
| **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 |
| **Type 2** | Non-maskable Hardware Interrupt—A result of placing a logic 1 on the NMI input pin to the microprocessor. This input is non-maskable, which means that it cannot be disabled. |
| **Type 3** | One-Byte Interrupt—A special one-byte instruction (INT 3) that uses this vector to access its interrupt-service procedure. The INT 3 instruction is often used to store a breakpoint in a program for debugging. |
| **Type 4** | Overflow—A special vector used with the INTO instruction. The INTO instruction interrupts the program if an overflow condition exists, as reflected by the overflow flag (OF). |
| **Type 5** | BOUND—An instruction that compares a register with boundaries stored in the memory. If the contents of the register are greater than or equal to the first word in memory and less than or equal to the second word, no interrupt occurs because the contents of the register is within bounds. If the contents of the register are out-of-bounds, a type 5 interrupt ensues. |
| **Type 6** | Invalid Opcode—Occurs whenever an undefined opcode is encountered in a program. |
| **Type 7** | Coprocessor Not Available—Occurs when a coprocessor is not found in the system, as dictated by the machine status word (MSW) coprocessor control bits. If an ESC or WAIT instruction executes and the coprocessor is not found, a type 7 exception or interrupt occurs. |
| **Type 8** | Double Fault—Activated whenever two separate interrupts occur during the same instruction. |

# Interrupt Vector Table

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

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

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

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

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

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

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

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

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

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

# Interrupt Vector Table

- Interrupt vector table consists of 256 entries each containing 4 bytes.
- Each entry contains the offset and the segment address of the interrupt vector each 2 bytes long.
- Table starts at the memory address 00000F.
- First 32 vectors are spared for various microprocessor families.
- The rest 224 vectors are user definable.
- **The lower the vector number, the higher the priority.**

# The Operation of Real Mode Interrupt

1. The contents of the flag registers are pushed onto the stack.
2. Both the interrupt (IF) and (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) is pushed onto the stack.
4. The contents of the instruction pointer (IP) is 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 interrupt vector.
6. While returning from the interrupt-service routine by the ins. IRET, flags return to their state prior to the interrupt and and operation restarts at the prior IP address. The return address (CS and IP) is not always the next instruction, with some interrupt types it is the current instruction.

# Example

- At what address should vector 50, CS50, and IP50 be stored in memory?
- Each vector requires four bytes of memory
- Address = 50 x 4 = 200
- Converting to binary
  - 200 = 1100 1000b
  - Address = C8h
- IP50 is stored in 00C8h
- CS50 is stored in 00CAh

# Interrupt Vector Table

**FIGURE 12–3** The protected mode interrupt descriptor.

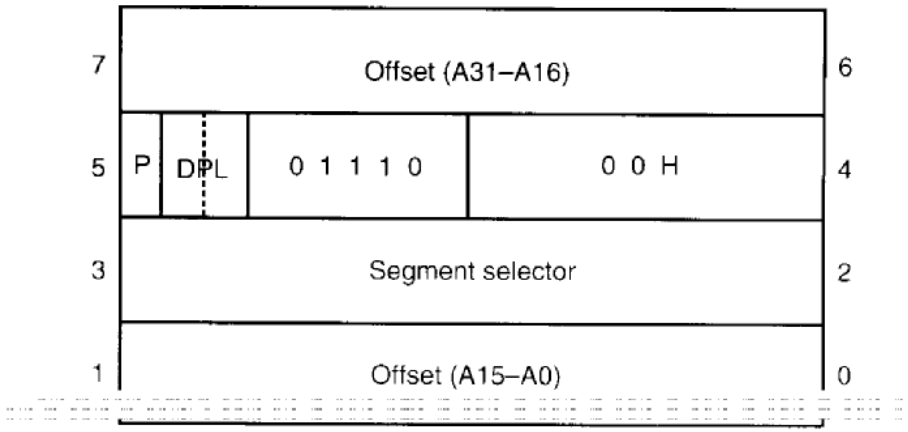
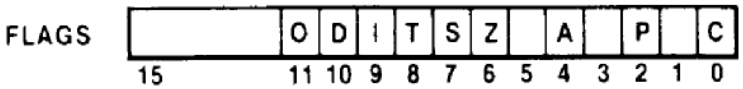



**FIGURE 12–4** The flag register. (Courtesy of Intel Corporation.)



# The Operation of Protected Mode Interrupt

1. Protected mode interrupts function like the real mode interrupts, except that the interrupt vector table is different.
2. It is replaced by the interrupt descriptor table.
3. Each entry is 8 byte long and there are 256 of them similar to the interrupt vectors of the real mode.
4. This table is located by the system at a memory location described by the Interrupt Descriptor Table Address Register (IDTR).

| Mnemonic | Meaning | Format | Operation | Flags Affected |
|---|---|---|---|---|
| CLI | Clear interrupt flag | CLI | 0 → (IF) | IF |
| STI | Set interrupt flag | STI | 1 → (IF) | IF |
| INT n | Type n software interrupt | INT n | (Flags) → ((SP) − 2)<br>0 → TF,IF<br>(CS) → ((SP) − 4)<br>(2 + 4 · n) → (CS)<br>(IP) → ((SP) − 6)<br>(4 · n) → (IP) | TF, IF |
| IRET | Interrupt return | IRET | ((SP)) → (IP)<br>((SP) + 2) → (CS)<br>((SP) + 4) → (Flags)<br>(SP) + 6 → (SP) | All |
| INTO | Interrupt on overflow | INTO | INT 4 steps | TF, IF |
| HLT | Halt | HLT | Wait for an external interrupt or reset to occur | None |
| WAIT | Wait | WAIT | Wait for $\overline{\text{TEST}}$ input to go active | None |

**Figure 11–4** Interrupt instructions.

# Interrupt instructions

- Interrupt enable flag (IF) causes external interrupts to be enabled.
- INT n initiates a vectored call of a subroutine.
- INTO instruction should be used after each arithmethic instruction where there is a possibility of an overflow.
- HLT waits for an interrupt to occur.
- WAIT waits for $\overline{\text{TEST}}$ input to go high.

# External hardware-interrupt Interface

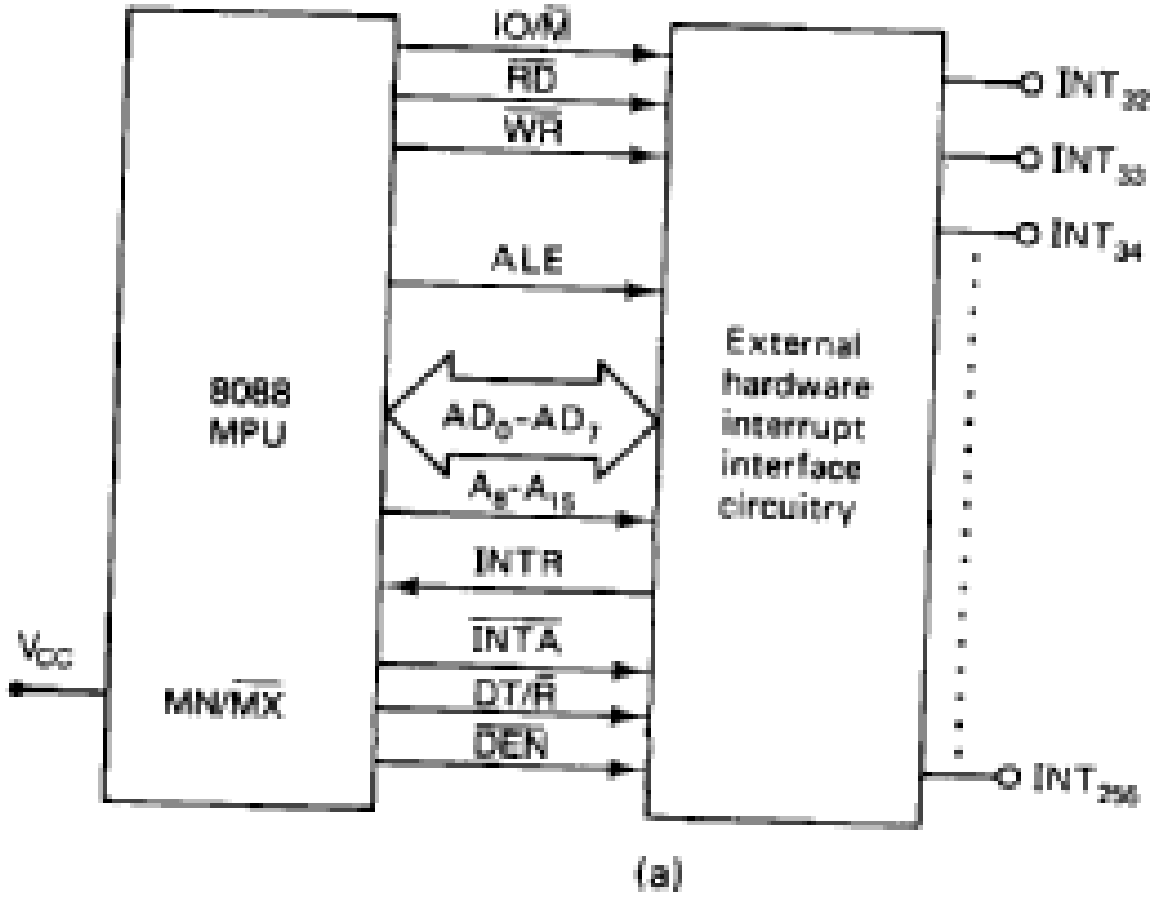



**Figure 11–5** (a) Minimum-mode 8088 system external hardware-interrupt interface. (b) Minimum-mode 8086 system external hardware-interrupt interface.

# External hardware-interrupt Interface

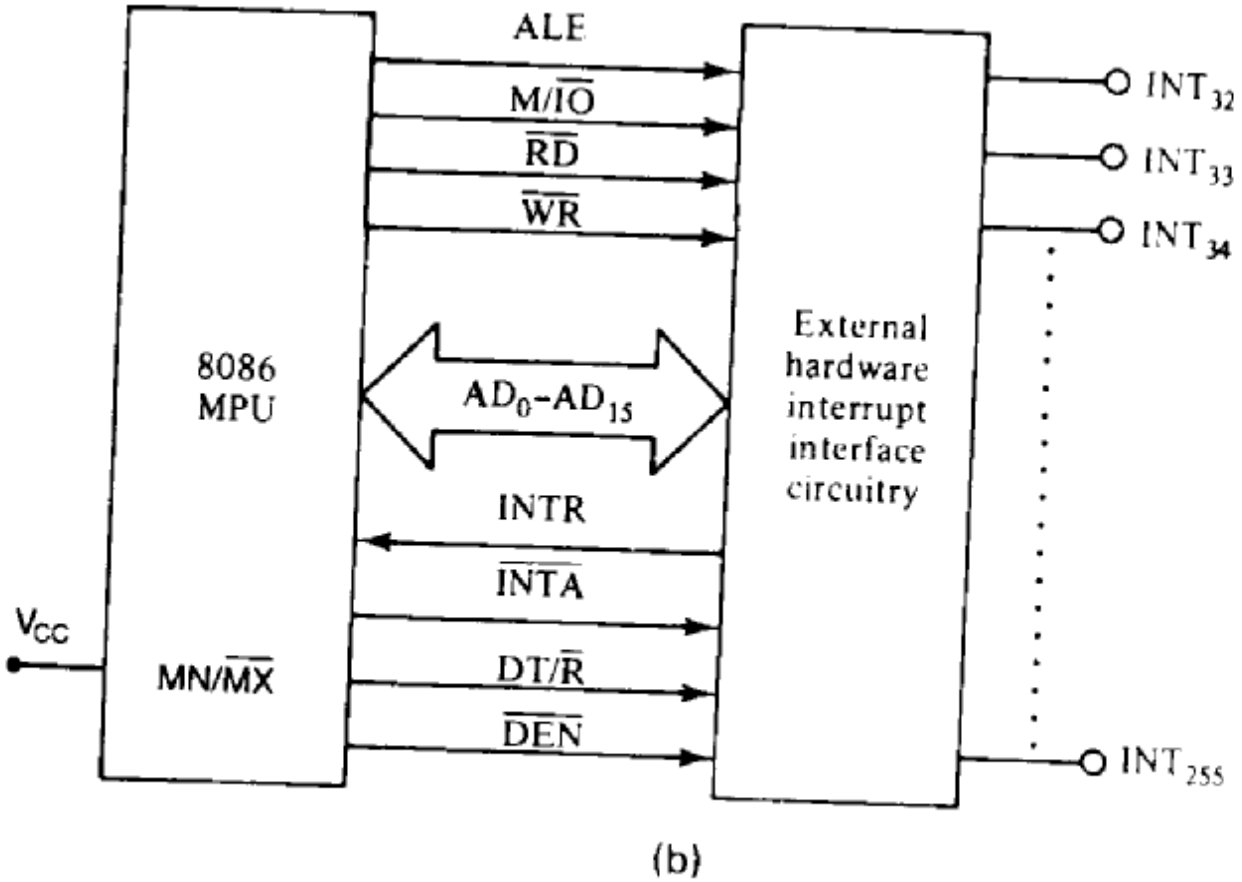



**Figure 11–5** (a) Minimum-mode 8088 system external hardware-interrupt interface. (b) Minimum-mode 8086 system external hardware-interrupt interface.

# External hardware-interrupt Interface

- Minimum mode hardware-interrupt interface:
  - 8088 samples INTR input during the last clock period of each instruction execution cycle. INTR is a level triggered input; therefore logic 1 input must be maintained there until it is sampled. Moreover, it must be removed before it is sampled next time. Otherwise, the same Interrupt Service is repeated twice.
  - $\overline{\text{INTA}}$ goes to 0 in the first interrupt bus cycle to acknowledge the interrupt after it was decided to respond to the interrupt.
  - It goes to 0 again the second bus cycle too, to request for the interrupt type number from the external device.
  - The interrupt type number is read by the processor and the corresponding int. CS and IP numbers are again read from the memory.

# External hardware-interrupt Interface

- Maximum mode hardware-interrupt interface:
  - The operation is similar with some differences.
  - Now bus status codes are used to generate some signals.
  - There is new signal called the bus priority lock signal $\overline{LOCK}$. It is used to signal to the bus arbiter that the bus is busy.

# External hardware-interrupt Interface

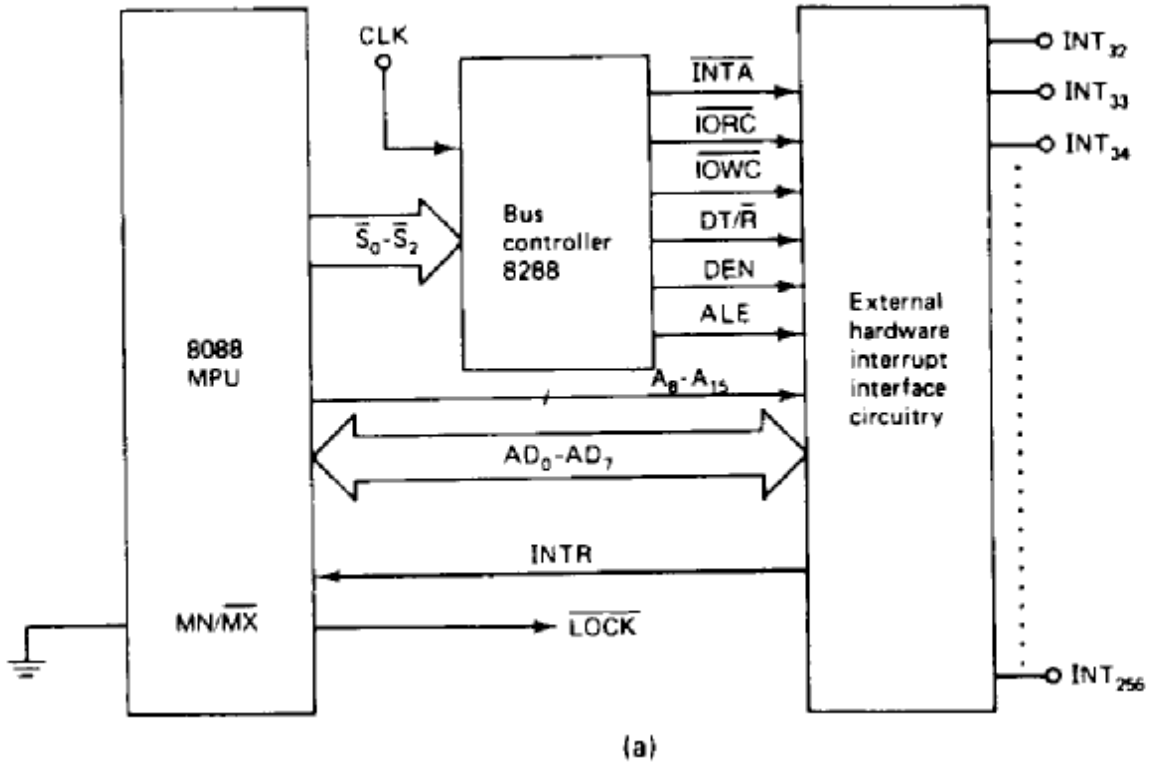



**Figure 11–6** (a) Maximum-mode 8088 system external hardware interrupt interface. (b) Maximum-mode 8086 system external hardware interrupt interface.

# External hardware-interrupt Interface

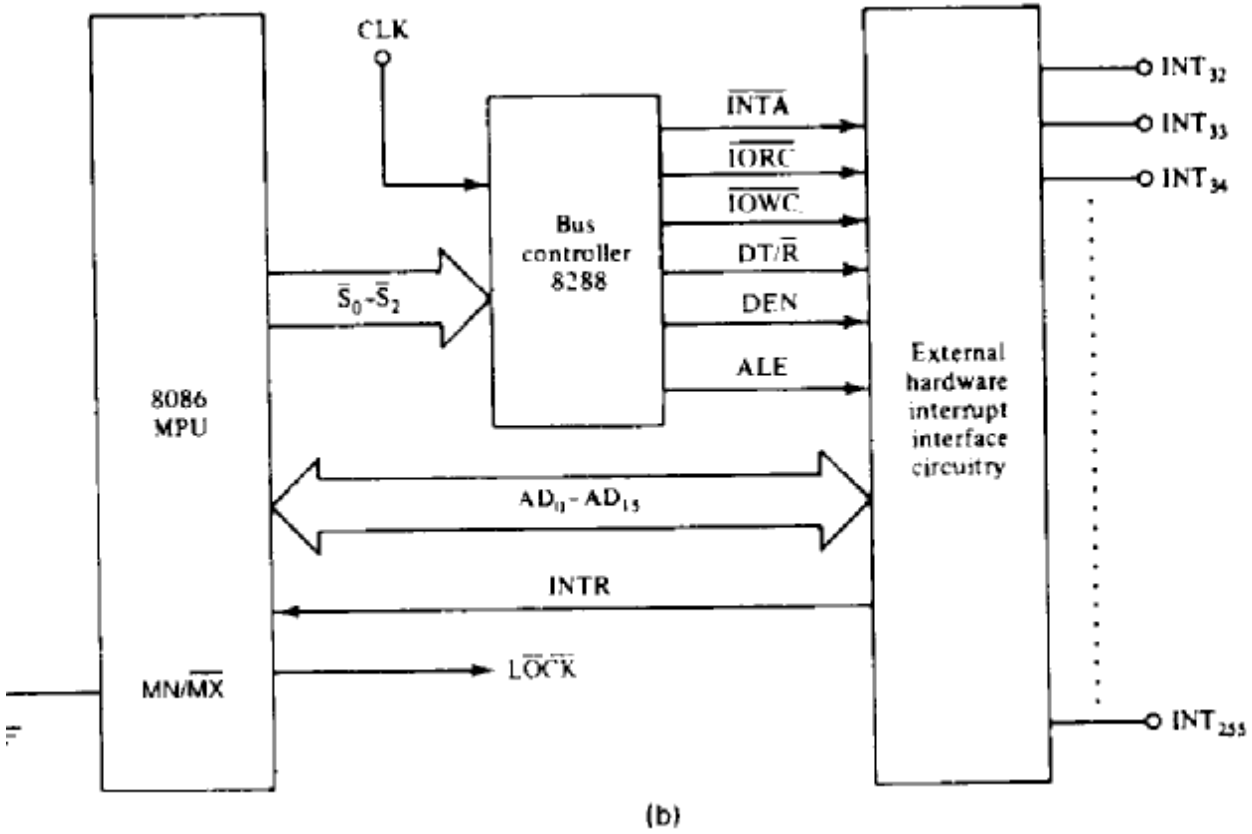



**Figure 11–6** (a) Maximum-mode 8088 system external hardware interrupt interface. (b) Maximum-mode 8086 system external hardware interrupt interface.

# External hardware-interrupt Interface

| Status inputs | | | CPU cycle | 8288 command |
|---|---|---|---|---|
| $\overline{S}_2$ | $\overline{S}_1$ | $\overline{S}_0$ | | |
| 0 | 0 | 0 | Interrupt acknowledge | $\overline{INTA}$ |
| 0 | 0 | 1 | Read I/O port | $\overline{IORC}$ |
| 0 | 1 | 0 | Write I/O port | $\overline{IOWC}$, $\overline{AIOWC}$ |
| 0 | 1 | 1 | Halt | None |
| 1 | 0 | 0 | Instruction fetch | $\overline{MRDC}$ |
| 1 | 0 | 1 | Read memory | $\overline{MRDC}$ |
| 1 | 1 | 0 | Write memory | $\overline{MWTC}$, $\overline{AMWC}$ |
| 1 | 1 | 1 | Passive | None |

**Figure 11–7** Interrupt bus status code. (Reprinted by permission of Intel Corporation. Copyright/Intel Corp. 1979)

# External hardware-interrupt Sequence

- Only after the interrupt processing sequence is carried out, the interrupt acknowledge signal issued upon the result of checks.
- Note that fetching the values of CS and IP, and carrying out PUSH and POP operations take less number of bus cycles in 8086 than 8088 because of 16 bit data bus.

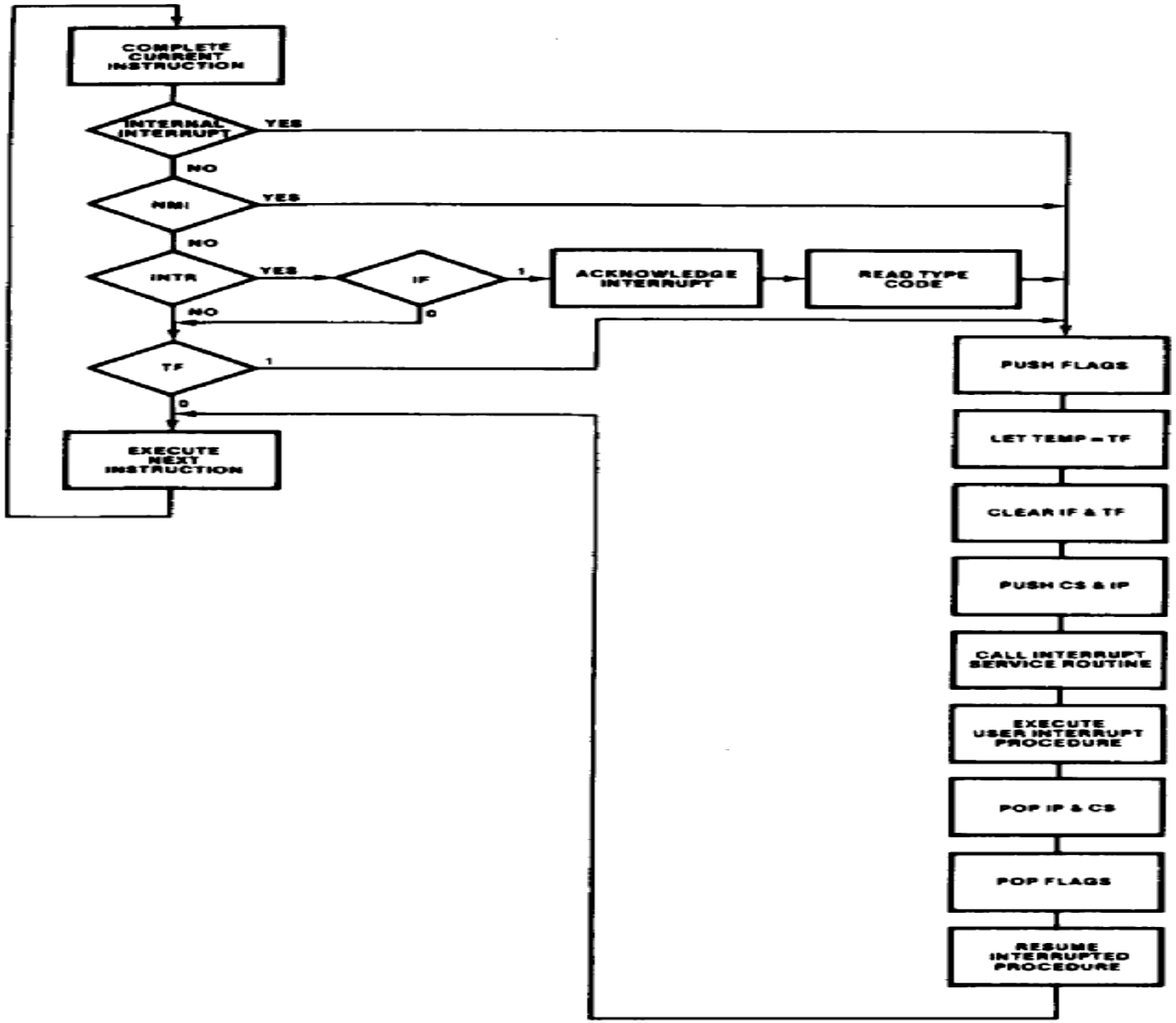



**Figure 11–8** Interrupt processing sequence of the 8088 and 8086 microprocessors. (Reprinted by permission of Intel Corporation. Copyright/Intel Corp. 1979)

# External hardware-interrupt Sequence

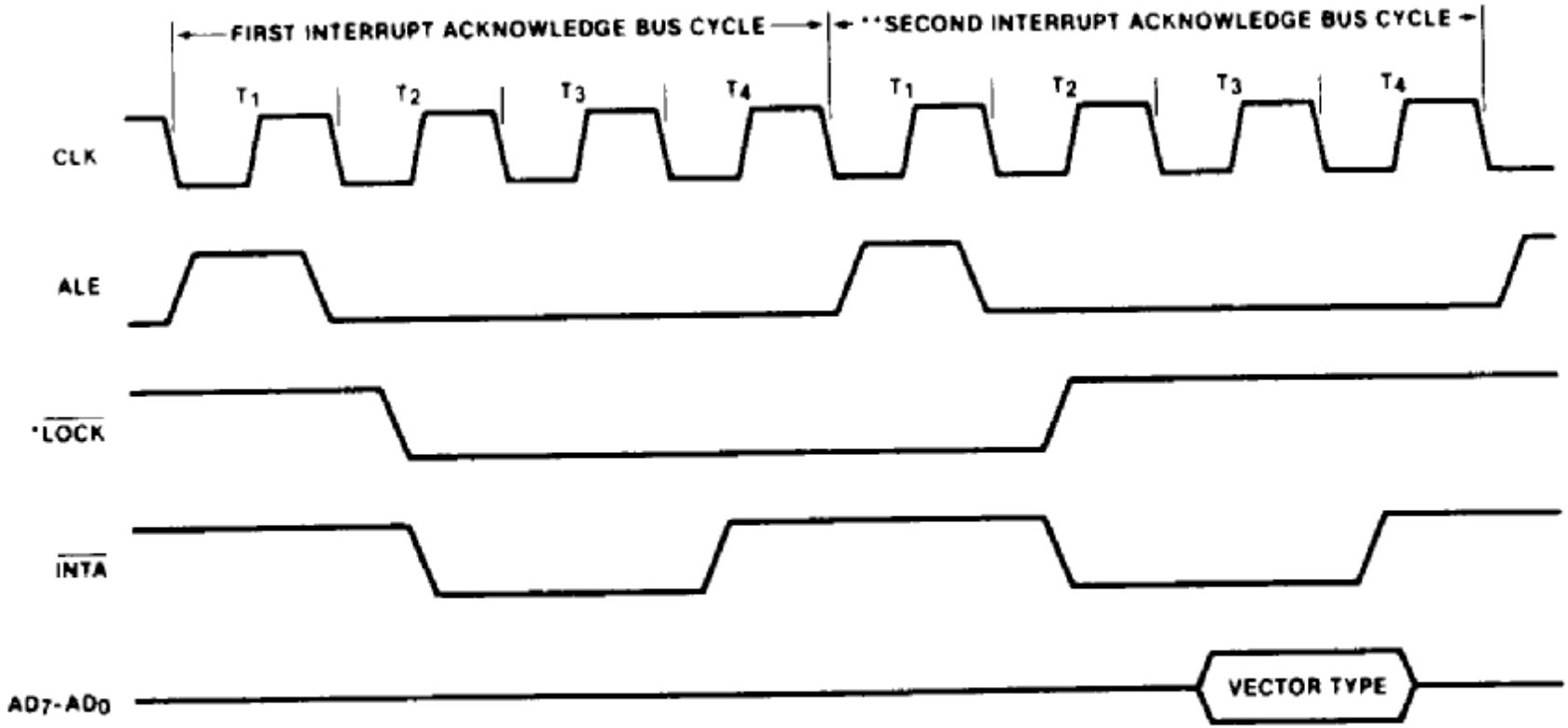



**Figure 11–9** Interrupt-acknowledge bus cycle. (Reprinted by permission of Intel Corporation. Copyright/Intel Corp. 1979)

# 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

INT 21h

<u>Initialization</u>
AH = 25h
AL = interrupt type number
DS:DX = address of new interrupt procedure

<u>Read the current vector</u>
AH = 35h
AL = interrupt type number
ES:BX = address stored at vector

<u>Terminate and stay resident</u>
AH = 31h
AL = 00
DX = number of paragraphs to reserve for the program

# Example-storing Interrupt Vector

## Storing an Interrupt Vector in the Vector Table

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

**EXAMPLE 12–4**

```
                                .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
```

# Example-storing Interrupt Vector

```
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
```

# Example

- An interrupting device interrupts the microprocessor each time the interrupt request input has a transition from 0 to 1.
- 74LS244 creates the interrupt type number 60H as a response to $\overline{\text{INTA}}$.
- Assume:
  - CS=DS=1000H
  - SS=4000H
  - Main program offset is 200H
  - Count offset is 100H
  - Interrupt-service routine code segment is 2000H:0000H
  - Interrupt-service routine code offset is 1000H
  - Stack has an offset of 500H to the current stack segment
  - Make a map of the memory space organisation
  - Write a main program and a service routine to count the number of positive interrupt transitions.

# Example – interrupt request counter

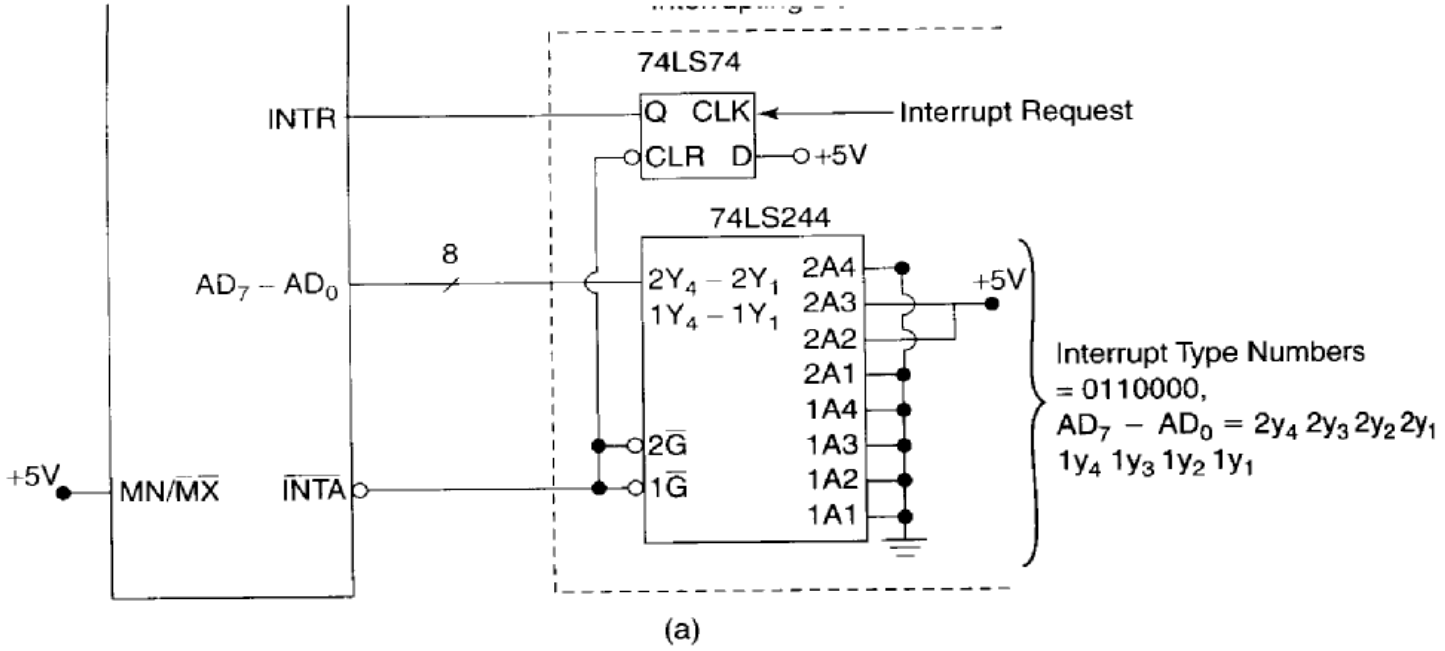
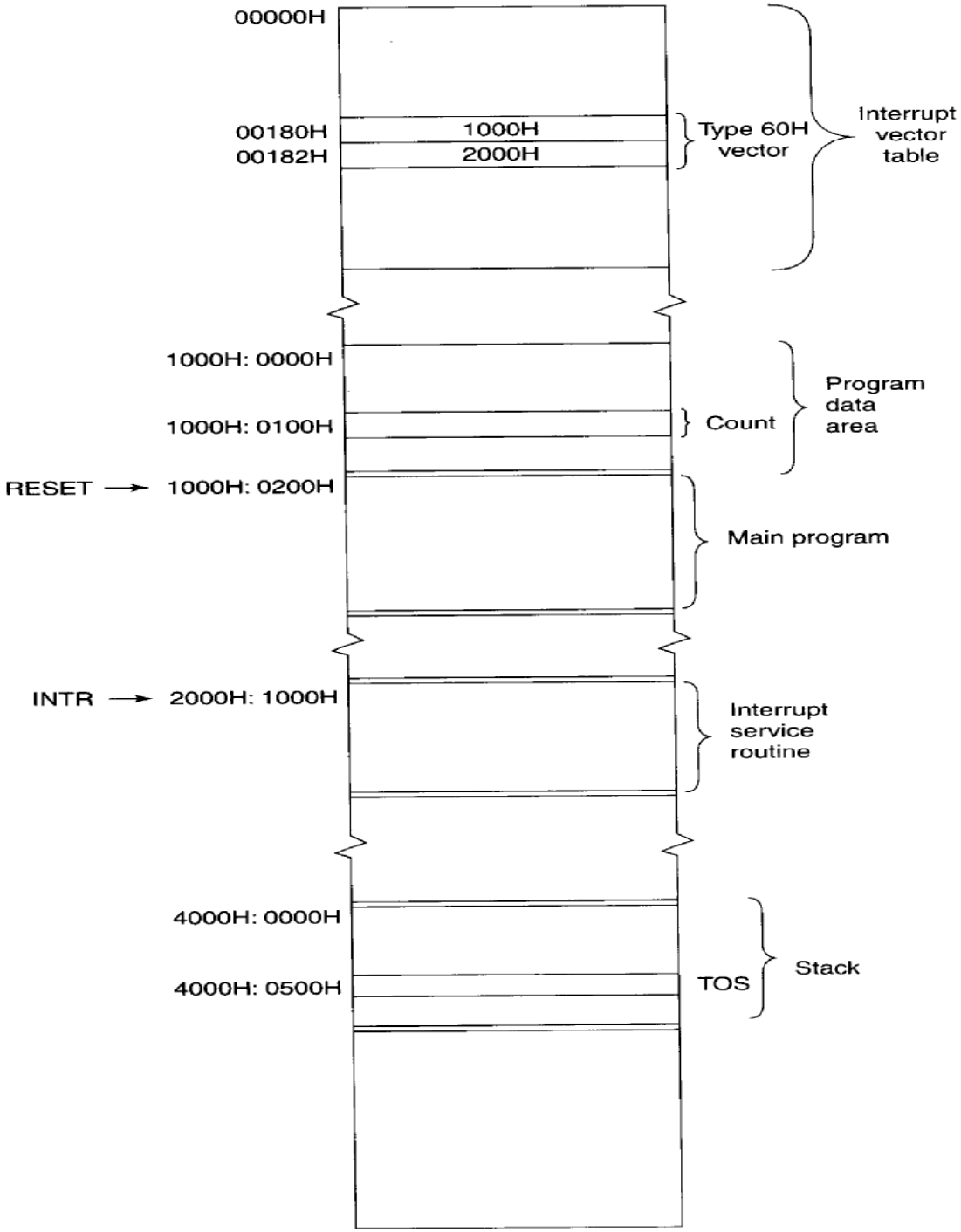
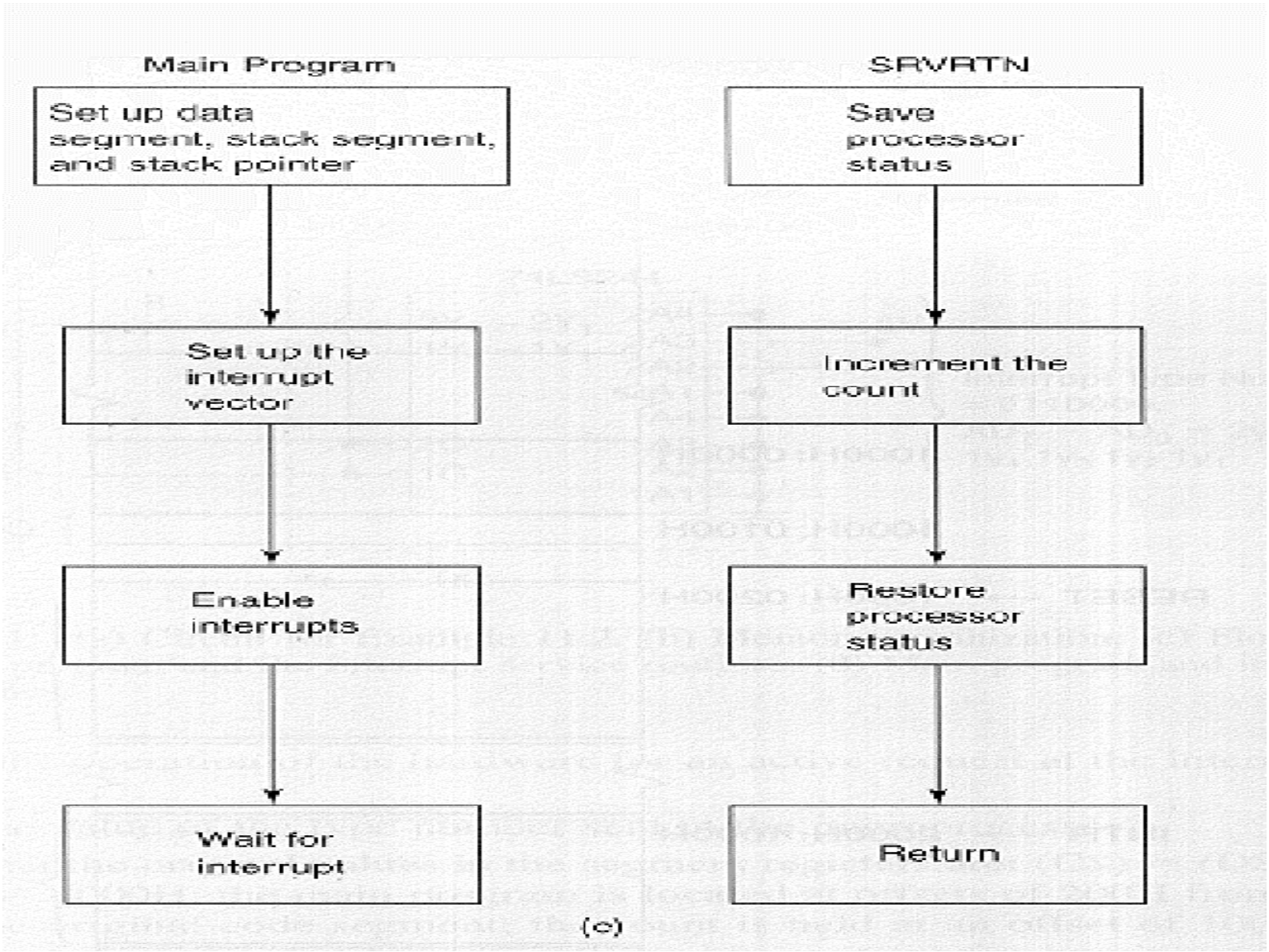



**Figure 11–11** (a) Circuit for Example 11.2. (b) Memory organization. (c) Flowcharts for the main program and the interrupt-service routines. (d) Main program and interrupt-service routines.

# Exa


00000H
00180H
00182H
1000H
2000H
Type 60H vector
Interrupt vector table
1000H: 0000H
1000H: 0100H
Count
Program data area
RESET
1000H: 0200H
Main program
INTR
2000H: 1000H
Interrupt service routine
4000H: 0000H
4000H: 0500H
TOS
Stack

# Example – interrupt request counter



# Example – interrupt request counter

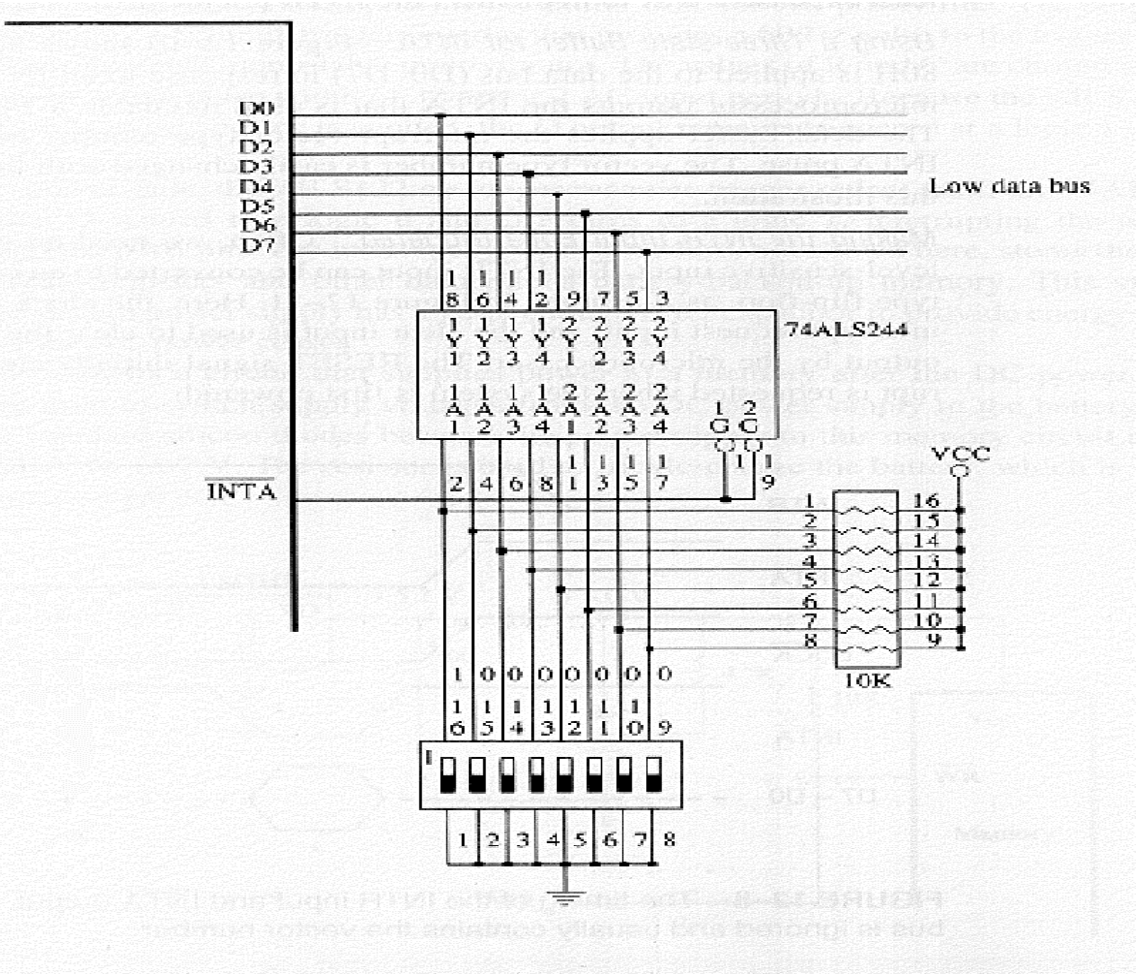
```
; Main program, START = 1000H:0200H
START:      MOV    AX,1000H        ;Set up data seg
            MOV    DX,AX
            MOV    AX,4000H        ;Set up stack seg
            MOV    SS,AX
            MOV    SP,0500H        ;Set  top of stack
            MOV    AX,0000H        ;Seg for int. Vector table
            MOV    ES,AX
            MOV    AX,1000H        ;Service routine offset
            MOV    [ES:180H],AX
            MOV    AX,2000H        ;Service routine seg
            MOV    [ES:182H],AX
            STI                    ;Enable interrupt
HERE:       JMP    HERE            ;Wait for interrupt
```

# Example – interrupt request counter

```
;Interrupt service routine, SRVRTN = 2000H:1000H
SRVRTN:     PUSH AX              ;save reg to be used
            MOV AL,[0100H]       ;get the count
            INC AL               ;increment the count
            DAA                  ;decimal adjust the count
            MOV [0100H],AL       ;save the updated count
            POP AX               ;restore the register used
            IRET                 ;return from the interrupt
```

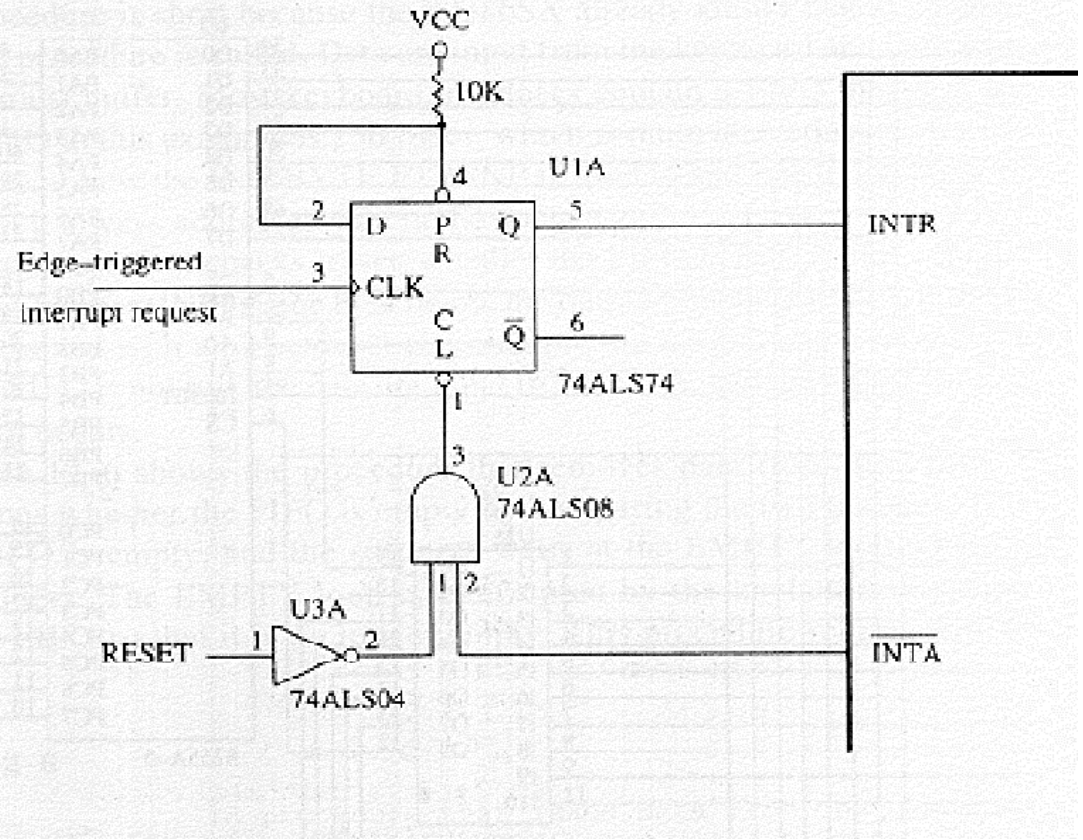
# Interrupt circuits

**URE 12–10** A circuit
t applies any interrupt
tor type number in re-
nse to INTA. Here the
uit is applying type
nber 80H.



# Interrupt circuits

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



# Interrupt circuits

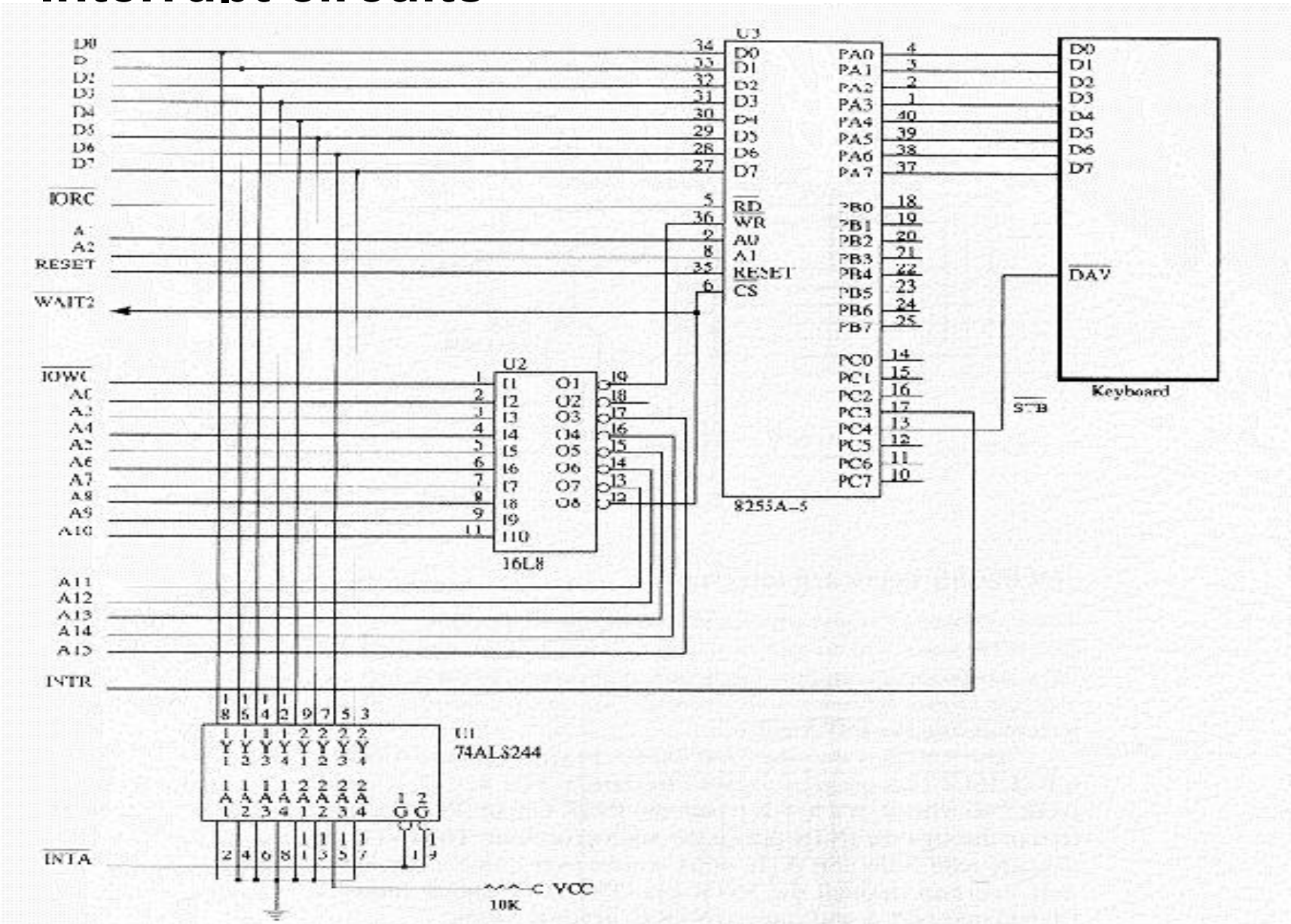



FIGURE 12–12 An 82C55 interfaced to a keyboard from the microprocessor system using interrupt vector 40H.

# Description

- 8255 is decoded at 0500h, 0502h, 0504h, and 0506h
- 8255 is operated at in mode 1 (stobed input)
- Whenever a key is typed , the INTR output (PC3) becomes a logic 1 and requests an interrupt thru the INTR pin on the microprocessor
- The INTR remains high until the ASCII data are read form port A.
- In other words, every time a key is typed the 8255 requests a type 40h interrupt thru 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
- Data are input from the keyboard and then stored in the FIFO (first in first out) buffer
- FIFO in our example is 256 bytes
- The 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)

# Example: “Read from the Keyboard routine” into FIFO

```
                ; interrupt service routine ro read a key from the
                ; keyboard
                PORTA       EQU    500h
                CNTR        EQU    506h
                FIFO        DB     256  DUP (?)
                INP         DW     ?
                OUTP        DW     ?
                KEY         PROC FAR USES AX BX DI DX
                MOV         BX, CS:INP
                MOV         DI, CS:OUTP
                INC         BL
                CMP         BX, DI          ;test for queue full
                JE          FULL            ; if queue is full
                DEC         BL
                MOV         DX, PORTA
                IN          AL,DX           ; read the key
                MOV         CS:[BX], AL
                INC         BYTE PTR INP
                JMP         DONE
FULL:           MOV         AL,8
                MOV         DX, CNTR
                OUT         DX,AL
DONE:           IRET
                KEY         ENDP
```

# Example contd: “Read from the FIFO into AH”

```
READ    PROC    FAR USES   BX DI DX
EMPTY:  MOV    BX, CS:INP
        MOV    DI, CS:OUTP
        CMP    BX,DI
        JE     EMPTY
        MOV    AH, CS:DI
        MOV    AL,9        ; enable 8255 interrupt
        MOV    DX, CNTR
        OUT    DX,AL
        INC    BYTE PTR CS:OUTP
        RET
READ    ENDP
```