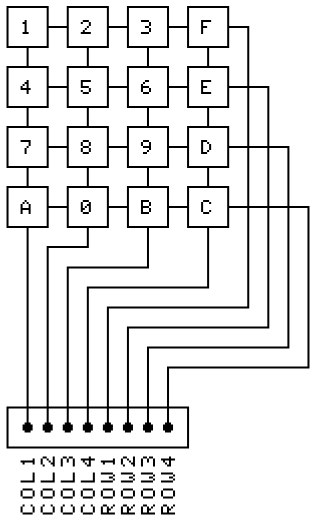
PIC Tutorial Nine - HEX Keypad

For these tutorials you require the Main Board, Keypad Board, LCD Board, IR Board and RS232 Board. Download zipped tutorial files.

These tutorials demonstrate how to read a HEX keypad, these are a standard device with 16 keys connected in a 4x4 matrix, giving the characters 0-9 and A-F. You can also use a 4x3 keypad, which gives the numbers 0-9, * and #.



This is how the HEX keypad is connected, each square with a number or letter in it is a push to make switch, which connects the horizontal wires (rows) with the vertical wires (columns). So if you press button 'A' it will connect COL1 with ROW4, or pressing button '6' will connect COL3 with ROW2. For a numeric (4x3) keypad COL4 will be missing, and 'A' and 'B' replaced with '*' and '#' but is otherwise the same. The sample programs use a lookup table for the keys, this would need to be changed to insert the correct values for the non-numeric characters.

As the switches are all interconnected, we need a way to differentiate between the different ones - the four resistors on the interface board pull lines COL1 to COL4 high, these four lines are the ones which are read in the program. So in the absence of any switch been pressed these lines will all read high. The four ROW connections are connected to output pins, and if these are set high the switches will effectively do nothing - connecting a high level to a high level, results in a high level.

In order to detect a switch we need to take the ROW lines low, so if we take all the ROW lines low - what happens?. Assuming we press button 1, this joins COL1 with ROW1, as ROW1 is now at a low level, this will pull COL1 down resulting in a low reading on COL1. Unfortunately if we press button 4, this joins COL1 with ROW2, as ROW2 is at a low level this also results in a low reading at COL1. This would only give us four possible choices, where each four buttons in a COL do exactly the same (e.g. 1, 4, 7, and A are the same).

The way round this is to only switch one ROW at a time low, so assuming we set ROW1 low we can then read just the top row of buttons, button 1 will take COL1 low, button2 will take COL2 low, and the same for buttons '3' and 'F' in COL3 and COL4. The twelve lower buttons won't have any effect as their respective ROW lines are still high. So to read the other buttons we need to take their respective ROW lines low, taking ROW2 low will allow us to read the second row of buttons (4, 5, 6, and E), again as the other three ROW lines are now high the other 12 buttons have no effect. We can then repeat this for the last two ROW's using ROW3 and ROW4, so we read four buttons at a time, taking a total of four readings to read the entire keypad - this is a common technique for reading keyboards, and is called 'Keyboard Scanning'.

One obvious problem is what happens if you press more than one key at a time?, there are a number of ways to deal with this, one way would be to check for multiple key presses and ignore them, a simpler way (and that used in the examples) is to accept the first key you find which is pressed. You will find that various commercial products deal with this situation in similar ways, some reject multiple key presses, and some just accept the first one.

As with previous tutorials, the idea is to give a tested working routine which can be used elsewhere, the subroutine to be called is Chk_Keys, the first thing this does is check to see if any of the 12 keys are pressed (by making all the ROW lines low) and waiting until no keys are pressed - this avoids problems with key presses repeating. Once no keys are pressed it jumps to the routine Keys which repeatedly scans the keyboard until a key is pressed, a call to one of my standard delay routines (call Delay20) provides a suitable de-bouncing delay. Once a key has been pressed the result is returned in the variable 'key', this is then logically ANDed with 0x0F to make absolutely sure it's between 0 and 15. Next this value is passed to a look-up table which translates the key to it's required value (in this case the ASCII value of the labels on the keys). Finally the ASCII value is stored back in the 'key' variable (just in case you might need it storing) and the routine returns with the ASCII value in the W register.

```
;Keypad subroutine

Chk_Keys        movlw   0x00           ;wait until no key pressed
                movwf   KEY_PORT       ;set all output pins low
                movf    KEY_PORT,   W
                andlw   0x0F           ;mask off high byte
                sublw   0x0F
                btfsc   STATUS, Z              ;test if any key pressed
                goto    Keys           ;if none, read keys
                call    Delay20
                goto    Chk_Keys       ;else try again

Keys            call    Scan_Keys
                movlw   0x10           ;check for no key pressed
                subwf   key, w
                btfss   STATUS, Z
                goto    Key_Found
                call    Delay20
                goto    Keys
Key_Found       movf    key, w
                andlw   0x0f
                call    Key_Table      ;lookup key in table
                movwf   key            ;save back in key
                return                 ;key pressed now in W

Scan_Keys       clrf    key
                movlw   0xF0           ;set all output lines high
                movwf   KEY_PORT
                movlw   0x04
                movwf   rows           ;set number of rows
                bcf     STATUS, C              ;put a 0 into carry
Scan            rrf     KEY_PORT, f
                bsf     STATUS, C              ;follow the zero with ones
;comment out next two lines for 4x3 numeric keypad.
                btfss   KEY_PORT, Col4
                goto    Press
                incf    key, f
                btfss   KEY_PORT, Col3
                goto    Press
                incf    key, f
                btfss   KEY_PORT, Col2
                goto    Press
                incf    key, f
                btfss   KEY_PORT, Col1
                goto    Press
                incf    key, f
                decfsz  rows, f
                goto    Scan
Press           return
```

For the full code, with the equates, variable declarations and look-up table, consult the code for the examples below.

Tutorial 9.1

Tutorial 9.1 simply reads the keyboard and displays the value of the key pressed on the LCD module, it shows this as both HEX and ASCII, so if you press key '0' it will display '30 0'.

Tutorial 9.2

This tutorial implements a simple code lock, you enter a 4 digit code and it responds on the LCD with either 'Correct Code' or 'Wrong Code', it uses all sixteen available I/O lines on a 16F628 which leaves no spare I/O line for opening an electrical lock. However, using a 4x3 keypad would free one I/O line, or a 16F876 could be used giving plenty of free I/O. I've set the code length at 4 digits, it could very easily be altered from this. The secret code is currently stored within the program source code, as the 16F628 and 16F876 have internal EEPROM the code could be stored there and be changeable from the keypad.

Tutorial 9.3

Tutorial 9.3 reads the keypad and sends the ASCII code via RS232 at 9600 baud, you can use HyperTerminal on a PC to display the data. Basically this is the same as 9.1 but uses a serial link to a PC instead of the LCD screen.

Tutorial 9.4

A common use for keyboard scanning applications, this example reads the keypad and transmits IR remote control signals to control a Sony TV, this works in a very similar way to the circuit used in your TV remote control. For this application I've modified the keypad routines slightly - as we want the keys to repeat I've removed the first part which waits for a key to be released, and as we don't need ASCII values back from the key presses, I've removed the look-up table which gave us the ASCII values. I've replaced that with a jump table, this jumps to one of 16 different small routines which send the correct SIRC code for each button pressed - it's a lot neater than checking which button was pressed. The controls I've chosen to include are, 0-9, Prog Up, Prog Down, Vol Up, Vol Down, On Off, and Mute. If you want to change which button does what, you can either alter the equates for the button values, or change the order of the Goto's in the jump table, notice that these are currently in a peculiar order, this is to map the numeric buttons correctly.