Problem Description and theory behind:

Matrix keypads are very useful when designing certain systems which needs user input. These keypads are constructed by arranging push button switches in rows and columns as shown in Fig.1. Scanning keypad to detect pressed keys involves several steps and there are several methods to achieve this. Below we will be issuing a quick description of those functionalities.

Button

The button is one of the simplest input elements. It consists of two contacts which are connected if the button is pressed. So if one of the contacts is connected to for example GND and the other is connected to the microcontroller input and to VCC through a pull-up resistor (either internally by the microcontroller, or with an external resistor), then the controller will read HIGH as long as the button is not pressed, and will read LOW while the button is pressed.

A somewhat annoying property of switches and buttons is their bouncing. This is due to the mechanical contacts, which do not get a proper contact right away, causing the signal to change between LOW and HIGH for several times until settling at the appropriate voltage level. The bouncing effects can last for several milliseconds and are disastrous for interrupt-driven applications. In consequence, mechanical buttons and switches must be debounced. This can either be done in hardware, e.g. with a capacitor, for very short bouncing durations and/or slow clock frequencies perhaps by the controller with noise cancellation, or in software with the help of a timeout. As a rule of thumb, a timeout of ***5 ms*** generally suffices.

Matrix Keypad

The matrix keypad consists of several buttons which are arranged in a matrix array, see Figure below. Such an arrangement saves connections, so instead of n2 pins for n2 buttons, only 2n pins are required.

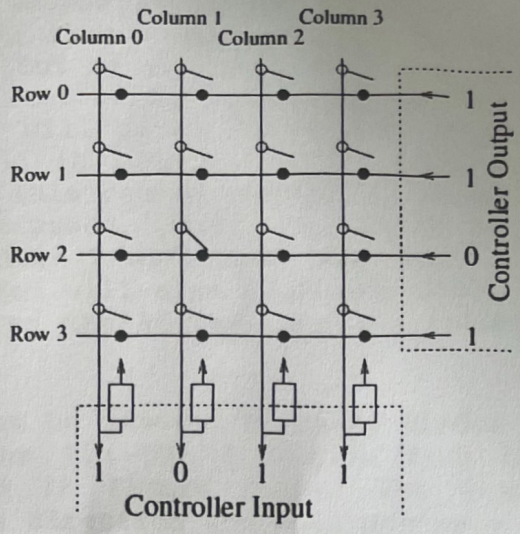

Fig.1: Operating principle of a matrix keypad for 4 × 4 keys.

As Figure above shows, all buttons in the same row are connected by a common row wire at one contact, and all buttons in the same column are connected by a common column wire at the other contact. If the column lines are connected to pull-up resistors so that the column lines are HIGH by default, and if one row line is set to LOW, then a pressed button in this row will cause the corresponding column line to go LOW. Thus, the keypad is read by alternately setting each of the row lines to LOW and reading the state of the column lines. The button in <row, col> = <i, j> is pressed → col j reads LOW when row i is set to LOW and all other rows are HIGH. The period of changing the row should be in the ms range.

Although the basic principle is very simple, there are some things to bear in mind when multiplexing a keypad in this manner. First, you must leave sometime between setting the row and reading the column. The line needs some time to attain the LOW voltage level, so if you read too fast after setting the row, you will still read HIGH even though a button in the row has been pressed. Depending on the hardware, on your algorithm for reading the keypad, and on the speed of your controller, the effect of reading too fast can be that you do not recognize any pressed buttons, or that you recognize the pressed button belatedly, that is, when you are already in the next row, and will attribute it to the wrong row.

The time required between setting the row and reading the columns depends on the characteristics of the connections and on the controller's input delay. To make the duration between setting the row and reading the columns as large as possible, it is generally a good idea to initially select the first row and then to read the columns first and select the next row afterwards in the subsequent periodically executed code.

Another issue to bear in mind when using a multiplexed keypad is the following: Assume that you press the buttons <0, 1>, <0, 2>

and <1, 1> on the keypad. Then you select row 1 by setting it to LOW. By rights, you should now read LOW on column 1 and HIGH on all other columns. But if you look at Figure 5.2 (a) and consider the voltage levels caused by the three pressed buttons, you will find that column 2 will be pulled to LOW as well. The problem is that since point (1) is LOW, points (2) will be LOW. Since button <0, 1> is pressed, this causes points (3) to become LOW, and since button <0, 2> is pressed, point (4), which is on column 2, will follow suit and column 2 will read LOW. Hence, you will recognize a phantom button. You will also produce a short between rows 0 and 1 in the process, so the keypad rows and columns must be current protected.

This effect can be removed by using diodes in the connections made by the buttons (in the direction from the columns to the rows), as depicted in Figure below. The diodes will only let current pass in one direction and will act as a break in the other direction. In consequence, they will allow a column to change from HIGH to LOW if a button is pressed in the currently read row, but they will not allow a row that is HIGH to be pulled to LOW by a LOW column. In a matrix keypad which uses such diodes, you will not recognize any phantom buttons. However, cheap keypads come without diodes, and in their case there is no way you can avoid recognizing phantom buttons.

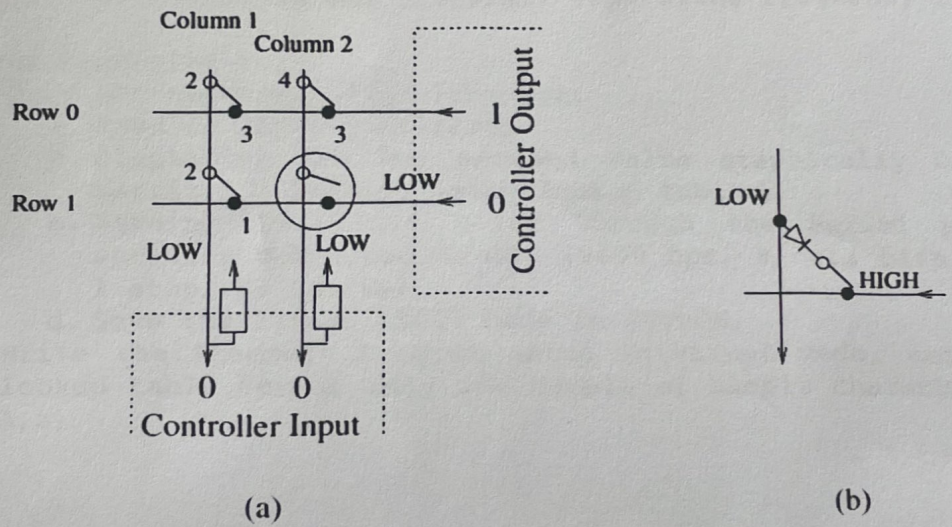

Fig.2: Recognition of a phantom button (a) and suppressing the effect with a diode (b)

Questions:

In below problem the method requested is detecting more than one key press at time and encode that information in single 8 bits variable. For example this keypad scan routine can be used to detect key presses like SHIFT+C in computer keypads.

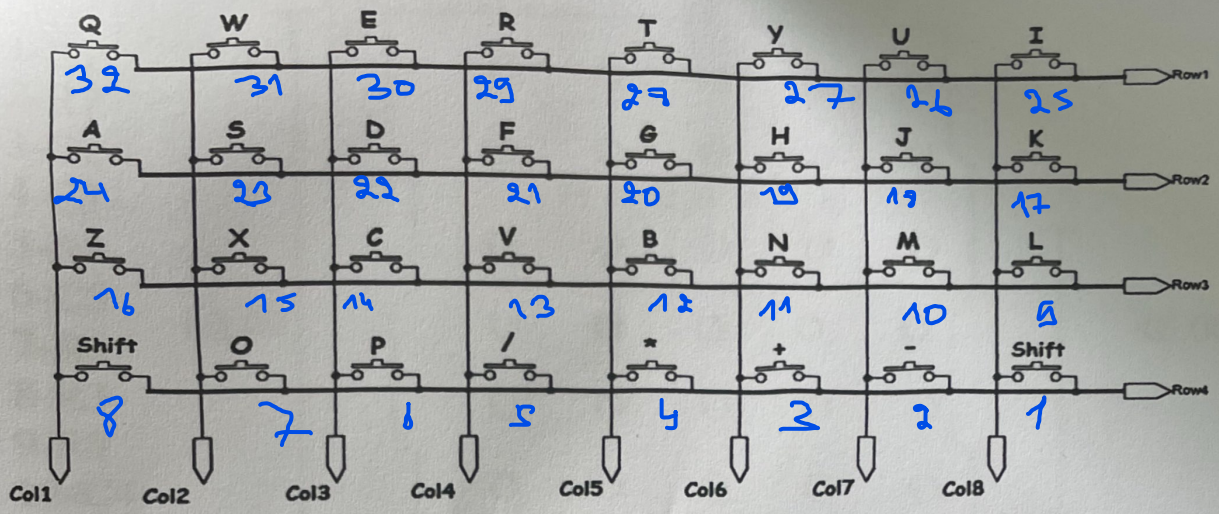

Fig.3: 4x8 Keypad

Characters supported are:
a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,/,*,+,-
Using the (RIGHT or LEFT shift) button:
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,/,*,+,-
The Microcontroller shall be able to display the pressed key on an 8x5 Matrix LED as shown in Fig.4.

The Microcontroller is the PIC16F877 (operating frequency is 4Mhz).

You are requested to:

1- Draw the necessary Algorithm for:
   a. Reading any key pressed,
   b. Displaying the key pressed value graphically on the 8x5 Matrix LED Display using lookup tables,
   c. Sending the ASCII value through the Serial port to a serially connected device (9600 bps, 8 data bits, 1 start, 1 stop, No Parity)
   d. Save the latest ASCII code in EEPROM.
2- Write the Assembly Program using interrupt mode, and draw the lookup table for at only one couple of sample characters (i.e.: A,a)

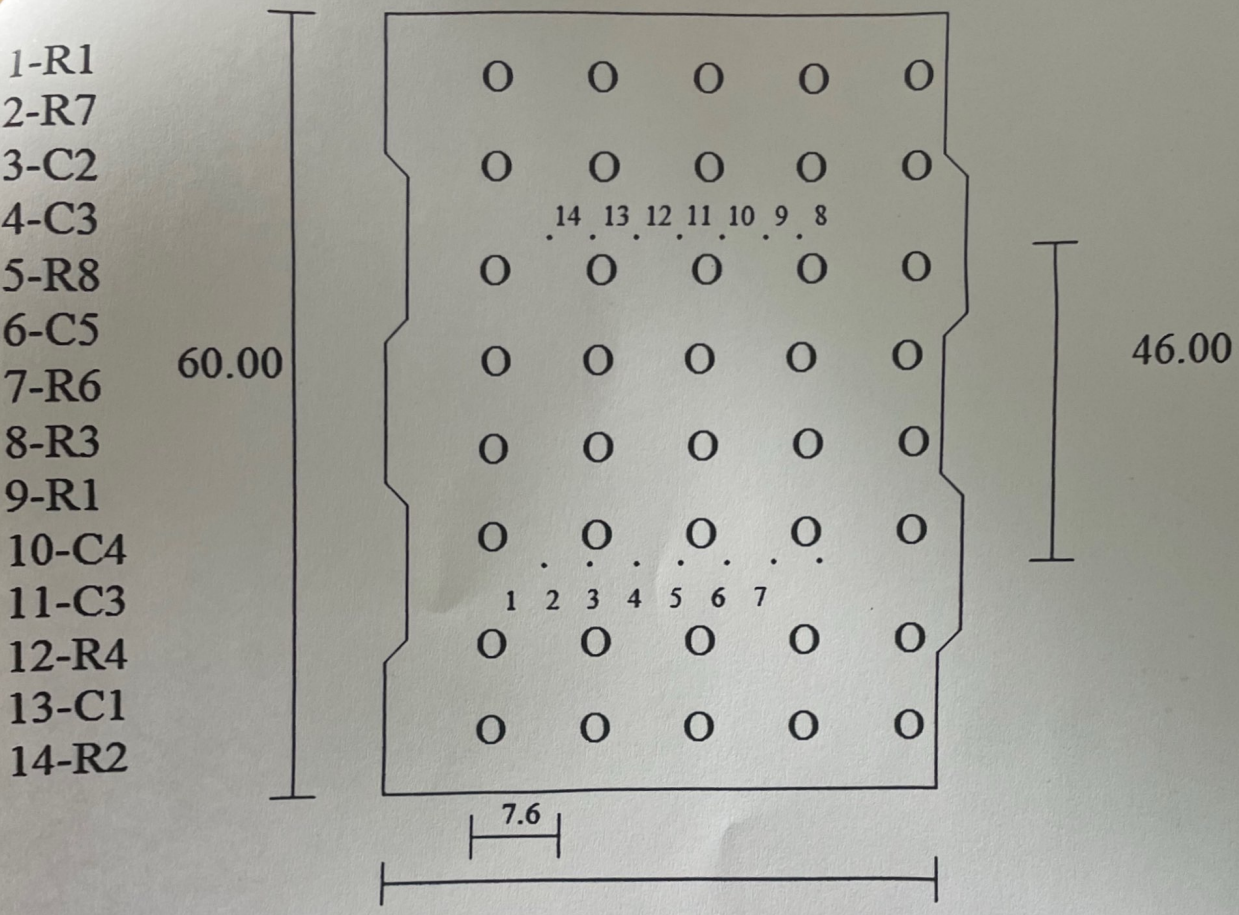

Fig.4: 8x5 LED Matrix Display