

FIGURE 10-16 An 8-digit LED display interfaced to the 8088 microprocessor through an 82C55 PIA

PAL16L8 is listed in Example 10–7. The PAL decodes the I/O address and also develops a lower write strobe for the $\overline{\text{WR}}$ pin of the 82C55.

EXAMPLE 10–7

AUTHOR Barry B. Brey
 COMPANY BreyCo
 DATE 7/6/96
 CHIP DECODERD PAL16L8

```
;pins 1 2 3 4 5 6 7 8 9 10
      A2 A3 A4 A5 A6 A7 A8 A9 A10 GND
```

```
;pins 11 12 13 14 15 16 17 18 19 20
      A11 CS IOM A12 A13 A14 A15 NC NC VCC
```

EQUATIONS

```
/CS = /A15 * /A14 * /A13 * /A12 * /A11 * A10 * A9 * A8 * /A6 * /A5 * /A4 * /A2 * /IOM
```

The resistor values are chosen in Figure 10–16 so that the segment current is 80 mA. This current is required to produce an average current of 10 mA per segment as the displays are multiplexed. A 6-digit display would use a segment current of 60 mA for an average of 10 mA per segment. In this type of display system, only one of the eight display positions is on at any given instant. The peak anode current in an 8-digit display is 560 mA (7 segments \times 80 mA), but the average anode current is 80 mA. In a 6-digit display, the peak current would be 420 mA (7 segments \times 60 mA). Whenever displays are multiplexed, we increase the segment current from 10 mA (for a display that uses 10 mA per segment as the nominal current) to a value equal to the number of display positions times 10 mA. This means that a 4-digit display uses 40 mA per segment, a 5-digit display uses 50 mA, and so on.

In this display, the segment load resistor passes 80 mA of current and has a voltage of approximately 3.0V across it. The LED is 1.65V nominally and a few tenths are dropped across the anode switch and the segment switch, hence a voltage of 3.0V across the segment load resistor. The value of the resistor is $3.0V/80mA = 37.5 \Omega$. The closest standard resistor value of 37 Ω is shown in Figure 10–16 for the segment load.

The resistor in series with the base of the segment switch assumes that the minimum gain of the transistor is 100. The base current is therefore $80mA/100 = 0.8 \text{ mA}$. The voltage across the base resistor is approximately 3.0V (the minimum logic 1 voltage level of the 82C55) minus the drop across the emitter-base junction (0.7V), or 2.3V. The value of the base resistor is $2.3V/0.8mA = 2.875 \text{ K}\Omega$. The closest standard resistor value is 2.7 $\text{K}\Omega$, but a 2.2 $\text{K}\Omega$ is chosen for this circuit.

The anode switch has a single resistor on its base. The current through the resistor is $560mA/100 = 5.6 \text{ mA}$ because the minimum gain of the transistor is 100. This exceeds the maximum current of 4.0 mA from the 82C55, but this is small enough that it will work without a problem. The maximum current assumes that you are using the port pin as a TTL input to another circuit. If the amount of current was over 8.0–10.0 mA, then appropriate circuitry in the form of either a Darlington pair or another transistor switch would be required. Here the voltage across the base resistor is 5.0V minus the drop across the emitter-base junction (0.7V) minus the voltage at the port pin (0.4V) for a logic 0 level. The value of the resistor is $3.9V/5.6mA = 696 \Omega$. The closest standard resistor value is 690 Ω , but a 1 $\text{K}\Omega$ is chosen in the example.

Before software to operate the display is examined, we must first program the 82C55. This is accomplished with the short sequence of instructions listed in Example 10–8. Here port A and port B are both programmed as outputs.

EXAMPLE 10-8

```

;programming the 82C55 PIA
;
0000 B0 80          MOV    AL,10000000B
0002 BA 0703       MOV    DX,703H          ;address command
0005 EE           OUT    DX,AL          ;program 82C55

```

The procedure to drive these displays is listed in Example 10-9. For this display system to function correctly, we must call this procedure often. Notice that the procedure calls another procedure (DELAY) that causes a 1 ms time delay. This time delay is not illustrated in this example, but is used to allow time for each display position to turn on. It is recommended by the manufacturers of LED displays that the display flash be between 100 Hz and 1,500 Hz. Using a 1 ms time delay, we light each digit for 1 ms for a total display flash rate of 1000 Hz / 8 display, or a flash rate of 125.

EXAMPLE 10-9

```

;A procedure that scans the 8-digit LED display.
;This procedure must be called from a program
;whenever possible to display 7-segment
;coded data from memory.
;
0006          DISP  PROC  NEAR
;
0006 9C          PUSHF          ;save registers
0007 50          PUSH  AX
0008 53          PUSH  BX
0009 52          PUSH  DX
000A 56          PUSH  SI
;
;setup registers for display
000B BB 0008     MOV    BX,8          ;load count
000E B4 7F      MOV    AH,7FH        ;load selection pattern
0010 BE 00FF R  MOV    SI,OFFSET MEM-1 ;address data
0013 BA 0701     MOV    DX,701H        ;address Port B
;
;display 8 digits
0016          DISP1:
0016 8A C4      MOV    AL,AH          ;select a digit
0018 EE        OUT    DX,AL
0019 4A        DEC    DX          ;address Port A
001A 8A 00     MOV    AL,[BX+SI]      ;get 7-segment data
001C EE        OUT    DX,AL
001D E8 029A R CALL  DELAY          ;wait one millisecond
0020 D0 CC     ROR    AH,1          ;address next digit
0022 42        INC    DX          ;address Port B
0023 4B        DEC    BX          ;adjust count
0024 75 F0     JNZ   DISP1        ;repeat 8 times
;
0026 5E        POP    SI          ;restore registers
0027 5A        POP    DX
0028 5B        POP    BX
0029 58        POP    AX
002A 9D        POPF
002B C3        RET
;
002C          DISP  ENDP

```

The display procedure (DISP) addresses an area of memory where the data, in 7-segment code, is stored for the eight display digits. The AH register is loaded with a code (7FH) that initially addresses the most-significant display position. Once this position is selected, the

contents of memory location MEM +7 is addressed and sent to the most-significant digit, selection code is then adjusted to select the next display digit, as is the address. This repeats eight times to display the contents of location MEM through MEM +7 on the eight display digits.

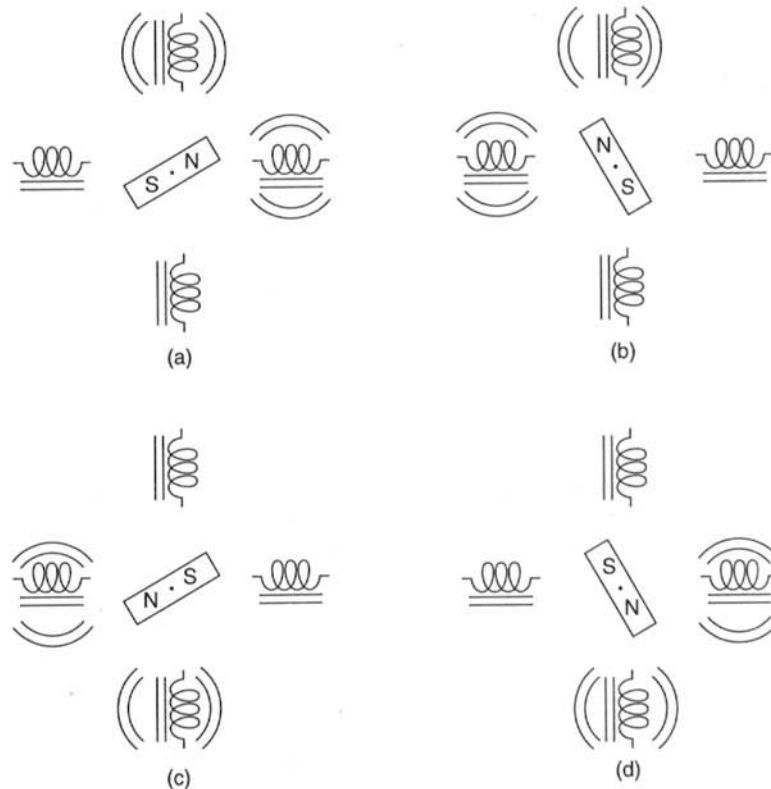
A Stepper Motor Interfaced to the 82C55. Another device often interfaced to a computer is the *stepper motor*. A stepper motor is a digital motor because it is moved in discrete steps as it traverses through 360°. A common stepper motor is geared to move perhaps 15° per step. An inexpensive stepper motor to 1° per step in a more costly high-precision stepper motor. In these cases, these steps are gained through many magnetic poles and/or gearing. Notice that the coils are energized in Figure 10-17. If less power is required, one coil may be energized at a time causing the motor to step at 45°, 135°, 225°, and 315°.

Figure 10-17 shows a four-coil stepper motor that uses an armature with a single pole. The illustration shows the stepper motor four times with the armature (permanent) rotated to four discrete places. This is accomplished by energizing the coils as shown. This is an illustration of full stepping. The stepper motor is driven using NPN Darlington pairs to provide a large current to each coil.

A circuit that can drive this stepper motor is illustrated in Figure 10-18 with the full circuit shown in place. This circuit uses the 82C55 to provide it with the drive signals used to rotate the armature of the motor in either the right-hand or left-hand direction.

A simple procedure that drives the motor (assuming port A is programmed in mode 0 as an output device) is listed in Example 10-10. This subroutine is called with CX holding the number of steps and direction of the rotation. If CX > 8000H, the motor spins in the right-hand direction; if CX < 8000H, it spins in the left-hand direction. The leftmost bit of CX is removed, and the remaining 15-bits contain the number of steps. Notice that the procedure uses a time delay between steps.

FIGURE 10-17 The stepper motor showing full-step operation. (a) 45° (b) 135° (c) 225° (d) 315°



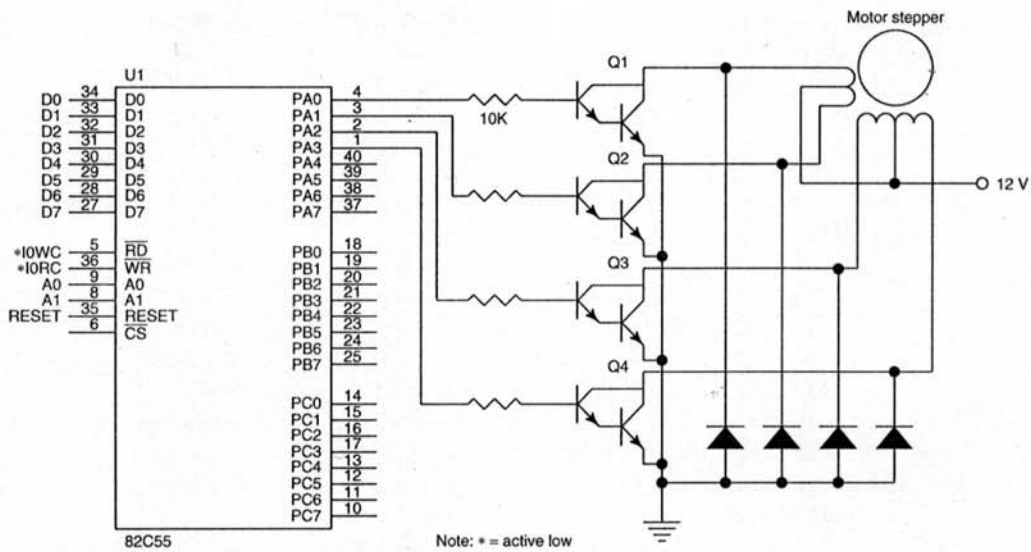


FIGURE 10-18 A stepper motor interfaced to the 82C55. This illustration does not show the decoder.

illustrated) that causes a 1 ms time delay. This time delay is required to allow the stepper motor armature time to move to its next position.

EXAMPLE 10-10

```

= 0040          PORT EQU 40H          ;assign Port A
                ;
                ;A procedure to control stepper motor.
                ;
0000          STEP PROC NEAR
0000          A0 0000 R              MOV AL, POS          ;get position
0003          81 F9 8000             CMP CX, 8000H
0007          77 10                  JA RH              ;if right-hand direction
0009          83 F9 00               CMP CX, 0
000C          74 14                  JE STEP_OUT       ;if no steps
000E
000E          D0 C0                  STEP1: ROL AL, 1      ;step left
0010          E6 40                  OUT PORT, AL
0012          E8 0011                CALL DELAY        ;wait one millisecond
0015          E2 F7                  LOOP STEP1       ;repeat until CX = 0
0017          EB 09                  JMP STEP_OUT
0019
0019          81 E1 7FFF             RH: AND CX, 7FFFH   ;clear bit 15
001D
001D          D0 C8                  RH1: ROR AL, 1    ;step right
001F          E6 40                  OUT PORT, AL
0021          E8 0006                CALL DELAY        ;wait one millisecond
0024          E2 F7                  LOOP RH1         ;repeat until CX = 0
0026
0026          A2 0000               STEP_OUT: MOV POS, AL ;save position
0029          C3                     RET
0029          STEP ENDP

```

The current position is stored in memory location POS, which must be initialized with 33H, 66H, 0CCH, or 99H. This allows a simple ROR (step right) or ROL (step left) instruction to rotate the binary bit pattern for the next step.

Stepper motors can also be operated in the half-step mode, which allows eight steps per sequence. This is accomplished by using the full-step sequence described with a half step obtained by energizing one coil interspersed between the full steps. Half stepping allows the armature to be positioned at 0° , 90° , 180° , and 270° . The half-step position codes are 11H, 22H, 44H, and 88H. A complete sequence of eight steps would follow as: 11H, 33H, 22H, 66H, 44H, 0CCH, 88H, and 99H. This sequence could either be output from a lookup table or generated with software.

Key Matrix Interface. Keyboards come in a vast variety of sizes, from the standard 101-key QWERTY keyboards interfaced to the microprocessor to small, specialized keyboards that may contain only 4 to 16 keys. This section of the text concentrates on the smaller keyboards that may be purchased, pre-assembled, or constructed out of individual key switches.

Figure 10-19 illustrates a small-key matrix that contains 16 switches interfaced to ports A and B of an 82C55. In this example, the switches are formed into a 4×4 matrix, but any matrix could be used, such as a 2×8 . Notice how the keys are organized into four rows (ROW0-ROW3) and four columns (COL0-COL3). Also notice that each row is connected to 5.0V through a 10 K Ω pull-up resistor to ensure that the row is pulled high when no push-button switch is closed.

The 82C55 is decoded (PAL program not shown) at I/O ports 50H-53H for an 8088 microprocessor. Port A is programmed as an input port to read the rows, and port B is programmed as an output port to select a column. For example, if 1110 is output to port B pins PB3-PB0, column zero has a logic 1, so the four keys in column zero are selected. Notice that with a logic 0 on PB0, the only switches that can place a logic 0 onto port A are switches 0-3. If switches 4-F are closed, the corresponding port A pins remain a logic 1. Likewise, if a 1101 is output to port B, switches 4-7 are selected and so forth.

A flowchart of the software required to read a key from the keyboard matrix and de-bounce the key is illustrated in Figure 10-20. De-bouncing is normally accomplished with a short time delay of from 10-20 ms. The flowchart contains three main sections. The first waits for the release of a key. This seems awkward, but software executes very quickly in a microprocessor and there is a possibility that the program will return to the top of this program before the key is released, so we must wait for a release first. Next the flowchart shows that we wait for a keystroke. Once the keystroke is detected, the position of the key is calculated in the final part of the flowchart.

The software uses a procedure called SCAN to scan the keys, and another called DELAY to waste 10 ms of time for de-bouncing. The main keyboard procedure is called KEY, and it appears with the others in Example 10-11. Note that the SCAN procedure is generic, so it can handle any configuration of keyboard from a 2×2 matrix to an 8×8 . Changing the two equates at the start of the program (ROW and COL) will change the configuration of the software for any size keyboard. Also note that the example does not show the steps required to initialize the 82C55 so that port A is an input port and port B is an output port.

EXAMPLE 10-11

```

;A keyboard procedure that scans the keyboard and
;returns with the numeric code of the key in AL.
;
= 0004      ROWS EQU 4           ;number of rows
= 0004      COLS EQU 4          ;number of columns
= 0050      PORTA EQU 50H       ;port A address
= 0051      PORTB EQU 51H      ;port B address

0000      KEY PROC NEAR USES CX

0001      E8 002F              CALL SCAN           ;test all keys
0004      75 FA                JNZ KEY              ;if key closed
0006      E8 0048              CALL DELAY          ;wait for about 10 ms
0009      E8 0027              CALL SCAN           ;test all keys
000C      75 F2                JNZ KEY              ;if key closed

```

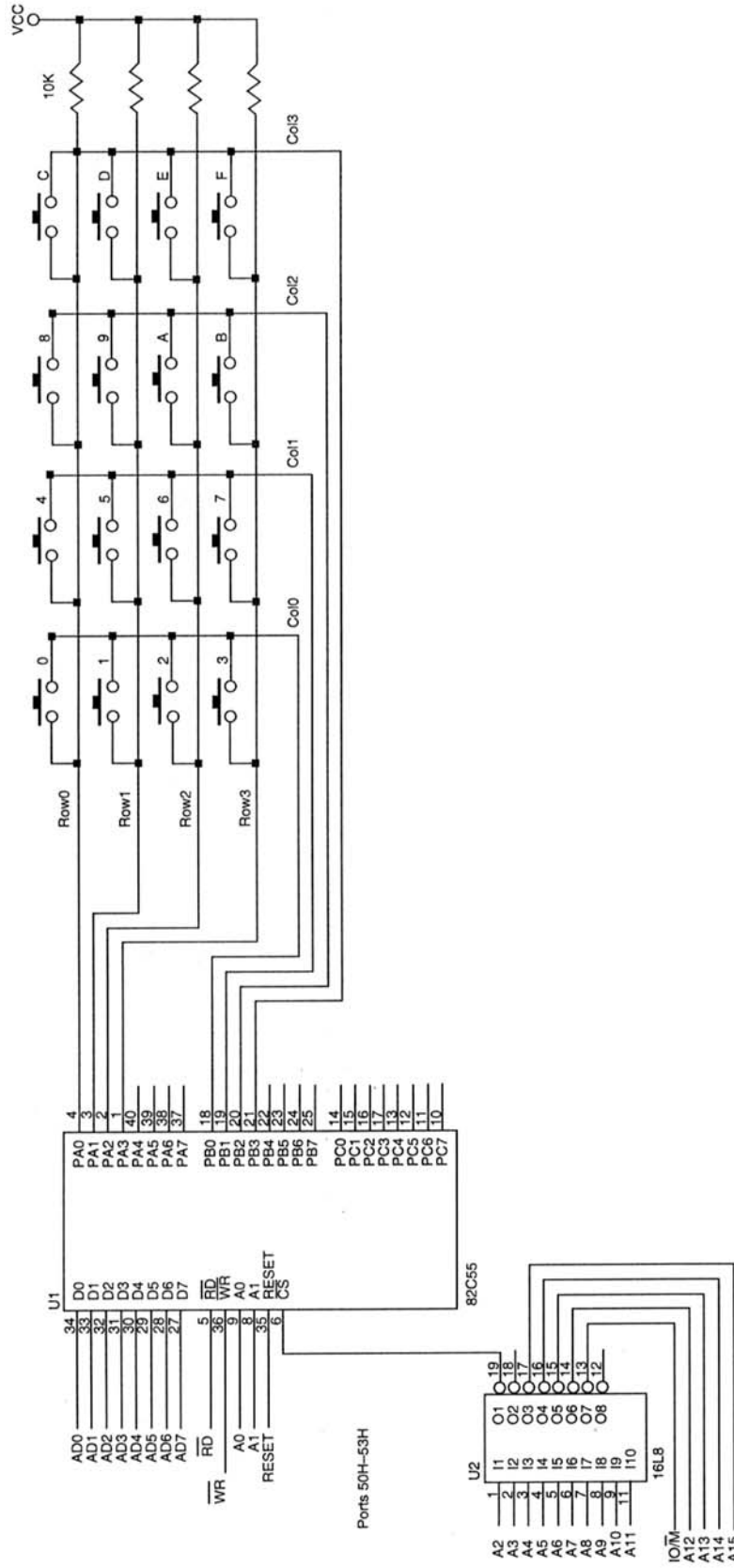
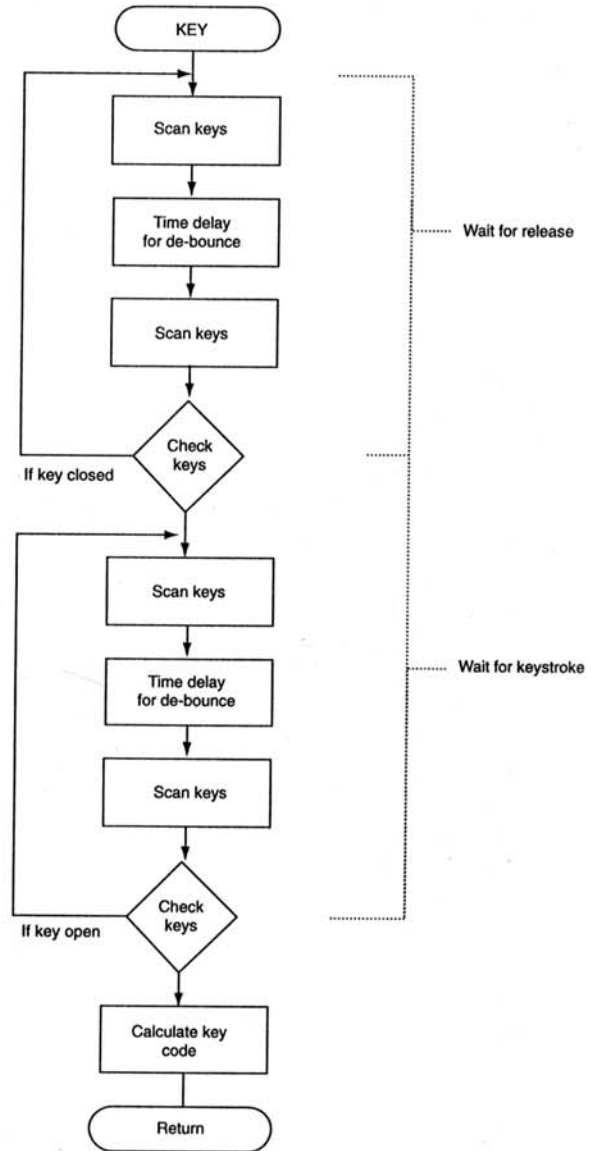


FIGURE 10-19 A 4 x 4 keyboard matrix connected to an 8088 microprocessor through the 82C55 PIA

FIGURE 10-20 The flow-chart of a keyboard-scanning procedure



```

000E      000E      KEY1:
000E      E8 0022      CALL  SCAN          ;test all keys
0011      74 FB        JZ    KEY1          ;if no key closed
0013      E8 003B      CALL  DELAY         ;wait for about 10 ms
0016      E8 001A      CALL  SCAN          ;test all keys
0019      74 F3        JZ    KEY1          ;if no key closed
001B      50          PUSH  AX              ;save row codes
001C      B0 04        MOV  AL, COLS       ;calculate starting row key
001E      2A C1        SUB  AL, CL
0020      B5 04        MOV  CH, ROWS
0022      F6 E5        MUL  CH
0024      8A C8        MOV  CL, AL
0026      FE C9        DEC  CL
0028      58          POP  AX
0029
0029      0029      KEY2:
0029      D0 C8        ROR  AL, 1          ;find row position
002B      FE C1        INC  CL
002D      72 FA        JC   KEY2
002F      8A C1        MOV  AL, CL          ;mode code to AL
    
```



```

                                RET
0033      KEY      ENDP

0033      SCAN    PROC    NEAR USES BX
0034  B1 04      MOV    CL,ROWS      ;form row mask
0036  B7 FF      MOV    BH,0FFH
0038  D2 E7      SHL    BH,CL
003A  B9 0004    MOV    CX,COLS      ;load column count
003D  B3 FE      MOV    BL,0FEH      ;get selection code
003F      SCAN1:
003F  8A C3      MOV    AL,BL      ;select column
0041  E6 51      OUT    PORTB,AL
0043  D0 C3      ROL    BL,1
0045  E4 50      IN     AL,PORTA      ;read rows
0047  0A C7      OR     AL,BH
0049  3C FF      CMP    AL,0FFH      ;test for a key
004B  75 02      JNZ   SCAN2
004D  E2 F0      LOOP  SCAN1
004F      SCAN2:
                                RET

0051      SCAN    ENDP

0051      DELAY   PROC    NEAR USES CX
0052  B9 1388    MOV    CX,5000      ;10ms (8MHz clock)
0055      DELAY1:
0055  E2 FE      LOOP  DELAY1
                                RET

0059      DELAY   ENDP

```

A note about the SCAN procedure. The time between where the keyboard column is selected and where the rows are read is very short. In a very high-speed system, a small time delay must be placed between these two points for the data at port A to settle to its final state. In most cases, this is not needed.