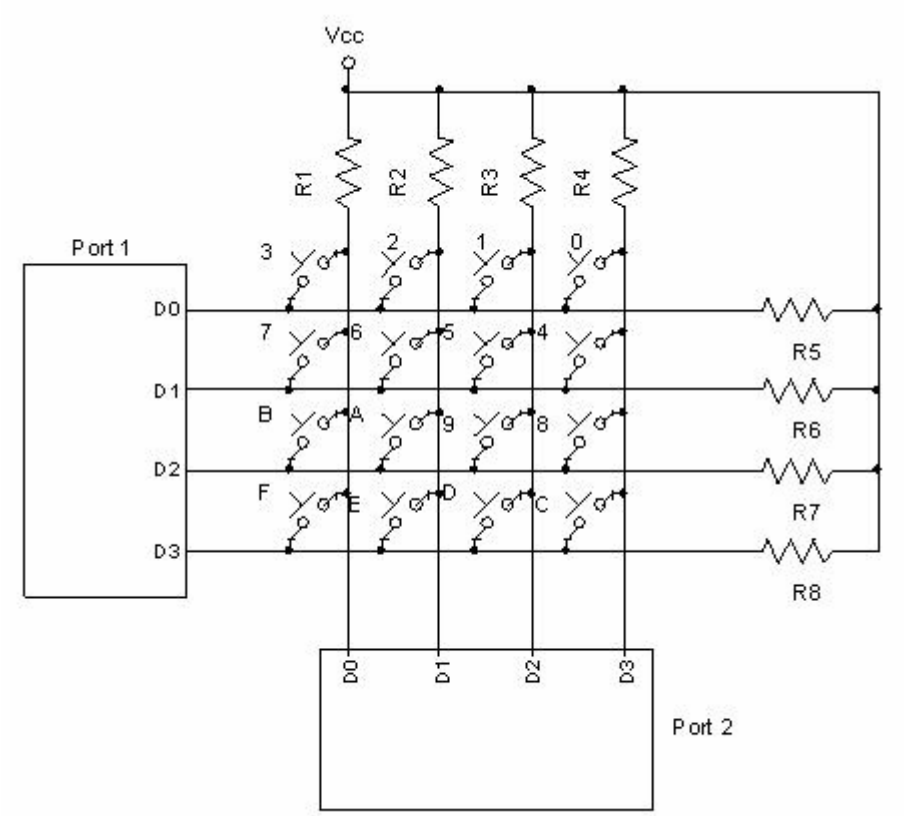
Interfacing the Keyboard to the 8051

At the lowest level, keyboards are organized in a matrix of rows and columns. The CPU accesses both rows and column through ports; therefore, with two 8-bit ports, an 8*8 matrix of keys can be connected to a microprocessor. When a key pressed, a row and column make a connect; otherwise, there is no connection between row and column. In IBM PC keyboards, a single microcontroller (consisting of microprocessor, RAM and EPROM, and several ports all on a single chip) takes care of software and hardware interfacing of keyboard. In such systems it is the function of programs stored in the EPROM of microcontroller to scan the keys continuously, identify which one has been activated, and present it to the motherboard. In this section we look at the mechanism by which the 8051 scans and identifies the key.

Scanning and identifying the key

Figure13.5 shows a 4*4 matrix connected to two ports. The rows are connected to an output port and the columns are connected to an input port. If no key has been pressed, reading the input port will yield 1s for all columns since they are all connected to high (Vcc) If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground. It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed. How it is done is explained next.



Grounding rows and reading columns

To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, and then it reads the columns. If the data read from the columns is D3-D0=1111, no key has been pressed and the process continues until a key press is detected. However, if one of the column bits has a zero, this means that a key press has occurred. For example, if D3-D0=1101, this means that a key in the D1 column has been pressed. After a key press is detected, the microcontroller will go through the process of identifying the key. Starting with the top row, the microcontroller grounds it by providing a low to row D0 only; then it reads the columns. If the data read is all1s, no key in that row is activated and the process is moved to the next row. It grounds the next row, reads the columns, and checks for any zero. This process continues until the row is identified. After identification of the row in which the key has been pressed, the next task is to find out which column the pressed key belongs to. This should be easy since the microcontroller knows at any time which row and column are being accessed.

Assembly language program for detection and identification of key activation is given below. In this program, it is assumed that P1 and P2 are initialized as output and input, respectively. Program13.1 goes through the following four major stages:

1. To make sure that the preceding key has been released, 0s are output to all rows at once, and the columns are read and checked repeatedly until all the columns are high. When all columns are found to be high, the program waits for a short amount of time before it goes to the next stage of waiting for a key to be pressed.

2) To see if any key is pressed, the columns are scanned over and over in an infinite loop until one of them has a 0 on it. Remember that the output latches connected to rows still have their initial zeros (provided in stage 1), making them grounded. After the key press detection, it waits 20ms for the bounce and then scans the columns again. This serves two functions: (a) it ensures that the first key press detection was not an erroneous one due to spike noise, and(b) the 20ms delay prevents the same key press from being interpreted as a multiple key press. If after the 20-ms delay the key is still pressed, it goes to the next stage to detect which row it belongs to; otherwise, it goes back into the loop to detect a real key press

3) To detect which row the key press belongs to, it grounds one row at a time, reading the columns each time. If it finds that all columns are high, this means that the key press cannot belong to that row; therefore, it grounds the next row and continues until it finds the row the key press belongs to. Upon finding the row that the key press belongs to, it sets up the starting address for the look-up table holding the scan codes (or the ASCII value) for that row and goes to the next stage to identify the key.
4) To identify the key press, it rotates the column bits, one bit at a time, into the carry flag and checks to see if it is low. Upon finding the zero, it pulls out the ASCII code for that key from the look-up table; Otherwise, it increments the pointer to point to the next element of the look-up table.

While the key press detection is standard for all keyboards, the process for determining which key is pressed varies. The look-up table method shown in program can be modified to work with any matrix up to 8*8.

Keypad Interfacing

……
…

```
;Keyboard subroutine. This program sends the ASCII code
;for pressed key to P0.1
;P1.0-P1.3 connected to rows P2.0-P2.3 connected to columns

       MOV P2,#0FFH             ;make P2 an input port
K1:    MOV P1,#0                ;ground all rows at once
       MOV A,P2                 ;read all col. (ensure all keys
open)
       ANL A,00001111B          ;masked unused bits
       CJNE A,#00001111B,K1   ;check til all keys released
K2:    ACALL DELAY              ;call 20 msec delay
       MOV A,P2                 ;see if any key is pressed
       ANL A,#00001111B         ;mask unused bits
       CJNE A,#00001111B,OVER ;key pressed, await closure
       SJMP K2                  ;check il key pressed
OVER:  ACALL DELAY              ;wait 20 msec debounce time
       MOV A,P2                 ;check key closure
       ANL A,#00001111B         ;mask unused bits
       CJNE A,#00001111B,OVER1;key pressed, find row
       SJMP K2                  ;if none, keep polling
OVER1: MOV P1,#11111110B        ;ground row 0
       MOV A,P2                 ;read all columns
       ANL A,#00001111B         ;mask unused bits
       CJNE A,#00001111B,ROW_0;key row 0, find the col.
       MOV P1,#11111101B        ;ground row 1
       MOV A,P2                 ;read all columns
       ANL A,#00001111B         ;mask unused bits
       CJNE A,#00001111B,ROW_1;keyrow 1, find the col.
       MOV P1,#11111011B        ;ground row 2
       MOV A,P2                 ;read all columns
       ANL A,#00001111B         ;mask unused bits
       CJNE A,#00001111B,ROW_2;key row 2, find the col.
       MOV P1,#11110111B        ;ground row 3
       MOV A,P2                 ;read all columns
       ANL A,#00001111B         ;mask unused bits
       CJNE A,#00001111B,ROW_3;keyrow 3, find the col.
       LJMP K2                  ;if none, false input, repeat
ROW_0: MOV DPTR,#KCODE0         ;set DPTR=start of row 0
       SJMP FIND                ;find col. key belongs to
ROW_1: MOV DPTR,#KCODE1         ;set DPTR=start of row 1
       SJMP FIND                ;find col. key belongs to
ROW_2: MOV DPTR,#KCODE2         ;set DPTR=start of row 2
```

```
       SJMP FIND              ;find col. key belongs to
ROW_3: MOV DPTR,#KCODE3       ;set DPTR=start of row 3
FIND:  RRC A                  ;see if any CY bit low
       JNC MATCH              ;if zero, get the ASCII code
       INC DPTR               ;point to next col. address
       SJMP FIND              ;keep searching
MATCH: CLR A                  ;set A=0 (match is found)
       MOVC A,@A+DPTR         ;get ASCII code from table
       MOV  P0,A              ;display pressed key
       LJMP  K1

;ASCII LOOK-UP TABLE FOR EACH ROW

      ORG     300H
KCODE0: DB     '0','1','2','3'          ;ROW 0
KCODE1: DB     '4','5','6','7'          ;ROW 1
KCODE2: DB     '8','9','A','B'          ;ROW 2
KCODE3: DB     'C','D','E','F'          ;ROW 3

      END
```