Converting from ASCII to Binary

Conversions from ASCII to binary usually start with keyboard entry. If a single key is typed, the conversion occurs when a 30H is subtracted from the number. If more than one key is typed, conversion from ASCII to binary still requires 30H to be subtracted, but there is one additional step. After subtracting 30H, the number is added to the result after the prior result is first multiplied by 10.

The algorithm for converting from ASCII to binary is:

1. Divide
1. Begin with a binary result of 0.
2. Subtract 30H from the character typed on the keyboard to convert it to BCD.
3. Multiply the result by 10, and then add the new BCD digit.
4. Repeat steps 2 and 3 until the character typed is not an ASCII-coded number.

Example 7–25 illustrates a procedure (READN) used in a program that implements this algorithm. Here, the binary number returns in the AX register as a 16-bit result, which is then stored in memory location TEMP. If a larger result is required, the procedure must be reworked for a 32-bit addition. Each time this procedure is called, it reads a number from the keyboard until any key other than 0 through 9 is typed.

EXAMPLE 7–25

```
                        ;A program that reads one decimal number from the
                        ;keyboard and stores the binary value at TEMP.
                        ;
                                .MODEL SMALL            ;select TINY model
0000                            .DATA                   ;start DATA segment
0000  0000              TEMP    DW   ?                  ;define TEMP
0000                            .CODE                   ;start CODE segment
                                .STARTUP                ;start program
0017  E8 0007                   CALL READN              ;read a number
001A  A3 0000 R                 MOV  TEMP,AX            ;save it in TEMP
                                .EXIT                   ;exit to DOS
                        ;
                        ;The READN procedure reads a decimal number from the
                        ;keyboard and returns its binary value in AX.
                        ;
0021                    READN   PROC NEAR

0021  53                        PUSH BX                 ;save BX and CX
0022  51                       PUSH CX
0023  B9 000A                  MOV  CX,10              ;load 10 for decimal
0026  BB 0000                  MOV  BX,0               ;clear result
0029                    READN1:
0029  B4 01                     MOV  AH,1               ;read key with echo
002B  CD 21                     INT  21H

002D  3C 30                     CMP  AL,'0'
002F  72 14                     JB   READN2             ;if below '0'
0031  3C 39                     CMP  AL,'9'
0033  77 10                     JA   READN2             ;if above '9'

0035  2C 30                     SUB  AL,'0'             ;convert to ASCII

0037  50                        PUSH AX                 ;save digit
0038  8B C3                     MOV  AX,BX              ;multiply result by 10
003A  F7 E1                     MUL  CX
003C  8B D8                     MOV  BX,AX
003E  58                        POP  AX
003F  B4 00                     MOV  AH,0
0041  03 D8                     ADD  BX,AX              ;add digit value to result
```

```
0043  EB E4              JMP  READN1          ;repeat
0045              READN2:
0045  8B C3              MOV  AX,BX           ;get binary result into AX
0047  59                 POP  CX              ;restore CX and BX
0048  5B                 POP  BX
0049  C3                 RET

004A              READN   ENDP
                          END                  ;end of file
```

Displaying and Reading Hexadecimal

Hexadecimal data are easier to read from the keyboard and display than decimal data. These types of data are not used at the applications level, but at the system level. System-level data are often hexadecimal, and must either be displayed in hexadecimal form or read from the keyboard as hexadecimal data.

Reading Hexadecimal Data. Hexadecimal data appear as 0 to 9 and A to F. The ASCII codes obtained from the keyboard for hexadecimal data are 30H to 39H for the numbers 0 through 9, and 41H to 46H (A–F) or 61H to 66H (a–f) for the letters. To be useful, a procedure that reads hexadecimal data must be able to accept both lowercase and uppercase letters.

Example 7–26 shows two procedures: one (CONV) converts the contents of the data in AL from ASCII code to a single hexadecimal digit, and the other (READH) reads a four-digit hexadecimal number from the keyboard and returns with it in register AX. This procedure can be modified to read any-sized hexadecimal number from the keyboard.

EXAMPLE 7–26

```
                    ;A program that reads a 4-digit hexadecimal number from
                    ;the keyboard and stores the result in word-sized
                    ;memory location TEMP.
                    ;
                            .MODEL SMALL          ;select SMALL model
0000                        .DATA                 ;start DATA segment
0000  0000          TEMP    DW   ?                ;define TEMP
0000                        .CODE                 ;start CODE segment
                            .STARTUP              ;start program
0017  E8 0007               CALL READH            ;read hexadecimal number
001A  A3 0000 R             MOV  TEMP,AX          ;save it at TEMP
                            .EXIT                 ;exit to DOS
                    ;
                    ;The READH procedure that reads a 4-digit hexadecimal
                    ;number from the keyboard and returns it in AX.
                    ;This procedure does next check for errors and uses CONV.
                    ;
0021                READH   PROC NEAR

0021  51                    PUSH CX               ;save BX and CX
0022  53                    PUSH BX
0023  B9 0004               MOV  CX,4             ;load CX and SI with 4
0026  8B F1                 MOV  SI,CX
0028  BB 0000               MOV  BX,0             ;clear result
002B                READH1:
002B  B4 01                 MOV  AH,1             ;read a key with echo
002D  CD 21                 INT  21H
002F  E8 000A               CALL CONV             ;convert to binary
0032  D3 E3                 SHL  BX,CL
0034  02 D8                 ADD  BL,AL            ;form result in BX
0036  4E                    DEC  SI
0037  75 F2                 JNZ  READH1           ;repeat 4 times
0039  8B C3                 MOV  AX,BX            ;move result to AX
003B  5B                    POP  BX               ;restore BX and CX
```

```
003C  59                POP  CX
003D  C3                RET
003E            READH   ENDP
                ;
                ;The CONV procedure converts AL into hexadecimal.
                ;
003E            CONV    PROC NEAR

003E  3C 39             CMP  AL,'9'
0040  76 08             JBE  CONV2          ;if 0 through 9
0042  3C 61             CMP  AL,'a'
0044  72 02             JB   CONV1          ;if uppercase A through F
0046  2C 20             SUB  AL,20H         ;convert to uppercase
0048            CONV1:
0048  2C 07             SUB  AL,7
004A            CONV2:
004A  2C 30             SUB  AL,30H
004C  C3                RET
004D            CONV    ENDP
                        END                 ;end of file
```

Displaying Hexadecimal Data. To display hexadecimal data, a number must be divided into four-bit segments that are converted into hexadecimal digits. Conversion is accomplished by adding a 30H to the numbers 0 to 9 and a 37H to the letters A to F.

A procedure (DSIPH) that displays the contents of the AX register on the video display appears in the program of Example 7–27. Here, the number is rotated left so that the leftmost digit is displayed first. Because AX contains a four-digit hexadecimal number, the procedure displays four hexadecimal digits.

EXAMPLE 7–27

```
                ;A program that displays the hexadecimal value in AX.
                ;This program uses DISPH to display a 4-digit value.
                ;
                        .MODEL TINY         ;select TINY model
0000                    .CODE               ;start CODE segment
                        .STARTUP            ;start program
0100  B8 0ABC           MOV  AX,0ABCH       ;load AX with test data
0103  E8 0004           CALL DISPH          ;display AX in hexadecimal
                        .EXIT               ;exit to DOS
                ;
                ;The DISPH procedure displays AX as a 4-digit hex number.
                ;
010A            DISPH   PROC NEAR

010A  53                PUSH BX             ;save BX and CX
010B  51                PUSH CX
010C  B1 04             MOV  CL,4           ;load rotate count
010E  B5 04             MOV  CH,4           ;load digit count
0110            DISPH1:
0110  D3 C0             ROL  AX,CL          ;position digit
0112  50                PUSH AX
0113  24 0F             AND  AL,0FH         ;convert it to ASCII
0115  04 30             ADD  AL,30H
0117  3C 39             CMP  AL,'9'
0119  76 02             JBE  DISPH2
011B  04 07             ADD  AL,7
011D            DISPH2:
011D  B4 02             MOV  AH,2           ;display hexadecimal digit
011F  8A D0             MOV  DL,AL
0121  CD 21             INT  21H
0123  58                POP  AX
0124  FE CD             DEC  CH
```

```
0126  75 E8              JNZ  DISPH1          ;repeat for 4 digits
0128  59                 POP  CX              ;restore registers
0129  5B                 POP  BX
012A  C3                 RET

012B            DISPH    ENDP
                         END                  ;end of file
```

Using Lookup Tables for Data Conversions

Lookup tables are often used to convert data from one form to another. A lookup table is formed in the memory as a list of data that is referenced by a procedure to perform conversions. In the case of many lookup tables, the XLAT instruction can often be used to look up data in a table, provided that the table contains eight-bit wide data and its length is less than or equal to 256 bytes.

Converting from BCD to 7-segment Code. One simple application that uses a lookup table is BCD to 7-segment code conversion. Example 7–28 illustrates a lookup table that contains the 7-segment codes for the numbers 0 to 9. These codes are used with the 7-segment display pictured in Figure 7–1. This 7-segment display uses active high (logic 1) inputs to light a segment. The code is arranged so that the **a** segment is in bit position 0 and the **g** segment is in bit position 6. Bit position 7 is 0 in this example, but it can be used for displaying a decimal point.

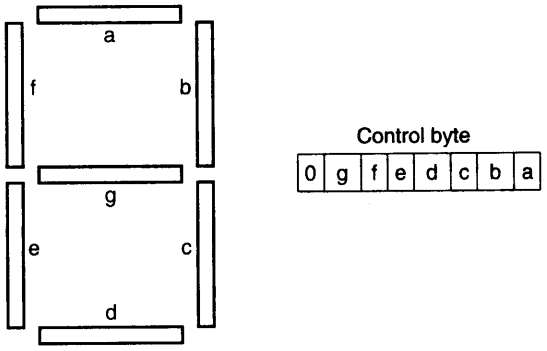

FIGURE 7–1 The 7-segment display.

EXAMPLE 7–28

```
0000            SEG7     PROC  FAR

0000  53                 PUSH  BX
0001  BB 0008 R          MOV   BX,OFFSET TABLE
0004  2E: D7             XLAT  CS:TABLE       ;see text
0006  5B                 POP   BX
0007  CB                 RET

0008  3F         TABLE   DB    3FH            ;0
0009  06                 DB    6              ;1
000A  5B                 DB    5BH            ;2
000B  4F                 DB    4FH            ;3
000C  66                 DB    66H            ;4
000D  6D                 DB    6DH            ;5
000E  7D                 DB    7DH            ;6
000F  07                 DB    7              ;7
0010  7F                 DB    7FH            ;8
0011  6F                 DB    6FH            ;9

0012            SEG7     ENDP
```

The procedure that performs the conversion contains only two instructions and assumes that AL contains the BCD digit to be converted to 7-segment code. One of the instructions addresses the lookup table by loading its address into BX, and the other performs the conversion and returns the 7-segment code in AL.

Because the lookup table is located in the code segment and the XLAT instruction accesses the data segment by default, the XLAT instruction includes a segment override. Notice that a dummy operand (TABLE) is added to the XLAT instruction so that the (CS:) code segment override prefix can be added to the instruction. Normally,

XLAT does not contain an operand unless its default segment must be overridden. The LODS and MOVS instructions are also overridden in the same manner as XLAT by using a dummy operand.

Using a Lookup Table to Access ASCII Data. Some programming techniques require that numeric codes be converted to ASCII character strings. For example, suppose that you need to display the days of the week for a calendar program. Because the number of ASCII characters in each day is different, some type of lookup table must be used to reference the ASCII-coded days of the week.

The program in Example 7–29 shows a table that references ASCII-coded character strings located in the code segment. Each character string contains an ASCII-coded day of the week. The table references each day of the week. The procedure that accesses the day of the week uses the AL register, and the numbers 0 to 6 to refer to Sunday through Saturday. If AL contains a 2 when this procedure is called, the word *"Tuesday"* is displayed on the video screen.

EXAMPLE 7–29

```
                        ;A program that displays the current day of the
                        ;week by using the system clock/calendar.
                        ;
                                .MODEL SMALL            ;select SMALL model
0000                            .DATA                   ;start DATA segment
0000  000E R 0015 R     DTAB    DW   SUN,MON,TUE,WED,THU,FRI,SAT
      001C R 0024 R
      002E R 0037 R
      003E R
000E  53 75 6E 64 61 79 SUN     DB   'Sunday$'
      24
0015  4D 6F 6E 64 61 79 MON     DB   'Monday$'
      24
001C  54 75 65 73 64 61 TUE     DB   'Tuesday$'
      79 24
0024  57 65 64 6E 65 73 WED     DB   'Wednesday$'
      64 61 79 24
002E  54 68 75 72 73 64 THU     DB   'Thursday$'
      61 79 24
0037  46 72 69 64 61 79 FRI     DB   'Friday$'
      24
003E  53 61 74 75 72 64 SAT     DB   'Saturday$'
      61 79 24
0000                    .CODE                           ;start CODE segment
                        .STARTUP                        ;start program
0017  B4 2A                     MOV  AH,2AH             ;get day of week
0019  CD 21                     INT  21H                ;access DOS
001B  E8 0004                   CALL DAYS               ;display day of week
                        .EXIT                           ;exit to DOS

0022                    DAYS    PROC NEAR

0022  52                       PUSH DX                 ;save DX and SI
0023  56                       PUSH SI
0024  BE 0000 R                 MOV  SI,OFFSET DTAB     ;address table
0027  B4 00                     MOV  AH,0               ;find day of week
0029  03 C0                     ADD  AX,AX
002B  03 F0                     ADD  SI,AX
002D  8B 14                     MOV  DX,[SI]            ;get day of week
002F  B4 09                     MOV  AH,9               ;display string
0031  CD 21                     INT  21H
0033  5E                       POP  SI                 ;restore registers
0034  5A                       POP  DX
0035  C3                       RET
```

```
0036                            DAYS    ENDP
                                        END                     ;end of file
```

This procedure first addresses the table by loading its address into the SI register. Next, the number in AL is converted into a 16-bit number and doubled because the table contains two bytes for each entry. This index is then added to SI to address the correct entry in the lookup table. The address of the ASCII character string is now loaded into DX by the MOV DX,CS:[SI] instruction.

Before the INT 21H DOS function is called, the DS register is placed on the stack and loaded with the segment address of CS. This allows DOS function number 09H (display a string) to be used to display the day of the week. This procedure converts the numbers 0 to 6 to the days of the week.

An Example Program Using Data Conversions

A program example is required to combine some of the data-conversion DOS functions. Suppose that you must display the time and date on the video screen. This example program (see Example 7–30) displays the time as 10:45 A.M. and the date as Tuesday, May 14, 2002. The program is short because it calls a procedure that displays the time and a second procedure that displays the date.

The time is available from DOS, using an INT 21H function call number 2CH. This returns with the hours in CH and minutes in CL. Also available are seconds in DH and hundredths of seconds in DL. The date is available by using INT 21H function call number 2AH. This leaves the day of the week in AL, the year in CX, the day of the month in DH, and the month in DL.

EXAMPLE 7–30

```
                                ;A program that displays the time and date in the
                                ;form:  10:45 A.M.,  Tuesday May 14, 2002.
                                ;
                                        .MODEL SMALL            ;select SMALL model
                                        .NOLISTMACRO            ;don't expand macros
0000                                    .DATA                   ;start CODE segment
0000  0026 R 002F R             DTAB    DW   SUN,MON,TUE,WED,THU,FRI,SAT
      0038 R 0042 R
      004E R 0059 R
      0062 R
000E  006D R 0076 R             MTAB    DW   JAN,FEB,MAR,APR,MAY,JUN
      0080 R 0087 R
      008E R 0093 R
001A  0099 R 009F R                     DW   JUL,AUG,SEP,OCT,NOV,DCE
      00A7 R 00B2 R
      00BB R 00C5 R
0026  53 75 6E 64 61 79        SUN     DB   'Sunday, $'
      2C 20 24
002F  4D 6F 6E 64 61 79        MON     DB   'Monday, $'
      2C 20 24
0038  54 75 65 73 64 61        TUE     DB   'Tuesday, $'
      79 2C 20 24
0042  57 65 64 6E 65 73        WED     DB   'Wednesday, $'
      64 61 79 2C 20 24
004E  54 68 75 72 73 64        THU     DB   'Thursday, $'
      61 79 2C 20 24
0059  46 72 69 64 61 79        FRI     DB   'Friday, $'
      2C 20 24
0062  53 61 74 75 72 64        SAT     DB   'Saturday, $'
      61 79 2C 20 24
006D  4A 61 6E 75 61 72        JAN     DB   'January $'
      79 20 24
0076  46 65 62 72 75 61        FEB     DB   'February $'
      72 79 20 24
0080  4D 61 72 63 68 20        MAR     DB   'March $'
```

```
      24
0087  41 70 72 69 6C 20  APR     DB   'April $'
      24
008E  4D 61 79 20 24      MAY     DB   'May $'
0093  4A 75 6E 65 20 24   JUN     DB   'June $'
0099  4A 75 6C 79 20 24   JUL     DB   'July $'
009F  41 75 67 75 73 74   AUG     DB   'August $'
      20 24
00A7  53 65 70 74 65 6D   SEP     DB   'September $'
      62 65 72 20 24
00B2  4F 63 74 6F 62 65   OCT     DB   'October $'
      72 20 24
00BB  4E 6F 76 65 6D 62   NOV     DB   'November $'
      65 72 20 24
00C5  44 65 63 65 6D 62   DCE     DB   'December $'
      65 72 20 24
0000                              .CODE                ;start CODE segment
                          DISP    MACRO  CHAR
                                  PUSH AX              ;;save AX and DX
                                  PUSH DX
                                  MOV  DL,CHAR         ;;display character
                                  MOV  AH,2
                                  INT  21H
                                  POP  DX              ;;restore AX and DX
                                  POP  AX
                                  ENDM
                                  .STARTUP             ;start program
0017  E8 0007                    CALL TIMES           ;display time
001A  E8 00A3                     CALL DATES           ;display date
                                  .EXIT                ;exit to DOS

0021                      TIMES   PROC NEAR

0021  B4 2C                       MOV  AH,2CH          ;get time from DOS
0023  CD 21                       INT  21H
0025  B7 41                       MOV  BH,'A'          ;set 'A' for AM
0027  80 FD 0C                    CMP  CH,12
002A  72 05                       JB   TIMES1          ;if below 12:00 noon
002C  B7 50                       MOV  BH,'P'          ;set 'P' for PM
002E  80 ED 0C                    SUB  CH,12           ;adjust to 12 hours
0031                      TIMES1:
0031  0A ED                       OR   CH,CH           ;test for 0 hour
0033  75 02                       JNE  TIMES2          ;if not 0 hour
0035  B5 0C                       MOV  CH,12           ;change 0 hour to 12
0037                      TIMES2:
0037  8A C5                       MOV  AL,CH
0039  B4 00                       MOV  AH,0
003B  D4 0A                       AAM                  ;convert hours
003D  0A E4                       OR   AH,AH
003F  74 0D                       JZ   TIMES3          ;if no tens of hours
0041  80 C4 30                    ADD  AH,'0'          ;convert tens
                                  DISP AH              ;display tens
004E                      TIMES3:
004E  04 30                       ADD  AL,'0'          ;convert units
                                  DISP AL              ;display units
                                  DISP ':'             ;display colon
0064  8A C1                       MOV  AL,CL
0066  B4 00                       MOV  AH,0
0068  D4 0A                       AAM                  ;convert minutes
006A  05 3030                     ADD  AX,3030H
006D  50                          PUSH AX
                                  DISP AH              ;display tens
```

```
0078  58                          POP  AX
                                  DISP AL              ;display units
                                  DISP ' '             ;display space
                                  DISP BH              ;display 'A' or 'P'
                                  DISP '.'             ;display .
                                  DISP 'M'             ;display M
                                  DISP '.'             ;display .
                                  DISP ' '             ;display space
00BF  C3                          RET

00C0                      TIMES   ENDP

00C0                      DATES   PROC NEAR

00C0  B4 2A                       MOV  AH,2AH          ;get date from DOS
00C2  CD 21                       INT  21H
00C4  52                          PUSH DX
00C5  B4 00                       MOV  AH,0            ;get day of week
00C7  03 C0                       ADD  AX,AX
00C9  BE 0000 R                   MOV  SI,OFFSET DTAB  ;address day table
00CC  03 F0                       ADD  SI,AX
00CE  8B 14                       MOV  DX,[SI]         ;address day of week
00D0  B4 09                       MOV  AH,9            ;display day of week
00D2  CD 21                       INT  21H
00D4  5A                          POP  DX
00D5  52                          PUSH DX
00D6  8A C6                       MOV  AL,DH           ;get month
00D8  FE C8                       DEC  AL
00DA  B4 00                       MOV  AH,0
00DC  03 C0                       ADD  AX,AX
00DE  BE 000E R                   MOV  SI,OFFSET MTAB  ;address month table
00E1  03 F0                       ADD  SI,AX
00E3  8B 14                       MOV  DX,[SI]         ;address month
00E5  B4 09                       MOV  AH,9            ;display month
00E7  CD 21                       INT  21H
00E9  5A                          POP  DX
00EA  8A C2                       MOV  AL,DL           ;get day of month
00EC  B4 00                       MOV  AH,0
00EE  D4 0A                       AAM                  ;convert to BCD
00F0  0A E4                       OR   AH,AH
00F2  74 0D                       JZ   DATES1          ;if tens is 0
00F4  80 C4 30                    ADD  AH,30H          ;convert tens
                                  DISP AH              ;display tens
0101                      DATES1:
0101  04 30                       ADD  AL,30H          ;convert units
                                  DISP AL              ;display units
                                  DISP ','             ;display comma
                                  DISP ' '             ;display space
0121  81 F9 07D0                  CMP  CX,2000         ;test for year 2000
0125  72 19                       JB   DATES2          ;if below year 2000
0127  83 E9 64                    SUB  CX,100          ;scale to 1900 - 1999
                                  DISP '2'             ;display 2
                                  DISP '0'             ;display 0
013E  EB 14                       JMP  DATES3
0140                      DATES2:
                                  DISP '1'             ;display 1
                                  DISP '9'             ;display 9
0154                      DATES3:
0154  81 E9 076C                  SUB  CX,1900         ;scale to 00 - 99
0158  8B C1                       MOV  AX,CX
015A  D4 0A                       AAM                  ;convert to BCD
```

```
015C  05 3030                     ADD  AX,3030H        ;convert to ASCII
                                  DISP AH              ;display tens
                                  DISP AL              ;display units
0173  C3                          RET

0174                        DATES ENDP
                                  END                  ;end of file
```

This procedure uses two ASCII lookup tables that convert the day and month to ASCII character strings. It also uses the AAM instruction to convert from binary to BCD for the time and date. The displaying of data is handled in two ways: by character string (function 09H) and by single character (function 06H).

The memory model (SMALL) consists of two segments: .DATA and .CODE. The data segment contains the character strings used with the procedures that display time and date. The code segment contains TIMES and DATES procedures, and a macro (DISP) that displays an ASCII character. The main program is very short and consists of two CALL instructions. The year 2000 problem is corrected in this program, but not the year 2100 problem.

Numeric Sort Program

At times, numbers must be sorted into numeric order. This is often accomplished with a bubble sort. Figure 7–2 shows five numbers that are sorted with a bubble sort. Notice that the set of five numbers is tested four times with four passes. For each pass, two consecutive numbers are compared and sometimes exchanged. Also notice that during the first pass, there are four comparisons, during the second three, etc.

Example 7–31 illustrates a program that accepts 10 numbers from the keyboard (0–65535). After these 16-bit numbers are accepted and stored in memory section ARRAY, they are sorted by using the bubble-sorting technique. This bubble sort uses a flag to determine whether any numbers were exchanged in a pass. If no numbers were exchanged, the numbers are in order and the sort terminates.

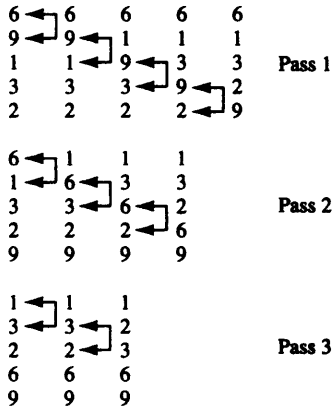

FIGURE 7–2 A bubble sort showing data as they are sorted. Note: Sorting five numbers may require four passes.

EXAMPLE 7–31

```
                            .MODEL SMALL
0000                        .DATA
0000  000A [                ARRAY DW 10 DUP (?)
;array
          0000
              ]
0014  0D 0A 45 6E 74 65     MES1  DB 13,10,'Enter 10 numbers:',13,10,10,'$'
      72 20 31 30 20 6E
      75 6D 62 65 72 73
      3A 0D 0A 0A 24
002B  0D 0A 0A 53 6F 72     MES2  DB 13,10,10,'Sorted Data:',13,10,10,'$'
      74 65 64 20 44 61
      74 61 3A 0D 0A 0A
      24
0000                        .CODE

                            DISP   MACRO PARA

                                   PUSH  AX
                                   MOV   AH,6
                                   MOV   DL,PARA
                                   INT   21H
```

```
                                    POP   AX

                                    ENDM
                                    GET   MACRO

                                    .REPEAT
                                       MOV   AH,6
                                       MOV   DL,-1
                                       INT   21H
                                    .UNTIL (AL>='0' && AL <='9') || AL==13 || AL==','
                                    DISP  AL
                                    .IF AL==13
                                       DISP  10
                                    .ENDIF
                                    .IF AL>='0' && AL<='9'
                                       SUB   AL,'0'
                                    .ENDIF

                                    ENDM

                                    STRING   MACRO WHERE

                                       MOV   DX,OFFSET WHERE
                                       MOV   AH,9
                                       INT   21H

                                    ENDM
                              .STARTUP

                                    STRING   MES1
001E  FC                            CLD
001F  B9 000A                       MOV   CX,10
0022  BF 0000 R                     MOV   DI,OFFSET ARRAY
0025  8C D8                         MOV   AX,DS
0027  8E C0                         MOV   ES,AX
                                    .REPEAT
0029  E8 0026                          CALL   GETN     ;get 10 numbers
                                    .UNTILCXZ
                                    STRING    MES2
0035  E8 008B                       CALL  SORT        ;sort 10 numbers
0038  B9 0009                       MOV   CX,9
003B  BE 0000 R                     MOV   SI,OFFSET ARRAY
                                    .REPEAT           ;display 10 numbers
003E  E8 0061                          CALL  DISPN
                                       DISP  ','
                                    .UNTILCXZ
004B  E8 0054                       CALL  DISPN
                                    .EXIT

0052                          GETN  PROC  NEAR

0052  BD 000A                       MOV   BP,10
0055  BB 0000                       MOV   BX,0
                                    .WHILE 1
                                       GET
                                       .BREAK .IF AL==13 || AL==','
0094  93                             XCHG .AX,BX
0095  F7 E5                          MUL   BP
0097  93                             XCHG  AX,BX
0098  B4 00                          MOV   AH,0
009A  03 D8                          ADD   BX,AX
                                       .ENDW
009E  8B C3                          MOV   AX,BX
00A0  AB                             STOSW
```

```
00A1  C3                                    RET

00A2                                GETN    ENDP

00A2                                DISPN   PROC    NEAR

00A2  BB 000A                               MOV     BX,10
00A5  53                                    PUSH    BX
00A6  AD                                    LODSW
                                            .REPEAT
00A7  BA 0000                                  MOV     DX,0
00AA  F7 F3                                    DIV     BX
00AC  52                                       PUSH    DX
                                            .UNTIL AX==0
                                            .WHILE 1
00B1  58                                       POP     AX
                                            .BREAK .IF AL==10
00B6  04 30                                    ADD     AL,'0'
                                               DISP    AL
                                            .ENDW
00C2  C3                                    RET

00C3                                DISPN   ENDP

00C3                                SORT    PROC    NEAR

00C3  BB 0009                               MOV     BX,9
                                            .REPEAT
00C6  8B CB                                    MOV   CX,BX
00C8  BE 0000 R                                MOV   SI,OFFSET ARRAY
00CB  B2 00                                    MOV   DL,0
                                               .REPEAT
00CD  AD                                          LODSW
00CE  3B 04                                       CMP   AX,[SI]
                                                  .IF !CARRY?
00D2  8B 2C                                          MOV   BP,[SI]
00D4  89 6C FE                                       MOV   [SI-2],BP
00D7  89 04                                          MOV   [SI],AX
00D9  FE C2                                          INC   DL
                                                  .ENDIF
                                               .UNTILCXZ
00DD  4B                                       DEC   BX
                                            .UNTIL BX==0 || DL==0
00E6  C3                                    RET

00E7                                SORT    ENDP
                                            END
```

Once the numbers are sorted, they are displayed on the video screen in ascending numerical order. No provision is made for errors as each number is typed. The program terminates after sorting one set of 10 numbers and must be invoked again to sort 10 new numbers.

7–4 INTERRUPT HOOKS

Hooks are used to tap into or intercept the interrupt structure of the microprocessor. For example, we might hook into the keyboard interrupt so that we can detect a special keystroke called a *hot key*. Whenever the hot key is typed, we can access a terminate and stay resident (TSR) program that performs a special task. Some examples of hot key applications are pop-up calculators and pop-up clocks.

Intercepting an Interrupt

In order to intercept an interrupt, we must use a DOS function call that reads the current address from the interrupt vector. DOS function call number 35H is used to read the current interrupt vector and DOS function call number 25H is used to change the address of the current vector. In both DOS function calls, AL indicates the vector type number (00H–FFH) and AH indicates the DOS function call number.

When the vector is read by using function 35H, the offset address is returned in register BX and the segment address is in register ES. These two registers are saved so that they can be restored when the interrupt hook is removed from memory. When the vector is set, it is set to the address stored at the memory location addressed by DS:DX.

The process of installing an interrupt handler through a hook is illustrated in the program of Example 7–32. This program intercepts the divide error interrupt by first reading the current interrupt vector address and storing it into a double-word memory location for access by the new interrupt service procedure. Next, the address of the new interrupt service procedure, stored in DS:DX, is placed into the vector using DOS function call number 25H.

EXAMPLE 7–32

```
                        ;A sequence of instructions that show the installation
                        ;or a new interrupt for vector 0 (divide error).
                        ;Note this is not a complete program.
                        ;
                                .MODEL TINY
0000                            .CODE
                                .STARTUP
0100  EB 05                     JMP  MAIN               ;skip

0102  00000000          ADDR    DD   ?                  ;old interrupt vector

0106                    NEW     PROC FAR                ;new interrupt procedure

0106  CF                        IRET                    ;do nothing interrupt

0107                    NEW     ENDP

0107                    MAIN:

0107  8C C8                     MOV  AX,CS              ;address CS with DS
0109  8E D8                     MOV  DS,AX

                        ;get vector 0 address

010B  B8 3500                   MOV  AX,3500H
010E  CD 21                     INT  21H

                        ;save vector address at ADDR

0110  89 1E 0102 R              MOV  WORD PTR ADDRESS,BX
0114  8C 06 0104 R              MOV  WORD PTR ADDRESS+2,ES

                        ;install new interrupt vector 0 address

0118  B8 2500                   MOV  AX,2500H
011B  BA 0106 R                 MOV  DX,OFFSET NEW
011E  CD 21                     INT  21H

                        ;other installation software continues here
```

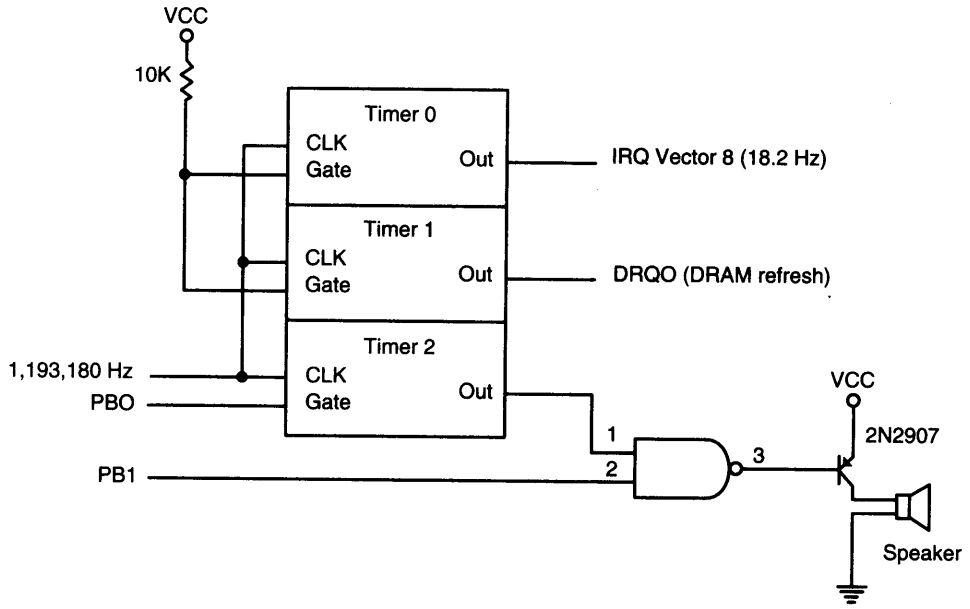

FIGURE 7–3 The speaker and timer circuit in the personal computer (I/O ports 40–43H program the timer, and I/O port 61H programs PB0 and PB1).

Example TSR Alarm

A simple example showing an interrupt hook and TSR causes a beep on the speaker after one hour or one-half hour. We all seem to get lost in computer processing, and this program makes it easy to keep track of time because of the audible beep.

The beep is caused by using timer 2 of the timer found inside the PC in order to generate an audio tone at the speaker. (See Section 12–5 for a discussion of the timer and see Figure 7–3 for its connection in the computer.) Programming timer 2 with a particular beep frequency or tone is accomplished by programming timer 2 with 1,193,180, divided by the desired tone. For example, if we divide 1,193,180 by 800, the speaker generates an 800 Hz audio tone. See the BEEP procedure (shown in Example 7–33) for programming the timer, and turning the speaker on and off after a short wait determined by the number of clock ticks. This procedure uses six clock ticks to produce a beep lasting 1/3 second. Note that each clock tick occurs about 18.2 times a second (the actual time is closer to 18.206). This is accomplished by using the user wait timer locations in the first segment of the memory. The user wait timer is updated 18.2 times per second by the computer so that it can be used to time events. The program that uses the BEEP procedure causes an audio tone of 1000 Hz, 1200 Hz, and 1400 Hz (each with a 1/3-second duration) to repeat four times.

EXAMPLE 7–33

```
                        ;A program that beeps the speaker with some sample audio
                        ;tones that each have a duration of 1/3 second.
                        ;
                                .MODEL TINY
0000                            .CODE
                                .STARTUP
0100  B8 0000                   MOV   AX,0
0103  8E D8                     MOV   DS,AX             ;address segment 0000H
0105  B9 0004                   MOV   CX,4              ;set count to 4
0108  E4 61                     IN    AL,61H            ;enable timer and speaker
```

```
010A  0C 03              OR   AL,3             ;set PB0 and PB1
010C  E6 61              OUT  61H,AL
010E                MAIN1:
010E  BB 03E8            MOV  BX,1000          ;select 1000 Hz tone
0111  E8 0018            CALL BEEP
0114  BB 04B0            MOV  BX,1200          ;select 1200 Hz tone
0117  E8 0012            CALL BEEP
011A  BB 0578            MOV  BX,1400          ;select 1400 Hz tone
011D  E8 000C            CALL BEEP
0120  E2 EC              LOOP MAIN1            ;repeat 4 times

0122  E4 61              IN   AL,61H           ;turn speaker off
0124  34 03              XOR  AL,3             ;clear PB0 and PB1
0126  E6 61              OUT  61H,AL
                         .EXIT
                    ;
                    ;The BEEP procedure programs timer 2 to beep the speaker
                    ;for 1/3 of a second with the frequency BX.
                    ;***input parameters***
                    ;BX = desired audio tone
                    ;***uses***
                    ;WAITS procedure to wait for 1/3 second
                    ;
012C                BEEP    PROC NEAR             ;beep speaker 1/3 second

012C  B8 34DC            MOV  AX,34DCH         ;load AX with 1,193,180
012F  BA 0012            MOV  DX,12H
0132  F7 F3              DIV  BX               ;find count
0134  E6 42              OUT  42H,AL           ;program timer 2
0136  8A C4              MOV  AL,AH
0138  E6 42              OUT  42H,AL
013A  E8 0001            CALL WAITS            ;wait 1/3 second
013D  C3                 RET

013E                BEEP    ENDP
                    ;
                    ;the WAITS procedure waits 1/3 of a second
                    ;***uses***
                    ;memory doubleword location 0000:46CH to time the wait
                    ;
013E                WAITS   PROC NEAR

013E  BA 0006            MOV  DX,6             ;number of clock ticks
0141  BB 0000            MOV  BX,0
0144  03 16 046C         ADD  DX,DS:[46CH]     ;get tick count plus time
0148  13 1E 046E         ADC  BX,DS:[46EH]
014C                WAIT1:
014C  8B 2E 046C         MOV  BP,DS:[46CH]     ;test for elapsed time
0150  A1 046E            MOV  AX,DS:[46EH]
0153  2B EA              SUB  BP,DX
0155  1B C3              SBB  AX,BX
0157  72 F3              JC   WAIT1            ;keep testing

0159  C3                 RET

015A                WAITS   ENDP
                            END
```

The CHIME program (see Example 7–34) hooks into interrupt vector 8 and beeps the speaker once each half-hour and twice on the hour. This program is a TSR and remains active until the computer is turned off. Note how the TSR is installed and how the interrupt vector is hooked. Also notice that the normal interrupt vector 8 procedure continues to execute, even as the beeper is activated.

EXAMPLE 7–34

```
                        ;A terminate and stay resident program that hooks into
                        ;interrupt vector 8 to beep the speaker one time per
                        ;half-hour and two times per hour.
                        ;***must be assembled as a .COM file*** for use with
                        ;version 5.10 of MASM
                                .MODEL TINY
0000                            .CODE
                                .STARTUP
0100  E9 00CE                   JMP   INSTALL           ;install interrupt
   =  03E8              TONE    EQU   1000              ;set tone at 1000 Hz
0103  00                COUNT   DB    0                 ;elapsed time counter
0104  00000000          ADD8    DD    ?                 ;old vector address
0108  00                PASS    DB    0                 ;1 or 2 beeps
0109  00                BEEP    DB    0                 ;beep or silent
010A  00                FLAG    DB    0                 ;busy flag

010B                    VEC8    PROC FAR                ;interrupt procedure

010B  2E: 80 3E 010A R          CMP   CS:FLAG,0         ;test busy flag
      00
0111  74 05                     JE    VEC81             ;if not busy
0113  2E: FF 2E 0104 R          JMP   CS:ADD8           ;if busy do normal INT 8
0118                    VEC81:
0118  9C                        PUSHF                   ;do normal INT 8
0119  2E: FF 1E 0104 R          CALL CS:ADD8
011E  2E: C6 06 010A R          MOV   CS:FLAG,1         ;show busy
      01
0124  FB                        STI                     ;allow other interrupts
0125  2E: 80 3E 0108 R          CMP   CS:PASS,0
      00
012B  75 2C                     JNE   VEC83             ;if beep counter active
012D  50                        PUSH AX                 ;save registers
012E  51                        PUSH CX
012F  52                        PUSH DX
0130  B4 02                     MOV   AH,2              ;get time from BIOS
0132  CD 1A                     INT    1AH
0134  80 FE 00                  CMP   DH,0              ;is it 00 seconds
0137  75 68                     JNE   VEC86             ;not time yet, so return
0139  80 F9 00                  CMP   CL,0              ;test for hour
013C  74 10                     JE    VEC82             ;if hour beep 2 times
013E  80 F9 30                  CMP   CL,30H            ;test for half-hour
0141  75 5E                     JNE   VEC86             ;if not half-hour
0143  E8 0065                   CALL BEEPS              ;start speaker beep
0146  2E: C6 06 0108 R          MOV   CS:PASS,1         ;set number of beeps to 1
      01
014C  EB 53                     JMP   VEC86             ;end it
014E                    VEC82:
014E  E8 005A                   CALL BEEPS              ;start speaker beep
0151  2E: C6 06 0108 R          MOV   CS:PASS,2         ;set number of beeps to 2
      02
0157  EB 48                     JMP   VEC86             ;end it
0159                    VEC83:
0159  2E: 80 3E 0103 R          CMP   CS:COUNT,0        ;test for end of delay
      00
015F  74 07                     JE    VEC84             ;if time delay has elapsed
0161  2E: FE 0E 0103 R          DEC   CS:COUNT
0166  EB 3C                     JMP   VEC88             ;end it
0168                    VEC84:
0168  2E: 80 3E 0109 R          CMP   CS:BEEP,0         ;test beep on
      00
016E  75 1C                     JNE   VEC85             ;if beep is on
0170  2E: FE 0E 0108 R          DEC   CS:PASS           ;test for 2 beeps
```

```
0175  74 2D              JZ   VEC88              ;if second beep not needed
0177  2E: C6 06 0103 R    MOV  CS:COUNT,9         ;reset count
      09
017D  2E: C6 06 0109 R    MOV  CS:BEEP,1          ;beep on for second beep
      01
0183  50                  PUSH AX
0184  E4 61               IN   AL,61H             ;enable speaker for beep
0186  0C 03               OR   AL,3
0188  E6 61               OUT  61H,AL
018A  EB 17               JMP  VEC87              ;end it
018C                VEC85:
018C  2E: C6 06 0103 R    MOV  CS:COUNT,9         ;reset count
      09
0192  2E: C6 06 0109 R    MOV  CS:BEEP,0          ;show beep is off
      00
0198  50                  PUSH AX
0199  E4 61               IN   AL,61H             ;disable speaker
019B  34 03               XOR  AL,3
019D  E6 61               OUT  61H,AL
019F  EB 02               JMP  VEC87              ;end it
01A1                VEC86:
01A1  5A                  POP  DX                 ;restore registers
01A2  59                  POP  CX
01A3                VEC87:
01A3  58                  POP  AX
01A4                VEC88:
01A4  2E: C6 06 010A      MOV  CS:FLAG,0          ;show not busy
      00
01AA  CF                  IRET                    ;interrupt return

01AB                VEC8  ENDP
                    ;
                    ;The BEEPS procedure programs the speaker for the
                    ;frequency stored as TONE using an equate at assembly
                    ;time. The duration of the beep is 1/2 second.
                    ;***uses registers AX, CX, and DX***

01AB                BEEPS PROC NEAR               ;beep speaker

01AB  2E: 8B 0E 03E8      MOV  CX,CS:TONE         ;set tone
01B0  B8 34DC             MOV  AX,34DCH           ;load AX with 1,193,180
01B3  BA 0012             MOV  DX,12H
01B6  F7 F1               DIV  CX                 ;calculate count
01B8  E6 42               OUT  42H,AL             ;program timer 2
01BA  8A C4               MOV  AL,AH
01BC  E6 42               OUT  42H,AL

01BE  E4 61               IN   AL,61H             ;speaker on
01C0  0C 03               OR   AL,3
01C2  E6 61               OUT  61H,AL
01C4  2E: C6 06 0103 R    MOV  CS:COUNT,9         ;set count for 1/2 second
      09
01CA  2E: C6 06 0109 R    MOV  CS:BEEP,1          ;indicate beep is on
      01
01D0  C3                  RET

01D1                BEEPS ENDP

01D1                INSTALL:                      ;install interrupt VEC8
01D1  8C C8               MOV  AX,CS              ;overlap CS and DS
01D3  8E D8               MOV  DS,AX

01D5  B8 3508             MOV  AX,3508H           ;get current vector 8
01D8  CD 21               INT  21H                ;and save it
```

```
01DA  89 1E 0104 R        MOV  WORD PTR ADD8,BX
01DE  8C 06 0106 R        MOV  WORD PTR ADD8+2,ES

01E2  B8 2508             MOV  AX,2508H
01E5  BA 010B R           MOV  DX,OFFSET VEC8       ;address interrupt VEC8
01E8  CD 21               INT  21H                  ;install vector 8

01EA  BA 01D1 R           MOV  DX,OFFSET INSTALL    ;find paragraphs
01ED  B1 04               MOV  CL,4
01EF  D3 EA               SHR  DX,CL
01F1  42                  INC  DX

01F2  B8 3100             MOV  AX,3100H             ;exit to DOS as TSR
01F5  CD 21               INT  21H
                          END
```

The CHIME program uses several memory locations as flags to signal the operation of the interrupt service procedure. The first flag tested by CHIME is the busy flag (FLAG), which indicates that a part of the interrupt service procedure is active. If FLAG = 1 (busy condition), the procedure jumps to the normal vector 8 interrupt (JMP CS:ADD8), which ends VEC8's execution. If FLAG = 0 (not busy), the interrupt service procedure continues at VEC81. The default address for all direct memory data is the data segment. In the TSR software used in this example and others, it is important to use the segment override prefix (CS:) to ensure that the program addresses data in the code segment, where it appears.

At VEC81, the normal vector 8 interrupt is executed with a forced interrupt call (PUSHF followed by a CALL CS:ADD8). Upon return from the normal vector 8 interrupt (required to keep accurate time), the busy flag is set to show a busy condition (FLAG = 1) and other interrupts are enabled with the STI instruction.

The PASS flag is now tested to see if the VEC8 procedure is currently beeping the speaker. If PASS = 0 (not beeping speaker), the time of day is retrieved from BIOS by using the INT 1AH instruction. It is important not to access DOS from within a TSR or interrupt service procedure. If DOS is accessed at this time, it may be in the process of executing an operation that affects the interrupt. This would cause the program to crash. The INT 1AH instruction returns the number of seconds (DH), minutes (CL), and hours (CH) in BCD form. After obtaining the current time, the number of seconds is tested for zero. If it is not zero seconds, the interrupt procedure ends. If it is zero seconds, then CL is tested for 00 minute (hour) and 30 minutes. If either case is true, the speaker is enabled and TONE is programmed in the timer by a call to BEEPS. If neither case is true, the interrupt ends. Notice that the BEEPS procedure programs timer 2, enables the speaker, and sets the count to 9.

The time delay counter (COUNT) is decremented each time the interrupt occurs. If the count reaches zero, the procedure tests BEEP to control the speaker. If the speaker is beeping, the procedure turns it off and resets the time delay count to 9. If the speaker is not beeping, the procedure tests PASS to determine if another beep is required on the hour. The time delay is $^1/_2$ second (COUNT = 9) in this program and cannot be less. If a delay of less than $^1/_2$ second is chosen, the speaker will beep twice, for both the hour and half hour. The reason is that the clock (INT 1AH) is checked for the zero second. If a time delay of less than $^1/_2$ second is used, the half-hour will be picked up twice.

The TSR program is loaded into memory at the DOS command line by typing the name of the program; in this case, the program is called CHIME. If DOS version 5.0 or 6.X is in use, you can load CHIME into the upper memory or high memory area by typing LOADHIGH CHIME. Once this program loads into memory, it remains in the background, beeping off the time until the power to the computer is disconnected or until it is rebooted. This is an excellent, not too annoying addition to the system to keep track of time. The next section of the text describes hot keys. If desired, a hot key could be used to enable and disable CHIME.

Example Hot Key Program

Hot keys are keystrokes that invoke terminate and stay resident programs. For example, an ALT + C key could be defined as a hot key that calls a program that displays the time. Note that the hot key is detected inside most applications, but not at the DOS command line, where it may lock up the system if used. To detect a hot key, we usu-

ally hook into interrupt vector 9, which is the keyboard interrupt that occurs if any key is typed. This allows us to test the keyboard and detect a hot key before the normal interrupt processes the keystroke.

A hot key is installed with a TSR program and an interrupt hook. To illustrate a hot key program that can be useful, a program is developed that counts keystrokes. The keystroke counter program (see Example 7–35) is useful in a business environment that uses computers for data entry or other tasks. With this type of program, productivity can be assessed. The keystroke counter program counts each keystroke and only displays the count when the ALT + K key is pressed. (It is important to note that this program spies on workers and it is the duty of any company using the program to notify the worker. It may even be the responsibility of the company to obtain permission from the worker before a program such as this is placed into service.)

This program can be modified to keep track of keystrokes by the hour or any other time unit. In this example, the keystroke count (up to four billion) accumulates keystrokes for as long as power is applied to the computer. The program also stores the installation time for security purposes. This is important because if a machine is reset, the start time for this TSR will be reset.

This program hooks into interrupt 8 and 9 to count keys. The interrupt 9 hook detects the hot key (ALT + K) and counts keystrokes. When the hot key is detected, the 18.2 Hz interrupt 8 activates the hot key program that displays the keystroke count and time of installation. This type of TSR is often called a *pop-up program* because it pops up when the hot key is typed. Notice that this program uses INT 16H to test the keyboard. Never use a DOS INT 21H function call within a TSR or interrupt hook because serious problems can arise. This program also uses direct manipulation of the video text memory that begins at location B8000H. This memory is organized with two bytes per ASCII character. The first byte contains the ASCII code, and the following byte contains the background and character color.

EXAMPLE 7–35

```
                        ;A TSR program that counts keystrokes and reports the
                        ;time of installation and number of accumulated
                        ;keystrokes when the ALT-K key combination is activated.
                        ;***requires an 80386 or newer microprocessor***
                        ;XX for use on PC XXX
                                .MODEL TINY
                                .386
0000                            .CODE
                                .STARTUP
0100  E9 0241                   JMP  INSTALL            ;install VEC8 and VEC9

0103  00                HFLAG   DB   0                  ;Hot-key detected
0104  00000000          ADD8    DD   ?                  ;old vector 8 address
0108  00000000          ADD9    DD   ?                  ;old vector 9 address
010C  00000000          COUNT   DD   0                  ;Keystroke counter
0110  00                HOUR    DB   ?                  ;start-up time
0111  00                MIN     DB   ?
0112  0 0               SFLAG   DB   0                  ;start-up flag
0113  00                FLAG8   DB   0                  ;interrupt 8 busy
0114  25                KEY     DB   25H                ;scan code for K
0115  08                HMASK   DB   8                  ;alternate key mask
0116  08                MKEY    DB   8                  ;alternate key
0117  00A0 [            SCRN    DB   160 DUP (?)        ;screen buffer
            00
              ]
01B7  54 69 6D 65       MES1    DB   'Time = '
      20 3D 20
01BE  20 20 20 4B       MES2    DB   '   KeyStrokes = '
      65 79 53 74
      72 6F 6B 65
      73 20 3D 20

01CE                    VEC9    PROC FAR                ;keyboard intercept
```

```
01CE  FB                      STI                 ;enable interrupts
01CF  66| 50                  PUSH EAX            ;save EAX
01D1  E4 60                   IN   AL,60H         ;get scan code
01D3  2E: 3A 06 0114 R        CMP  AL,CS:KEY      ;test for K
01D8  75 16                   JNE  VEC91          ;no hot-key
01DA  B8 0000                 MOV  AX,0           ;address segment 0000
01DD  1E                      PUSH DS             ;save DS
01DE  8E D8                   MOV  DS,AX
01E0  A0 0417                 MOV  AL,DS:[417H]   ;get shift/alternate data
01E3  1F                      POP  DS
01E4  2E: 22 06 0115 R        AND  AL,CS:HMASK    ;isolate alternate key
01E9  2E: 3A 06 0116 R        CMP  AL,CS:MKEY     ;test for alternate key
01EE  74 2A                   JE   VEC93          ;if hot-key found
01F0                  VEC91:
01F0  51                      PUSH CX             ;add one to BCD COUNT
01F1  B9 0003                 MOV  CX,3
01F4  66| 2E: A1 010C R       MOV  EAX,CS:COUNT
01F9  66| 83 C0 01           ADD  EAX,1
01FD  27                      DAA                 ;make result BCD
01FE                  VEC92:
01FE  9C                      PUSHF
01FF  66| C1 C8 08           ROR  EAX,8
0203  9D                      POPF
0204  14 00                   ADC  AL,0           ;propagate carry
0206  27                      DAA
0207  E2 F5                   LOOP VEC92
0209  66| C1 C8 08           ROR  EAX,8
020D  66| 2E: A3 010C R       MOV  CS:COUNT,EAX
0212  59                      POP  CX
0213  66| 58                  POP  EAX
0215  2E: FF 2E 0108 R        JMP  CS:ADD9        ;do normal interrupt
021A                  VEC93:                      ;if hot-key pressed
021A  FA                      CLI                 ;interrupts off
021B  E4 61                   IN   AL,61H         ;clear keyboard and
021D  0C 80                   OR   AL,80H         ;throw away hot key
021F  E6 61                   OUT  61H,AL
0221  24 7F                   AND  AL,7FH
0223  E6 61                   OUT  61H,AL
0225  B0 20                   MOV  AL,20H         ;reset keyboard interrupt
0227  E6 20                   OUT  20H,AL
0229  FB                      STI                 ;enable interrupts
022A  2E: C6 06 0103 R        MOV  CS:HFLAG,1     ;indicate hot-key pressed
      01
0230  66| 58                  POP  EAX
0232  CF                      IRET

0233                  VEC9    ENDP

0233                  VEC8    PROC FAR            ;clock tick interrupt

0233  2E: 80 3E 0113 R        CMP  CS:FLAG8,0
      00
0239  74 05                   JZ   VEC81          ;if not busy
023B  2E: FF 2E 0104 R        JMP  CS:ADD8        ;if busy
0240                  VEC81:
0240  2E: 80 3E 0103 R        CMP  CS:HFLAG,0
      00
0246  75 37                   JNZ  VEC83          ;if hot-key detected
0248  2E: 80 3E 0112 R        CMP  CS:SFLAG,0
      00
024E  74 05                   JZ   VEC82          ;if start-up
0250  2E: FF 2E 0104 R        JMP  CS:ADD8        ;if not hot-key or start
0255                  VEC82:
```

```
0255  9C                    PUSHF                ;do old interrupt 8
0256  2E: FF 1E 0104 R      CALL CS:ADD8
025B  2E: C6 06 0113 R      MOV  CS:FLAG8,1      ;indicate busy
      01
0261  FB                    STI                  ;enable interrupts
0262  50                    PUSH AX
0263  51                    PUSH CX
0264  52                    PUSH DX
0265  B4 02                 MOV  AH,2            ;get start-up time
0267  CD 1A                 INT  1AH
0269  2E: 88 2E 0110 R      MOV  CS:HOUR,CH      ;save hour
026E  2E: 88 0E 0111 R      MOV  CS:MIN,CL       ;save minute
0273  5A                    POP  DX              ;restore registers
0274  59                    POP  CX
0275  58                    POP  AX
0276  2E: C6 06 0112 R      MOV  CS:SFLAG,1      ;indicate started
      01
027C  E9 00A5               JMP  VEC89           ;end it
027F                VEC83:                       ;do hot-key display
027F  9C                    PUSHF                ;do old interrupt 8
0280  2E: FF 1E 0104 R      CALL CS:ADD8
0285  2E: C6 06 0113 R      MOV  CS:FLAG8,1      ;indicate busy
      01
028B  FB                    STI                  ;enable interrupts
028C  50                    PUSH AX              ;save registers
028D  53                    PUSH BX
028E  B4 0F                 MOV  AH,0FH          ;get video mode
0290  CD 10                 INT  10H
0292  3C 03                 CMP  AL,3
0294  76 05                 JBE  VEC84           ;if DOS text mode
0296  5B                    POP  BX              ;ignore if graphics mode
0297  58                    POP  AX
0298  E9 0083               JMP  VEC88
029B                VEC84:                       ;for text mode
029B  51                    PUSH CX
029C  66| 52                PUSH EDX
029E  57                    PUSH DI
029F  56                    PUSH SI
02A0  1E                    PUSH DS
02A1  06                    PUSH ES
02A2  FC                    CLD
02A3  8C C8                 MOV  AX,CS           ;address this segment
02A5  8E C0                 MOV  ES,AX
02A7  B8 B800               MOV  AX,0B800H       ;address text memory
02AA  8E D8                 MOV  DS,AX
02AC  B9 00A0               MOV  CX,160          ;save top screen line
02AF  BF 0117 R             MOV  DI,OFFSET SCRN
02B2  BE 0000               MOV  SI,0
02B5  F3/ A4                REP  MOVSB
02B7  1E                    PUSH DS              ;swap segments
02B8  06                    PUSH ES
02B9  1F                    POP  DS
02BA  07                    POP  ES
02BB  BF 0050               MOV  DI,80           ;start display at center
02BE  BE 01B7 R             MOV  SI,OFFSET MES1
02C1  B4 0F                 MOV  AH,0FH          ;load white on black
02C3  B9 0007               MOV  CX,7
02C6                VEC85:
02C6  AC                    LODSB                ;display "Time = "
02C7  AB                    STOSW
02C8  E2 FC                 LOOP VEC85
02CA  2E: 8A 16 0111 R      MOV  DL,CS:MIN
02CF  2E: 8A 36 0110 R      MOV  DH,CS:HOUR
```

```
02D4  66| C1 E2 10          SHL  EDX,16
02D8  B9 0002               MOV  CX,2
02DB  B3 30                 MOV  BL,30H
02DD  E8 004B               CALL DISP              ;display hours
02E0  B0 3A                 MOV  AL,':'
02E2  AB                    STOSW                  ;display colon
02E3  B9 0002               MOV  CX,2
02E6  B3 80                 MOV  BL,80H
02E8  E8 0040               CALL DISP              ;display minutes
02EB  BE 01BE R             MOV  SI,OFFSET MES2    ;display KeyStrokes =
02EE  B9 0010               MOV  CX,16
02F1                VEC86:
02F1  AC                    LODSB
02F2  AB                    STOSW
02F3  E2 FC                 LOOP VEC86
02F5  66| 2E: 8B 16 010C R MOV  EDX,CS:COUNT      ;get count
02FB  B9 0008               MOV  CX,8
02FE  B3 30                 MOV  BL,30H
0300  E8 0028               CALL DISP              ;display count
0303                VEC87:
0303  B4 01                 MOV  AH,1              ;wait for any key (BIOS)
0305  CD 16                 INT  16H
0307  74 FA                 JZ   VEC87
0309  FC                    CLD
030A  BE 0117 R             MOV  SI,OFFSET SCRN    ;restore text
030D  BF 0000               MOV  DI,0
0310  B9 00A0               MOV  CX,160
0313  F3/ A4                REP  MOVSB
0315  07                    POP  ES
0316  1F                    POP  DS
0317  5E                    POP  SI
0318  5F                    POP  DI
0319  66| 5A                POP  EDX
031B  59                    POP  CX
031C  5B                    POP  BX
031D  58                    POP  AX
031E                VEC88:
031E  2E: C6 06 0103 R      MOV  CS:HFLAG,0        ;kill hot-key
      00
0324                VEC89:
0324  2E: C6 06 0113 R      MOV  CS:FLAG8,0        ;indicate not busy
      00
032A  CF                    IRET

032B                VEC8    ENDP
                    ;
                    ;The DISP procedure displays the BCD contents of EDX.
                    ;***input parameters***
                    ;CX = number of digits
                    ;BL = 30H for blank leading zeros or 80H for no blanking
                    ;ES = segment address of text mode display
                    ;DI = offset address of text mode display
                    ;
032B                DISP    PROC NEAR              ;display

032B  66| C1 C2 04         ROL  EDX,4             ;position number
032F  8A C2                 MOV  AL,DL
0331  24 0F                 AND  AL,0FH
0333  04 30                 ADD  AL,30H            ;convert to ASCII
0335  AB                    STOSW                  ;store in text display
0336  3A C3                 CMP  AL,BL             ;test for blanking
0338  74 04                 JE   DISP1             ;if blanking needed
033A  B3 80                 MOV  BL,80H            ;turn off blanking
```

```
033C  EB 03                JMP  DISP2            ;continue
033E                DISP1:
033E  83 EF 02             SUB  DI,2             ;blank digit
0341                DISP2:
0341  E2 E8                LOOP DISP
0343  C3                   RET

0344                DISP   ENDP

0344                INSTALL:                     ;install VEC8 and VEC9

0344  8C C8                MOV  AX,CS            ;load DS
0346  8E D8                MOV  DS,AX

0348  B8 3508              MOV  AX,3508H         ;get current vector 8
034B  CD 21                INT  21H              ;and save it
034D  89 1E 0104 R         MOV  WORD PTR ADD8,BX
0351  8C 06 0106 R         MOV  WORD PTR ADD8+2,ES

0355  B8 3509              MOV  AX,3509H         ;get current vector 9
0358  CD 21                INT  21H              ;and save it
035A  89 1E 0108 R         MOV  WORD PTR ADD9,BX
035E  8C 06 010A R         MOV  WORD PTR ADD9+2,ES

0362  B8 2508              MOV  AX,2508H
0365  BA 0233 R            MOV  DX,OFFSET VEC8   ;address interrupt procedure
0368  CD 21                INT  21H              ;install vector 8

036A  B8 2509              MOV  AX,2509H
036D  BA 01CE R            MOV  DX,OFFSET VEC9   ;address interrupt procedure
0370  CD 21                INT  21H              ;install vector 9

0372  BA 0344 R            MOV  DX,OFFSET INSTALL ;find paragraphs
0375  C1 EA 04             SHR  DX,4
0378  42                  INC  DX

0379  B8 3100              MOV  AX,3100H         ;set as a TSR
037C  CD 21                INT  21H
                           END
```

Note that the pop-up portion of this program only functions in the text mode and will count any unseen keystrokes that DOS generates. It also counts shift, alternate, and other keys as they are pressed and released. For example, the capital A will be counted as two or three keystrokes. This means that the count will be inflated. Even so, this program is useful for counting keystrokes by a given operator. If the operator reboots the system, the new reboot time is displayed and the count is cleared to zero.

The VEC9 interrupt service procedure intercepts all keystrokes. The IN AL,60H instruction reads the scan code from the keyboard interface within the personal computer. This is then tested for the K scan code. (Refer to Table 7–3 for the key scan codes.) If the K scan code is not found, the procedure increments the BCD count stored at location COUNT and returns to the normal keyboard interrupt handler. If the K scan code is detected, the contents of memory location 0000:0417 are tested for the alternate key. If an alternate key is detected, the program sets the HFLAG to 1, tosses away the hot key, and returns. Notice how the hot key is discarded by strobing I/O port number 61H. The keyboard is cleared by sending a logic 1 in bit position 7 of port 61H, followed by sending a logic 0 in bit position 7. The interrupt controller in the computer must also be cleared by sending a 20H out to I/O port number 20H.

The VEC8 interrupt service procedure tests the HFLAG for the hot key and the SFLAG for system startup. If the SFLAG = 0, the system has just been installed and the time is stored in HOUR and MIN. If the HFLAG = 1, a hot key was detected by VEC9. The VEC8 procedure responds to the hot key by storing the contents of the top line of the text display at memory array SCRN. Once the top line of the text display is stored, the message "Time = " is displayed, followed by the installation time. Next, the message "KeyStrokes = " is displayed, followed by the BCD number stored in COUNT. Recall that count is incremented each time VEC9 detects that a key is typed on the keyboard.

7–5 SUMMARY

1. The assembler program assembles modules that contain PUBLIC variables and segments, plus EXTRN (external) variables. The linker program links modules and library files to create a run-time program executed from the DOS command line. The run-time program usually has the extension EXE.
2. The MACRO and ENDM directives create a new opcode for use in programs. These macros are similar to procedures, except that there is no call or return. In place of them, the assembler inserts the code of the macro sequence into a program each time it is invoked. Macros can include variables that pass information and data to the macro sequence.
3. The DOS INT 21H function call provides a method of using the keyboard and video display. Function number 06H, placed into register AH, provides an interface to the keyboard and display. If DL = 0FFH, this function tests the keyboard for a keystroke. If no keystroke is detected, it returns equal. If a keystroke is detected, the standard ASCII character returns in AL. If an extended ASCII character is typed, it returns with AL = 00H, where the function must again be called to return with the extended ASCII character in AL. To display a character, DL is loaded with the character and AH with 06H before the INT 21H is used in a program.
4. Character strings are displayed by using function number 09H. The DS:DX register combination addresses the character string, which must end with a $.
5. The INT 10H instruction accesses video BIOS (basic I/O system) procedures that control the video display and keyboard. The video BIOS functions are independent of DOS and function with any operating system.
6. The mouse driver is installed at interrupt vector 33H.
7. Data conversion from binary to BCD is accomplished with the AAM instruction for numbers that are less than 100 or by repeated division by 10 for larger numbers. Once converted to BCD, a 30H is added to convert each digit to ASCII code for the video display.
8. When converting from an ASCII number to BCD, a 30H is subtracted from each digit. To obtain the binary equivalent, we multiply by 10.
9. Lookup tables are used for code conversion with the XLAT instruction if the code is an eight-bit code. If the code is wider than eight bits, a short procedure that accesses a lookup table provides the conversion. Lookup tables are also used to hold addresses so that different parts of a program or different procedures can be selected.
10. Interrupt hooks allow application software to gain access to or intercept an interrupt. We often hook into the timer click interrupt (vector 8) or the keyboard interrupt (vector 9).
11. A terminate and stay resident (TSR) program is a program that remains in the memory and is often accessed through a hooked interrupt, using either the timer click or a hot key.
12. A hot key is a key that activates a terminate and stay resident program through the keyboard interrupt hook.

7–6 QUESTIONS AND PROBLEMS

1. The assembler converts a source file to a(n) _____ file.
2. What files are generated from the source file TEST.ASM if it is processed by MASM?
3. The linker program links object files and _____ files to create an execution file.
4. What does the PUBLIC directive indicate when placed in a program module?
5. What does the EXTRN directive indicate when placed in a program module?
6. What directives appear with labels defined external?

7. Describe how a library file works when it is linked to other object files by the linker program.
8. What assembler language directives delineate a macro sequence?
9. What is a macro sequence?
10. How are parameters transferred to a macro sequence?
11. Develop a macro called ADD32 that adds the 32-bit contents of DX–CX to the 32-bit contents of BX–AX.
12. How is the LOCAL directive used within a macro sequence?
13. Develop a macro called ADDLIST PARA1,PARA2 that adds the contents of PARA1 to PARA2. Each of these parameters represents an area of memory. The number of bytes added are indicated by register CX before the macro is invoked.
14. Develop a macro that sums a list of byte-sized data invoked by the macro ADDM LIST,LENGTH. The label LIST is the starting address of the data block and length is the number of data added. The result must be a 16-bit sum found in AX at the end of the macro sequence.
15. What is the purpose of the INCLUDE directive?
16. Develop a procedure called RANDOM. This procedure must return an eight-bit random number in register CL at the end of the subroutine. (One way to generate a random number is to increment CL each time the DOS function 06H tests the keyboard and finds *no* keystroke. In this way, a random number is generated.)
17. Develop a macro that uses the REPEAT statement to insert 10 NOP instructions in a program.
18. Develop a macro that uses the IFB/IFNB statements to test the parameter PARA in the macro DISP MACRO PARA. If PARA is blank, display a carriage return/line feed combination. If PARA is not blank, display PARA as an ASCII-coded character.
19. Develop a procedure that displays a character string that ends with a 00H. Your procedure must use the DS:DX register to address the start of the character string.
20. Develop a procedure that reads a key and displays the hexadecimal value of an extended ASCII-coded keyboard character if it is typed. If a normal character is typed, ignore it.
21. Use BIOS INT 10H to develop a procedure that positions the cursor at line 3, column 6.
22. When a number is converted from binary to BCD, the _____ instruction accomplishes the conversion, provided the number is less than 100 decimal.
23. How is a large number (over 100 decimal) converted from binary to BCD?
24. A BCD digit is converted to ASCII code by adding a(n) _____.
25. An ASCII-coded number is converted to BCD by subtracting _____.
26. Develop a procedure that reads an ASCII number from the keyboard and stores it as a BCD number into memory array DATA. The number ends when anything other than a number is typed.
27. Explain how a three-digit ASCII-coded number is converted to binary.
28. Develop a procedure that converts all lowercase ASCII-coded letters into uppercase ASCII-coded letters. Your procedure may not change any other character except the letters a–z.
29. Develop a lookup table that converts hexadecimal data 00H–0FH into the ASCII-coded characters that represent the hexadecimal digits. Make sure to show the lookup table and any software required for the conversion.
30. Develop a program sequence that jumps to memory location ONE if AL = 6, TWO if AL = 7, and THREE if AL = 8.
31. Show how to use the XLAT instruction to access a lookup table called LOOK that is located in the stack segment.
32. Develop a short sequence of instructions that place the line MOV AL,6 into a program if the contents memory location BED are true. You must use the IF statement.
33. Write a program that displays the binary powers of 2 (in decimal) on the video screen for the powers 0 through 7. Your display shows 2^n = value for each power of 2.

34. Using the technique discussed in question number 16, develop a program that displays random numbers between 1 and 47 (or whatever) for your state's lottery.
35. Develop a program the hooks into interrupt vector 0 to display the following message on a divide error: "Oops, you have attempted to divide by 0."

CHAPTER 8

8086/8088 Hardware Specifications

INTRODUCTION

In this chapter, we describe the pin functions of both the 8086 and 8088 microprocessors and provide details on the following hardware topics: clock generation, bus buffering, bus latching, timing, wait states, and minimum mode operation versus maximum mode operation. The simple microprocessors are explained first, because of their simple structures, as an introduction to the Intel microprocessor family.

Before it is possible to connect or interface anything to the microprocessor, it is necessary to understand the pin functions and timing. Thus, the information in this chapter is essential to a complete understanding of memory and I/O interfacing, which we cover in the later chapters of the text.

CHAPTER OBJECTIVES

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

1. Describe the function of each 8086 and 8088 pin.
2. Understand the microprocessor's DC characteristics and indicate its fan-out to common logic families.
3. Use the clock generator chip (8284A) to provide the clock for the microprocessor.
4. Connect buffers and latches to the buses.
5. Interpret the timing diagrams.
6. Describe wait states and connect the circuitry required to cause various numbers waits.
7. Explain the difference between minimum and maximum mode operation.

8–1 PIN-OUTS AND THE PIN FUNCTIONS

In this section, we explain the function and (in certain instances) the multiple functions of each of the microprocessor's pins. In addition, we discuss the DC characteristics to provide a basis for understanding the later sections on buffering and latching.

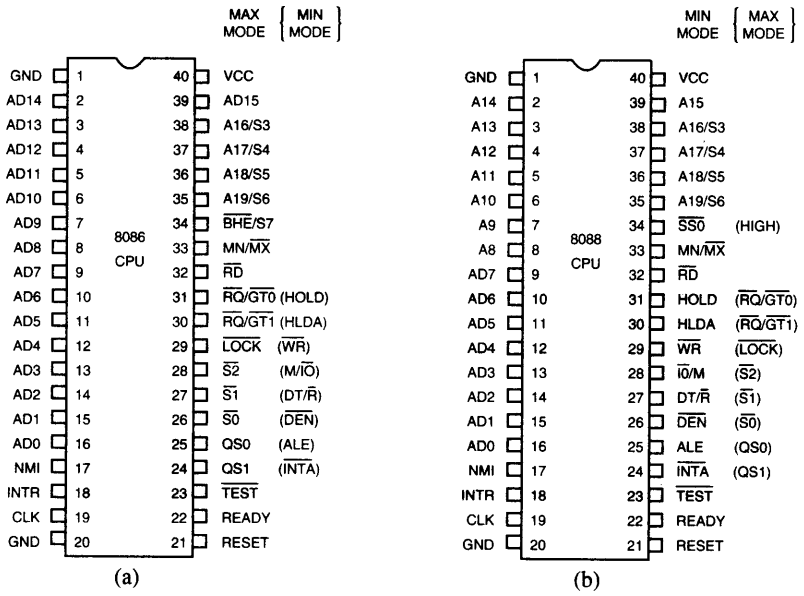

FIGURE 8–1 (a) The pin-out of the 8086 microprocessor; (b) the pin-out of the 8088 microprocessor.

The Pin-Out

Figure 8–1 illustrates the pin-outs of the 8086 and 8088 microprocessors. As a close comparison reveals, there is virtually no difference between these two microprocessors—both are packaged in 40-pin **dual in-line packages** (DIPs).

As mentioned in Chapter 1, the 8086 is a 16-bit microprocessor with a 16-bit data bus, and the 8088 is a 16-bit microprocessor with an 8-bit data bus. (As the pin-outs show, the 8086 has pin connections AD0–AD15, and the 8088 has pin connections AD0–AD7.) Data bus width is therefore the only major difference between these microprocessors.

There is, however, a minor difference in one of the control signals. The 8086 has an M/IO pin, and the 8088 has an IO/M pin. The only other hardware difference appears on Pin 34 of both chips: on the 8088, it is an SSO pin, while on the 8086, it is a BHE/S7 pin.

Power Supply Requirements

Both the 8086 and 8088 microprocessors require +5.0 V with a supply voltage tolerance of ±10 percent. The 8086 uses a maximum supply current of 360 mA, and the 8088 draws a maximum of 340 mA. Both microprocessors operate in ambient temperatures of between 32° F and about 180° F. This range is not wide enough to be used outdoors in the winter or even in the summer, but extended temperature-range versions of the 8086 and 8088 microprocessors are available. There is also a CMOS version, which requires a very low supply current and has an extended temperature range. The 80C88 and 80C86 are CMOS versions that require only 10 mA of power supply current and function in temperature extremes of –40° F through +225° F.

DC Characteristics

It is impossible to connect anything to the pins of the microprocessor without knowing the input current requirement for an input pin and the output current drive capability for an output pin. This knowledge allows the hardware designer to select the proper interface components for use with the microprocessor without the fear of damaging anything.

TABLE 8–1 Input characteristics of the 8086 and 8088 microprocessors.

Logic Level	*Voltage*	*Current*
0	0.8 V maximum	±10 μA maximum
1	2.0 V minimum	±10 μA maximum

TABLE 8–2 Output characteristics of the 8086 and 8088 microprocessors.

Logic Level	*Voltage*	*Current*
0	0.45 V maximum	2.0 mA maximum
1	2.4 V minimum	–400 μA maximum

Input Characteristics. The input characteristics of these microprocessors are compatible with all the standard logic components available today. Table 8–1 depicts the input voltage levels and the input current requirements for any input pin on either microprocessor. The input current levels are very small because the inputs are the gates connections of MOSFETs and represent only leakage currents.

Output Characteristics. Table 8-2 illustrates the output characteristics of all the output pins of these microprocessors. The logic 1 voltage level of the 8086/8088 is compatible with that of most standard logic families, but the logic 0 level is not. Standard logic circuits have a maximum logic 0 voltage of 0.4 V, and the 8086/8088 has a maximum of 0.45 V. Thus, there is a difference of 0.05 V.

This difference reduces the noise immunity from a standard level of 400 mV (0.8 V – 0.45 V) to 350 mV. (The **noise immunity** is the difference between the logic 0 output voltage and the logic 0 input voltage levels.) This reduced noise immunity may result in problems with long wire connections or too many loads. It is therefore recommended that no more than 10 loads of any type or combination be connected to an output pin without buffering. If this loading factor is exceeded, noise will begin to take its toll in timing problems.

Table 8–3 lists some of the more common logic families and the recommended fan-out from the 8086/8088. The best choice of component types for the connection to an 8086/8088 output pin is a LS, 74ALS, or 74HC logic component. Note that some of the fan-out currents calculate to more than 10 unit loads. It is therefore recommended that if a fan-out of more than 10 unit loads is required, the system should be buffered.

Pin Connections

AD7–AD0 The 8088 **address/data bus** lines compose the multiplexed address data bus of the 8088 and contain the rightmost eight bits of the memory address or I/O port number whenever ALE is active (logic 1) or data whenever ALE is active (logic 0). These pins are at their high-impedance state during a hold acknowledge.

A15–A8 The 8088 **address bus** provides the upper-half memory address bits that are present throughout a bus cycle. These address connections go to their high-impedance state during a hold acknowledge.

AD15–AD8 The 8086 **address/data bus** lines compose the upper multiplexed address/data bus on the 8086. These lines contain address bits A15–A8 whenever ALE is a logic 1, and data bus connections D15–D8. These pins enter a high-impedance state whenever a hold acknowledge occurs.

TABLE 8–3 Recommended fan-out from any 8086/8088 pin connection.

Family	*Sink Current*	*Source Current*	*Fan-out*
TTL (74)	–1.6 mA	40 μA	1
TTL (74LS)	–0.4 mA	20 μA	5
TTL (74S)	–2.0 mA	50 μA	1
TTL (74ALS)	–0.1 mA	20 μA	10
TTL (74AS)	–0.5 mA	25 μA	10
TTL (74F)	–0.5 mA	25 μA	10
CMOS (74HC)	–10 μA	10 μA	10
CMOS (CD4)	–10 μA	10 μA	10
NMOS	–10 μ	10 μA	10

A19/S6–A16/S3 The **address/status bus** bits are multiplexed to provide address signals A19–A16 and also status bits S6–S3. These pins also attain a high-impedance state during the hold acknowledge.

Status bit S6 always remains a logic 0, bit S5 indicates the condition of the IF flag bits, and S4 and S3 show which segment is accessed during the current bus cycle. See Table 8–4 for the truth table of S4 and S3. These two status bits can be used to address four separate 1M byte memory banks by decoding them as A21 and A20.

TABLE 8–4 Function of status bits S3 and S4.

S4	*S3*	*Function*
0	0	Extra segment
0	1	Stack segment
1	0	Code or no segment
1	1	Data segment

RD Whenever the **read signal** is a logic 0, the data bus is receptive to data from the memory or I/O devices connected to the system. This pin floats to its high-impedance state during a hold acknowledge.

READY This input is controlled to insert wait states into the timing of the microprocessor. If the READY pin is placed at a logic 0 level, the microprocessor enters into wait states and remains idle. If the READY pin is placed at a logic 1 level, it has no effect on the operation of the microprocessor.

INTR **Interrupt request** is used to request a hardware interrupt. If INTR is held high when IF = 1, the 8086/8088 enters an interrupt acknowledge cycle (INTA becomes active) after the current instruction has complete execution.

TEST The **Test** pin is an input that is tested by the WAIT instruction. If TEST is a logic 0, the WAIT instruction functions as a NOP. If TEST is a logic 1, the WAIT instruction waits for TEST to become a logic 0. This pin is most often connected to the 8087 numeric coprocessor.

NMI The **non-maskable interrupt** input is similar to INTR except that the NMI interrupt does not check to see whether the IF flag bit is a logic 1. If NMI is activated, this interrupt input uses interrupt vector 2.

RESET The **reset** input causes the microprocessor to reset itself if this pin is held high for a minimum of four clocking periods. Whenever the 8086 or 8088 is reset, it begins executing instructions at memory location FFFF0H and disables future interrupts by clearing the IF flag bit.

CLK The **clock pin** provides the basic timing signal to the microprocessor. The clock signal must have a duty cycle of 33 percent (high for one-third of the clocking period and low for two-thirds) to provide proper internal timing for the 8086/8088.

VCC This **power supply** input provides a +5.0 V, ±10 % signal to the microprocessor.

GND The **ground** connection is the return for the power supply. Note that the 8086/8088 microprocessors have two pins labeled GND—both must be connected to ground for proper operation.

MN/MX The **minimum/maximum mode** pin selects either minimum mode or maximum mode operation for the microprocessor. If minimum mode is selected, the MN/MX pin must be connected directly to +5.0 V.

BHE/S7 The **bus high enable** pin is used in the 8086 to enable the most-significant data bus bits (D15–D8) during a read or a write operation. The state of S7 is always a logic 1.

Minimum Mode Pins. Minimum mode operation of the 8086/8088 is obtained by connecting the MN/MX pin diretly to +5.0 V. Do not connect this pin to +5.0 V through a pull-up resistor or it will not function correctly.

IO/M or M/IO The IO/M (8088) or the M/IO (8086) pin selects memory or I/O. This pin indicates that the microprocessor address bus contains either a memory address or an I/O port address. This pin is at its high-impedance state during a hold acknowledge.

WR The **write line** is a strobe that indicates that the 8086/8088 is outputting data to a memory or I/O device. During the time that the WR is a logic 0, the data bus contains valid data for memory or I/O. This pin floats to a high-impedance during a hold acknowledge.

INTA The **interrupt acknowledge** signal is a response to the INTR input pin. The INTA pin is normally used to gate the interrupt vector number onto the data bus in response to an interrupt request.

ALE **Address latch enable** shows that the 8086/8088 address/data bus contains address information. This address can be a memory address or an I/O port number. Note that the ALE signal does not float during a hold acknowledge.

DT/R The **data transmit/receive** signal shows that the microprocessor data bus is transmitting (DT/R = 1) or receiving (DT/R = 0) data. This signal is used to enable external data bus buffers.

DEN **Data bus enable** activates external data bus buffers.

HOLD The **hold** input requests a direct memory access (DMA). If the HOLD signal is a logic 1, the microprocessor stops executing software and places its address, data, and control bus at the high-impedance state. If the HOLD pin is a logic 0, the microprocessor executes software normally.

HLDA **Hold acknowledge** indicates that the 8086/8088 has entered the hold state

SS0 The SS0 status line is equivalent to the S0 pin in maximum mode operation of the microprocessor. This signal is combined with IO/M and DT/R to decode the function of the current bus cycle (see Table 8–5)

Maximum Mode Pins. In order to achieve maximum mode for use with external coprocessors, connect the MN/MX pin to ground.

S2, S1, and S0 The status bits indicate the function of the current bus cycle. These signals are normally decoded by the 8288 bus controller described later in this chapter. Table 8–6 shows the function of these three status bits in the maximum mode.

TABLE 8–5 Bus cycle status (8088) using SS0.

IO/M	*DT/R*	*SS0*	*Function*
0	0	0	Interrupt acknowledge
0	0	1	Memory read
0	1	0	Memory write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	I/O read
1	1	0	I/O write
1	1	1	Passive

TABLE 8–6 Bus control functions generated by the bus controller (8288) using S2, S1, and S0.

S2	*S1*	*S0*	*Function*
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

TABLE 8–7 Queue status bits.

QS1	*QS0*	*Function*
0	0	Queue is idle
0	1	First byte of opcode
1	0	Queue is empty
1	1	Subsequent byte of opcode

RO/GT1 and RO/GT0 The **request/grant** pins request direct memory accesses (DMA) during maximum mode operation. These lines are bi-directional, and are used to both request and grant a DMA operation.

LOCK The **lock output** is used to lock peripherals off the system. This pin is activated by using the LOCK: prefix on any instruction.

QS1 and QS0 The **queue status** bits show the status of the internal instruction queue. These pins are provided for access by the numeric coprocessor (8087). See Table 8–7 for the operation of the queue status bits.

8–2 CLOCK GENERATOR (8284A)

This section describes the clock generator (8284A), the RESET signal, and introduces the READY signal for the 8086/8088 microprocessors. (The READY signal and its associated circuitry are treated in detail in Section 8–5.)

The 8284A Clock Generator

The 8284A is an ancillary component to the 8086/8088 microprocessors. Without the clock generator, many additional circuits are required to generate the clock (CLK) in an 8086/8088-based system. The 8284A provides the

following basic functions or signals: clock generation, RESET synchronization, READY synchronization, and a TTL-level peripheral clock signal. Figure 8–2 illustrates the pin-out of the 8284A clock generator.

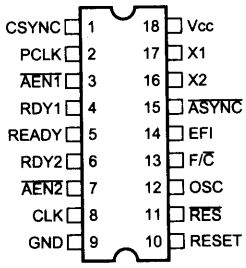

FIGURE 8–2 The pin-out of the 8284A clock generator.

Pin Functions. The 8284A is an 18-pin integrated circuit, designed specifically for use with the 8086/8088 microprocessors. The following is a list of each pin and its function:

AEN1 and AEN2 — The **address enable** pins are provided to qualify the bus ready signals, RDY1 and RDY2, respectively. Section 8–5 illustrates the use of these two pins, which are used to cause wait states, along with the RDY1 and RDY2 inputs. Wait states are generated by the READY pin of the 8086/8088 microprocessors, which is controlled by these two inputs.

RDY1 and RDY2 — The **bus ready** inputs are provided, in conjunction with the AEN1 and AEN2 pins, to cause wait states in an 8086/8088-based system.

ASYNC — The **ready synchronization** selection input selects either one or two stages of synchronization for the RDY1 and RDY2 inputs.

READY — **Ready** is an output pin that connects to the 8086/8088 READY input. This signal is synchronized with the RDY1 and RDY2 inputs.

X1 and X2 — The **crystal oscillator** pins connect to an external crystal used as the timing source for the clock generator and all its functions.

F/C — The **frequency/crystal select** input chooses the clocking source for the 8284A. If this pin is held high, an external clock is provided to the EFI input pin; if it is held low, the internal crystal oscillator provides the timing signal.

EFI — The **external frequency input** is used when the F/C pin is pulled high. EFI supplies the timing whenever the F/C pin is high.

CLK — The **clock output** pin provides the CLK input signal to the 8086/8088 microprocessors and other components in the system. The CLK pin has an output signal that is one-third of the crystal or EFI input frequency, and has a 33-percent duty cycle, which is required by the 8086/8088.

PCLK — The **peripheral clock** signal is one-sixth the crystal or EFI input frequency, and has a 50-percent duty cycle. The PCLK output provides a clock signal to the peripheral equipment in the system.

OSC — The **oscillator output** is a TTL-level signal that is at the same frequency as the crystal or EFI input. The OSC output provides an EFI input to other 8284A clock generators in some multiple-processor systems.

RES — The **reset input** is an active-low input to the 8284A. The RES pin is often connected to an RC network that provides power-on resetting.

RESET — The **reset output** is connected to the 8086/8088 RESET input pin.

CSYNC — The **clock synchronization** pin is used whenever the EFI input provides synchronization in systems with multiple processors. If the internal crystal oscillator is used, this pin must be grounded.

GND — The **ground** pin connects to ground.

Vcc — This **power supply** pin connects to +5.0 V with a tolerance of ±10 percent.

Operation of the 8284A

The 8284A is a relatively easy component to understand. Figure 8–3 illustrates the internal block diagram of the 8284A clock generator.

Operation of the Clock Section. The top half of the logic diagram represents the clock and reset synchronization section of the 8284A clock generator. As the diagram shows, the crystal oscillator has two inputs: X1 and X2. If a crystal is attached to X1 and X2, the oscillator generates a square-wave signal at the same frequency as the crystal. The square-wave signal is fed to an AND gate and also to an inverting buffer that provides the OSC output signal. The OSC signal is sometimes used as an EFI input to other 8284A circuits in a system.

An inspection of the AND gate reveals that when F/C is a logic 0, the oscillator output is steered through to the divide-by-3 counter. If F/C is a logic 1, then EFI is steered through to the counter.

The output of the divide-by-3 counter generates the timing for ready synchronization, a signal for another counter (divide-by-2), and the CLK signal to the 8086/8088 microprocessors. The CLK signal is also buffered before it leaves the clock generator. Notice that the output of the first counter feeds the second. These two cascaded counters provide the divide-by-6 output at PCLK, the peripheral clock output.

Figure 8–4 shows how an 8284A is connected to the 8086/8088. Notice that F/C and CSYNC are grounded to select the crystal oscillator; and that a 15 MHz crystal provides the normal 5 MHz clock signal to the 8086/8088, as well as a 2.5 MHz peripheral clock signal.

Operation of the Reset Section. The reset section of the 8284A is very simple: It consists of a Schmitt trigger buffer and a single D-type flip-flop circuit. The D-type flip-flop ensures that the timing requirements of the 8086/8088 RESET input are met. This circuit applies the RESET signal to the microprocessor on the negative edge (1-to-0 transition) of each clock. The 8086/8088 microprocessors sample RESET at the positive edge (0-to-1 transition) of the clocks; therefore, this circuit meets the timing requirements of the 8086/8088.

Refer to Figure 8–4. Notice that an RC circuit provides a logic 0 to the RES input pin when power is first applied to the system. After a short time, the RES input becomes a logic 1 because the capacitor charges toward +5.0 V through the resistor. A push-button switch allows the microprocessor to be reset by the operator. Correct reset

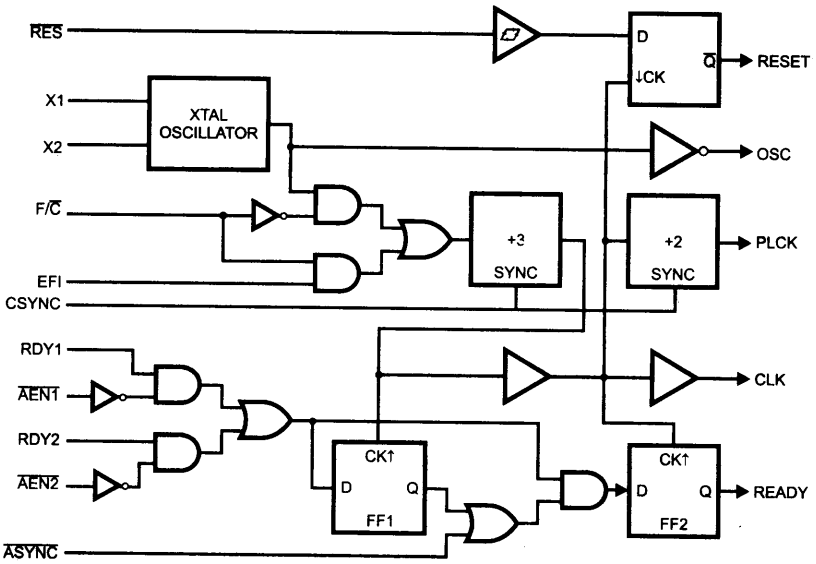

FIGURE 8–3 The internal block diagram of the 8284A clock generator.

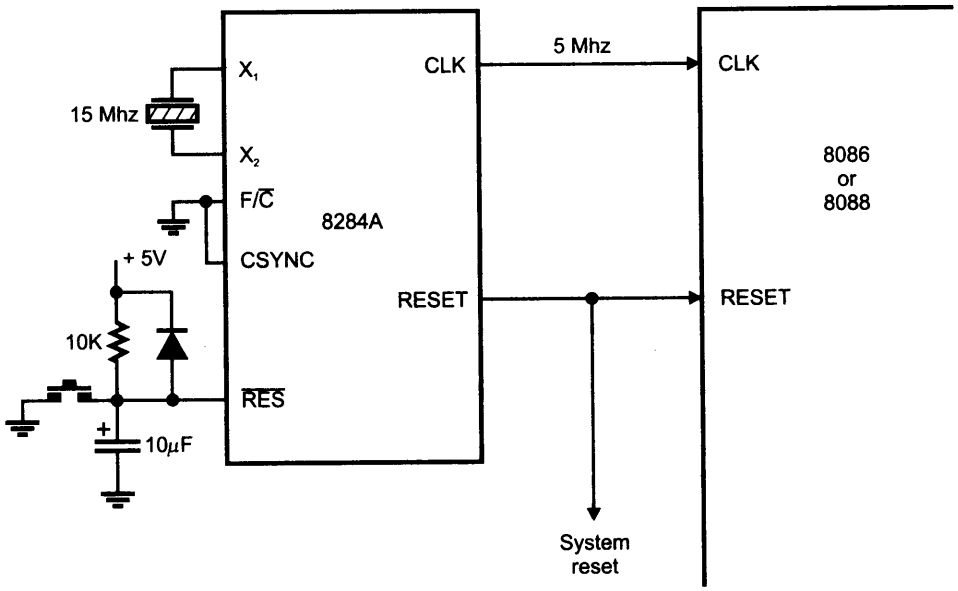

FIGURE 8–4 The clock generator (8284A) and the 8086 and 8088 microprocessor illustrating the connection for the clock and reset signals. A 15 MHz crystal provides the 5 MHz clock for the microprocessor.

timing requires the RESET input to become a logic 1 no later than four clocks after system power is applied, and to be held high for at least 50 μs. The flip-flop makes certain that RESET goes high in four clocks, and the RC time constant ensures that it stays high for at least 50 μs.

8–3 BUS BUFFERING AND LATCHING

Before the 8086/8088 microprocessors can be used with memory or I/O interfaces, their multiplexed buses must be demultiplexed. This section provides the detail required to demultiplex the buses and illustrates how the buses are buffered for very large systems. (Because the maximum fan-out is 10, the system must be buffered if it contains more than 10 other components.)

Demultiplexing the Buses

The address/data bus on the 8086/8088 is multiplexed (**shared**) to reduce the number of pins required for the 8086/8088 microprocessor integrated circuit. Unfortunately, this burdens the hardware designer with the task of extracting or demultiplexing information from these multiplexed pins.

Why not leave the buses multiplexed? Memory and I/O require that the address remains valid and stable throughout a read or write cycle. If the buses are multiplexed, the address changes at the memory and I/O, which causes them to read or write data in the wrong locations.

All computer systems have three buses: (1) an address bus that provides the memory and I/O with the memory address or the I/O port number, (2) a data bus that transfers data between the microprocessor and the memory and I/O in the system, and (3) a control bus that provides control signals to the memory and I/O. These buses must be present in order to interface to memory and I/O.

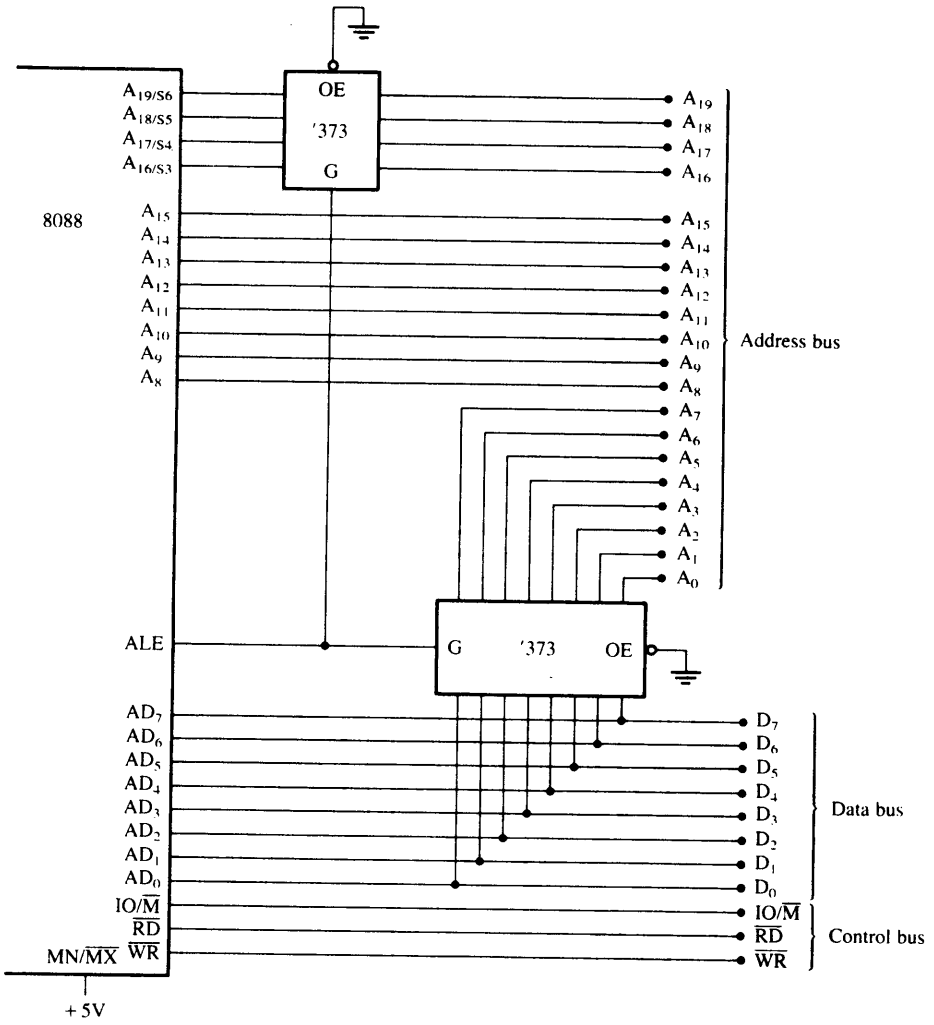

FIGURE 8–5 The 8088 microprocessor shown with a demultiplexed address bus. This is the model used to build many 8088-based systems.

Demultiplexing the 8088. Figure 8–5 illustrates the 8088 microprocessor and the components required to demultiplex its buses. In this case, two 74LS373 transparent latches are used to demultiplex the address/data bus connections AD7–AD0 and the multiplexed address/status connections A19/S6–A16/S3.

These transparent latches, which are like wires whenever the address latch enable pin (ALE) becomes a logic 1, pass the inputs to the outputs. After a short time, ALE returns to its logic 0 condition, which causes the latches to remember the inputs at the time of the change to a logic 0. In this case, A7–A0 are stored in the bottom latch and A19–A16 are stored in the top latch. This yields a separate address bus with connections A19–A0. These address connections allow the 8088 to address 1M bytes of memory space. The fact that the data bus is separate allows it to be connected to any eight-bit peripheral device or memory component.

Demultiplexing the 8086. Like the 8088, the 8086 system requires separate address, data, and control buses. It differs primarily in the number of multiplexed pins. In the 8088, only AD7–AD0 and A19/S6–A16/S3 are

multiplexed. In the 8086, the multiplexed pins include AD15–AD0, A19/S6–A16/S3, and $\overline{BHE}$/S7. All of these signals must be demultiplexed.

Figure 8–6 illustrates a demultiplexed 8086 with all three buses: address (A19–A0 and $\overline{BHE}$), data (D15–D0), and control (M/$\overline{IO}$, $\overline{RD}$, and $\overline{WR}$).

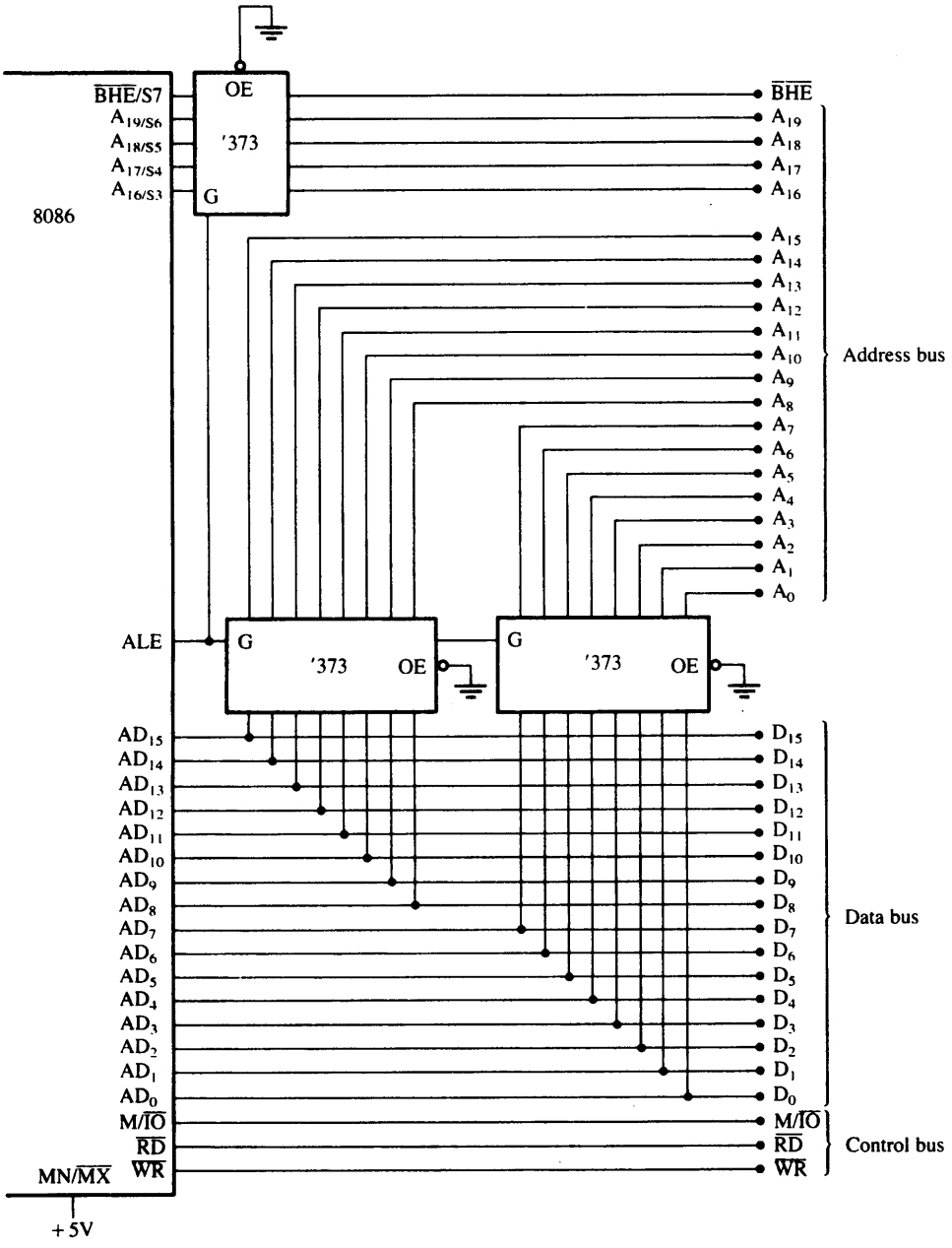

FIGURE 8–6 The 8086 microprocessor shown with a demultiplexed address bus. This is the model used to build many 8086-based systems.

This circuit shown in Figure 8–6 is almost identical to the one pictured in Figure 8–5, except that an additional 74LS373 latch has been added to demultiplex the address/data bus pins AD15–AD8 and a BHE/S7 input has been added to the top 74LS373 to select the high-order memory bank in the 16-bit memory system of the 8086. Here, the memory and I/O system see the 8086 as a device with a 20-bit address bus (A19–A0), a 16-bit data bus (D15–D0), and a three- line control bus (M/IO), RD, and WR).

The Buffered System

If more than 10 unit loads are attached to any bus pin, the entire 8086 or 8088 system must be buffered. The demultiplexed pins are already buffered by the 74LS373 latches, which have been designed to drive the high-capacitance buses encountered in microcomputer systems. The buffer's output currents have been increased so that more TTL unit loads may be driven: a logic 0 output provides up to 32 mA of sink current, and a logic 1 output provides up to 5.2 mA of source current.

A fully buffered signal will introduce a timing delay to the system. This causes no difficulty unless memory or I/O devices are used, which function at near the maximum speed of the bus. Section 8–4 discusses this problem and the time delays involved in more detail.

The Fully Buffered 8088. Figure 8–7 depicts a fully buffered 8088 microprocessor. Notice that the remaining eight address pins, A15–A8, use a 74LS244 octal buffer; the eight data bus pins, D7–D0, use a 74LS245 octal bi-directional bus buffer; and the control bus signals, IO/M, RD, and WR, use a 74LS244 buffer. A fully-buffered 8088 system requires two 74LS244s, one 74LS245, and two 74LS373s. The direction of the 74LS245 is controlled by the DT/R signal, and is enabled and disabled by the DEN signal.

The Fully Buffered 8086. Figure 8–8 illustrates a fully buffered 8086 microprocessor. Its address pins are already buffered by the 74LS373 address latches; its data bus employs two 74LS245 octal bi-directional bus buffers; and the control bus signals, M/IO, RD, and WR, use a 74LS244 buffer. A fully buffered 8086 system requires one 74LS244, two 74LS245s, and three 74LS373s. The 8086 requires one more buffer than the 8088 because of the extra eight data bus connections, D15–D8. It also has a BHE signal that is buffered for memory-bank selection.

8–4 BUS TIMING

It is essential to understand system bus timing before choosing a memory or I/O device for interfacing to the 8086 or 8088 microprocessors. This section provides insight into the operation of the bus signals, and the basic read and write timing of the 8086/8088. It is important to note that we discuss only the times that affect memory and I/O interfacing in this section.

Basic Bus Operation

The three buses of the 8086 and 8088—address, data, and control—function exactly the same way as those of any other microprocessor. If data are written to the memory (see the simplified timing for write in Figure 8–9), the microprocessor outputs the memory address on the address bus, outputs the data to be written into memory on the data bus, and issues a write (WR) to memory and IO/M = 0 for the 8088 and M/IO = 1 for the 8086. If data are read from the memory (see the simplified timing for read in Figure 8–10), the microprocessor outputs the memory address on the address bus, issues a read (RD) memory signal, and accepts the data via the data bus.

Timing in General

The 8086/8088 microprocessors use the memory and I/O in periods called **bus cycles.** Each bus cycle equals four system-clocking periods (T states). Some new microprocessors divide the bus cycle into as few as two clocking

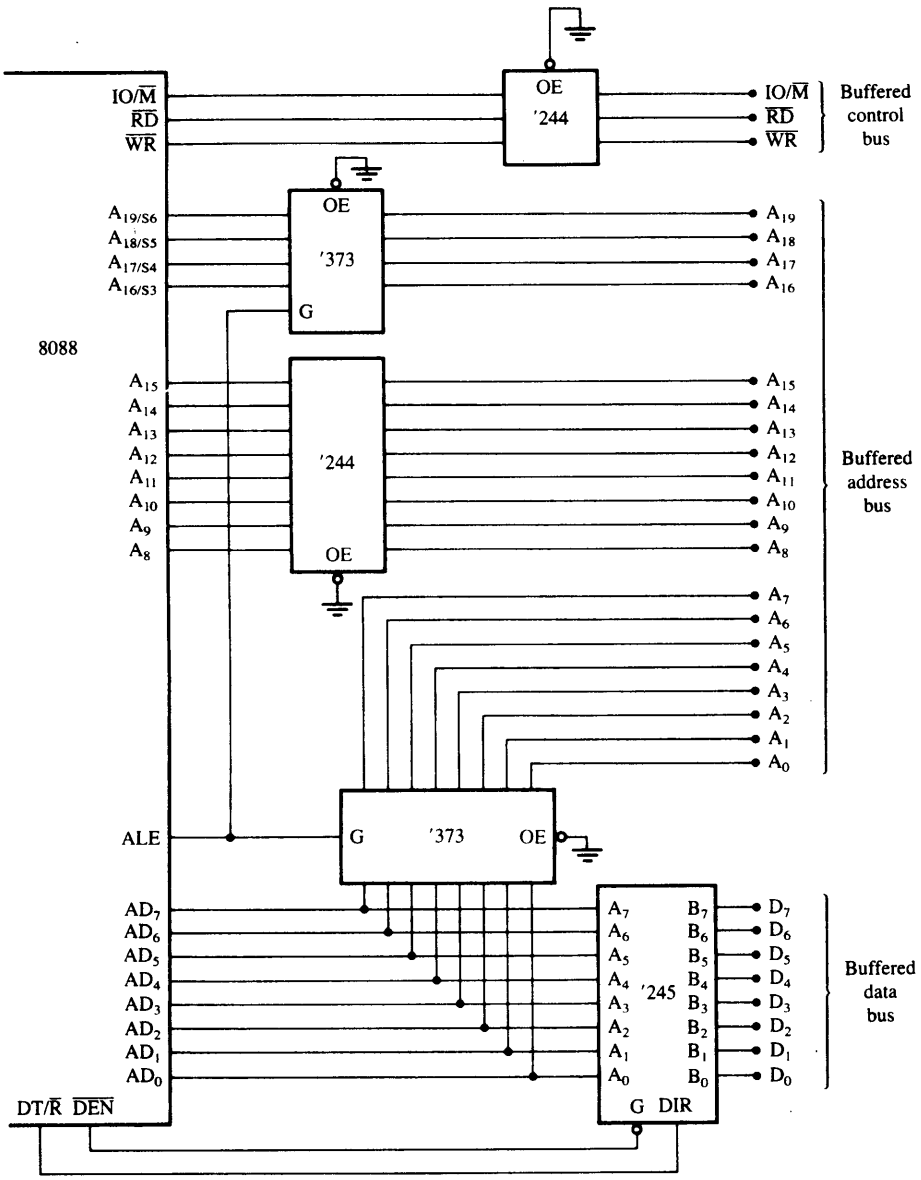

FIGURE 8–7 A fully buffered 8088 microprocessor.

periods. If the clock is operated at 5 MHz (the basic operating frequency for these two microprocessors), one 8086/8088 bus cycle is complete in 800 ns. This means that the microprocessor reads or writes data between itself and memory or I/O at a maximum rate of 1.25 million times a second. (Because of the internal queue, the 8086/8088 can execute 2.5 million instructions per second [MIPS] in bursts.) Other available versions of these microprocessors operate at much higher transfer rates due to higher clock frequencies.

During the first clocking period in a bus cycle, which is called T1, many things happen. The address of the memory or I/O location is sent out via the address bus and the address/data bus connections. (The address/data bus is multiplexed and sometimes contains memory-addressing information, sometimes data.) During T1, control