TECLADO MATRICIAL

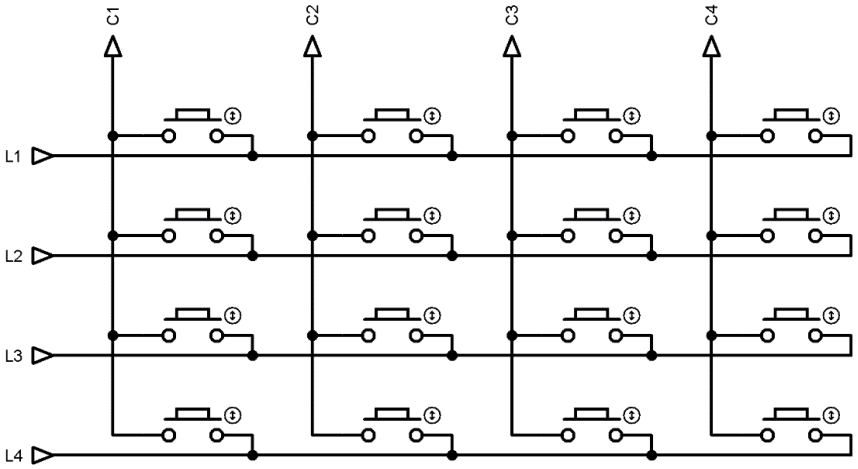
Universidade Tecnológica Federal do Paraná - UTFPR
Sistemas Microcontrolados
Monitor da disciplina: Luís Paulo Custódio

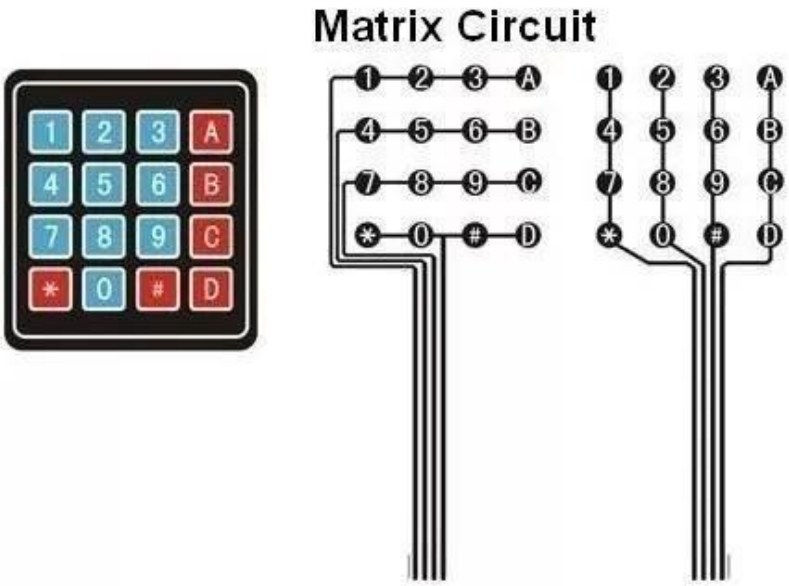
Introdução

Teclados são geralmente utilizados em aplicações na qual o usuário precisa interagir com um sistema, como computadores, calculadoras, controles remotos entre outros. Utilizando um teclado com 64 botões, caso cada botão fosse ligado diretamente a um pino do microcontrolador, seriam gastos 64 pinos o que tornaria a implementação impossível para a maioria dos microcontroladores. Para evitar este problema, as teclas são conectadas no formato de matriz 8×8 ocupando apenas 16 pinos. Essa técnica é chamada de multiplexação, para realizar a leitura das teclas.

O Teclado Matricial

Este teclado como o nome indica é formado de botões organizados em linhas e colunas de modo a formar uma matriz. Quando pressionado um botão, conecta a linha com a coluna na qual está ligado. A figura a seguir ilustra a ligação matricial.





O teclado matricial (neste caso) possui a seguinte pinagem (da esquerda para a direita):

Pino 1 – Primeira Linha (L1)
Pino 2 – Segunda Linha (L2)
Pino 3 – Terceira Linha (L3)
Pino 4 – Quarta Linha (L4)
Pino 5 – Primeira Coluna (C1)
Pino 5 – Segunda Coluna (C2)
Pino 5 – Terceira Coluna (C3)
Pino 8 – Quarta Coluna (C4)

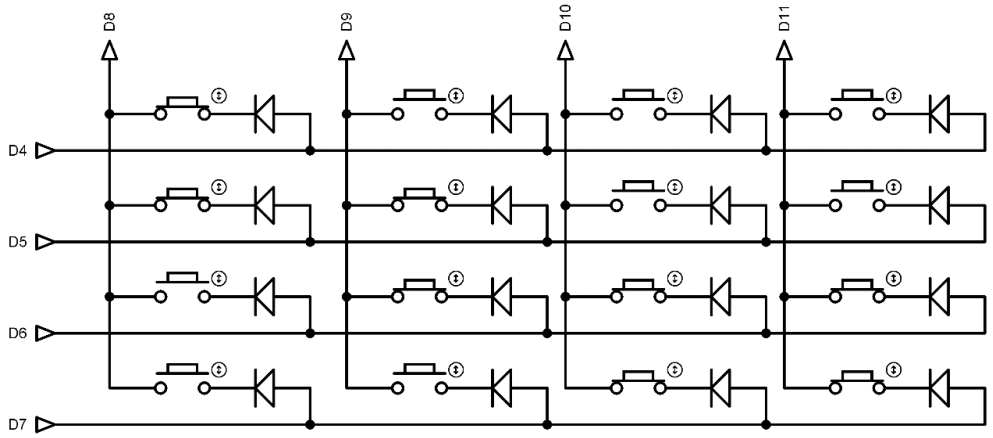
A Multiplexação

Essa técnica consiste no compartilhamento do mesmo barramento por vários dispositivos, entretanto apenas um deles utilizará o barramento por vez. No caso do teclado matricial, os barramentos serão as linhas do teclado e os dispositivos as colunas. Permite-se que apenas uma coluna se ligue as linhas por vez. Para desconectar as colunas que não devem ser lidas deve-se configurá-las como entradas (alta impedância).

No início os pinos conectados às linhas ou às colunas serão configurados como entradas com *pull up* ou *pull down* e as colunas ou linhas como entradas (alta impedância). Isso dependerá da lógica de acionamento utilizada no código do programa. A varredura consiste em ativar uma coluna por vez (saída em nível lógico alto ou baixo) e checar se houve uma alteração nas linhas. Esta é a forma mais usada de varredura por colunas. Caso uma alteração em uma linha seja identificada, o *bounce* da tecla deve ser devidamente tratado para que possa finalmente afirmar que o botão foi pressionado.

Pressionando várias teclas

Quando pressionadas 3 ou mais teclas um efeito conhecido como tecla fantasma pode ocorrer. Caso a tecla fantasma seja pressionada e em seguida uma das teclas anteriores for solta, a tecla que foi solta ainda será considerada como pressionada. Para solucionar este problema deve-se adicionar um diodo em cada botão para evitar que estes caminhos indesejados sejam formados, como mostra a figura abaixo.



Biblioteca do MikroC

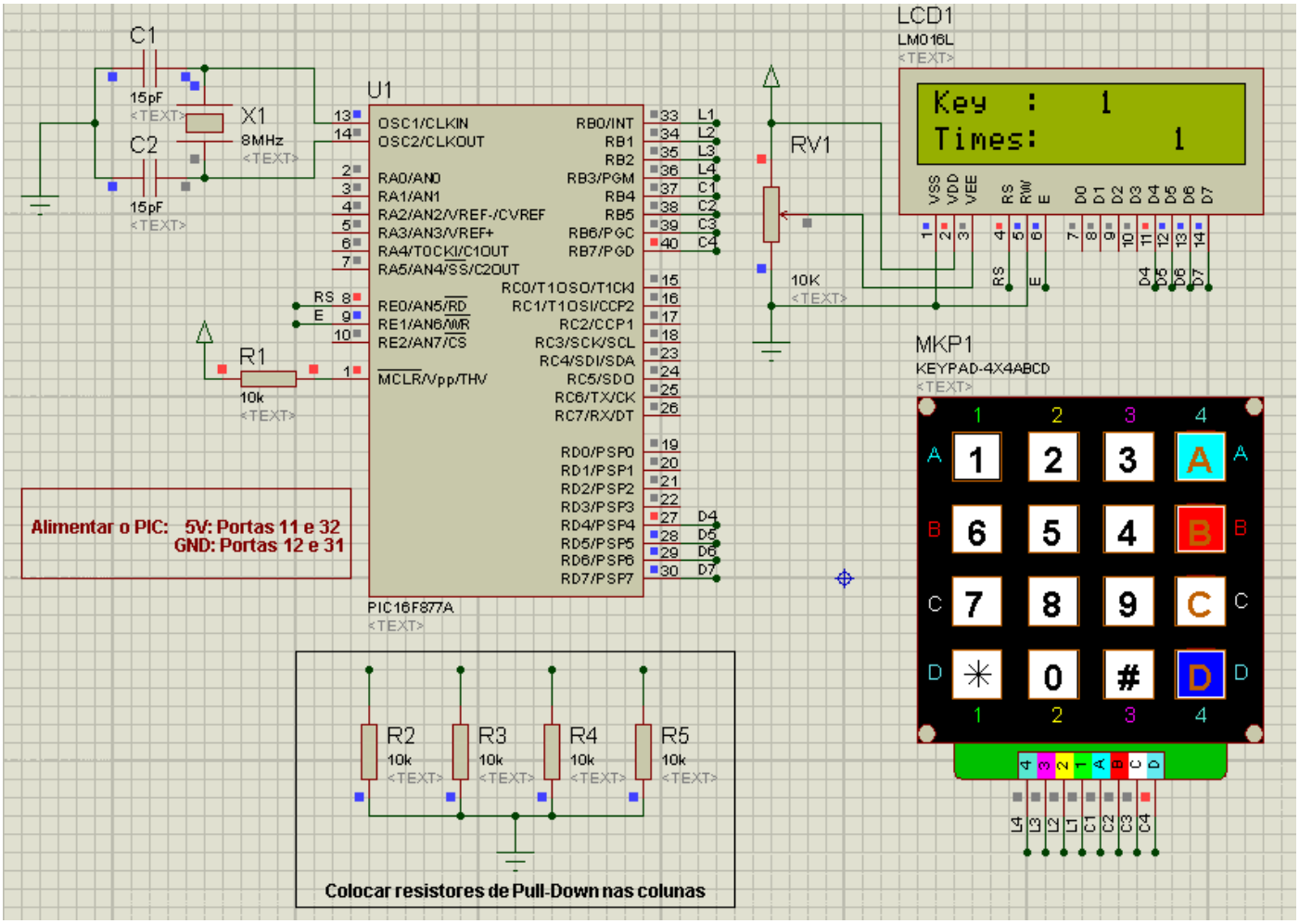
Esta biblioteca já vem pronta para usar o teclado matricial de 4x3 ou 4x4 (linhas x colunas) do MikroC PRO (algumas mudanças foram feitas em relação ao código original do programa). Apenas verificar as portas de conexão do teclado (no exemplo está no PORTB) e do display LCD (RS no pino RE0 e E no pino RE1, D4 a D7 nos pinos RD4 a RD7).

Este código simplesmente mostra o valor digitado em formato ASCII e mostra também quantas vezes este valor foi digitado.

Esta biblioteca está codificada para teclados matriciais de 4x3 e 4x4, portanto, para teclados maiores, é necessário um código diferente.

```
// Início do código

unsigned short kp, cnt, oldstate = 0;
char txt[6];

// Conexões do teclado matricial no PORTB
char  keypadPort at PORTB;

// Conexões do display LCD
sbit LCD_RS at RE0_bit;
sbit LCD_EN at RE1_bit;
sbit LCD_D4 at RD4_bit;
sbit LCD_D5 at RD5_bit;
sbit LCD_D6 at RD6_bit;
sbit LCD_D7 at RD7_bit;

sbit LCD_RS_Direction at TRISE0_bit;
sbit LCD_EN_Direction at TRISE1_bit;
sbit LCD_D4_Direction at TRISD4_bit;
sbit LCD_D5_Direction at TRISD5_bit;
sbit LCD_D6_Direction at TRISD6_bit;
sbit LCD_D7_Direction at TRISD7_bit;

void main() {
  ADCON1 = 0X06;                          // E/S digital
  cnt = 0;                                // Reseta o contador
  Keypad_Init();                          // Inicializa a função Keypad
  Lcd_Init();                             // Inicializa o LCD
  Lcd_Cmd(_LCD_CLEAR);                    // Limpa o display
  Lcd_Cmd(_LCD_CURSOR_OFF);               // Desliga o cursor
  Lcd_Out(1, 1, "1");                     // Escreve no LCD na linha 1 e coluna 1
  Lcd_Out(1, 1, "Key  :");                // Escreve o valor digitado no LCD
  Lcd_Out(2, 1, "Times:");                // Escreve quantas vezes o valor foi digitado no LCD

  do {
    kp = 0;                               // Reseta a variável kp

    // Aguarda até alguma tecla ser pressionada
    do

   // Estas duas funções têm a mesma função, a diferença está que a Key_Press não trava a varredura enquanto uma tecla está pressionada

   // kp = Keypad_Key_Press();             // Armazena a tecla pressionada na variável kp
      kp = Keypad_Key_Click();             // Armazena a tecla pressionada na variável kp

 while (!kp);
  // Prepara o valor para saída e transforma a tecla pressionada em valor ASCII

  switch (kp) {

      // Descomente este bloco de funções se o teclado for de 4x3
```

```
      //case 10: kp = 42; break;  // '*'
      //case 11: kp = 48; break;  // '0'
      //case 12: kp = 35; break;  // '#'
      //default: kp += 48;

      // Descomente este bloco de funções se o teclado for de 4x4
      case  1: kp = 49; break; // 1
      case  2: kp = 50; break; // 2
      case  3: kp = 51; break; // 3
      case  4: kp = 65; break; // A
      case  5: kp = 52; break; // 4
      case  6: kp = 53; break; // 5
      case  7: kp = 54; break; // 6
      case  8: kp = 66; break; // B
      case  9: kp = 55; break; // 7
      case 10: kp = 56; break; // 8
      case 11: kp = 57; break; // 9
      case 12: kp = 67; break; // C
      case 13: kp = 42; break; // *
      case 14: kp = 48; break; // 0
      case 15: kp = 35; break; // #
      case 16: kp = 68; break; // D

    }

    if (kp != oldstate) {                    // Se for pressionada uma tecla diferente da anterior
      cnt = 1;                               // Contador muda de valor
      oldstate = kp;                         // A tecla anterior passa a ser a atual
      }
    else {                                   // Se a tecla for a mesma que a anterior
      cnt++;                                 // Incrementa na variável contador
      }

    Lcd_Chr(1, 10, kp);                      // Mostra o valor pressionado em ASCII no LCD

    if (cnt == 255) {                        // Se a variável contador chegar a 255, ela estoura
      cnt = 0;                               // Reseta contador
      Lcd_Out(2, 10, "   ");
      }

    WordToStr(cnt, txt);                     // Transforma  o valor do contador em string
    Lcd_Out(2, 10, txt);                     // Mostra o valor do contador no LCD
  } while (1);                               // Laço infinito para outras ações
}

// Fim do código
```



Os resistores de *pull down* não são necessários para a simulação no Proteus, porém na montagem física do circuito, talvez seja necessário colocá-los para evitar que o teclado mostre teclas que não foram realmente pressionadas no display LCD. Valores de 4,7kΩ a 10kΩ são apropriados. As conexões do teclado matricial podem variar dependendo do modelo utilizado, portanto, verifique quais saídas são correspondentes às linhas e às colunas.

Algoritmo de exemplo

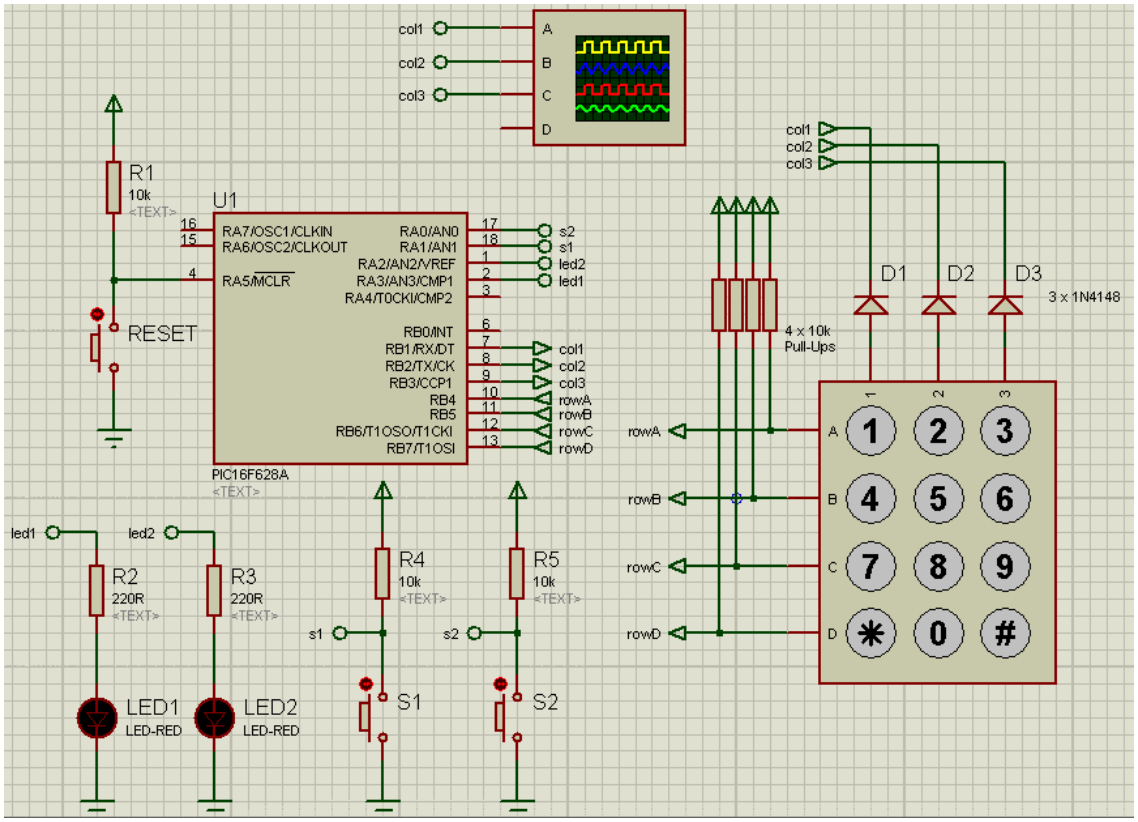
Este exemplo foi retirado do site: https://www.youtube.com/watch?v=ZKJ4wM3690I&list=PLZ8dBTV2_5HQTv6DRKZTp9L0iRReXis0q&index=25

Acesse-o para melhores explicações do funcionamento deste código e do circuito.

```
#define S2      RA0_bit
#define S1      RA1_bit
#define led2    RA2_bit
#define led1    RA3_bit

#define col_1   RB1_bit
#define col_2   RB2_bit
#define col_3   RB3_bit
#define row_A   RB4_bit
#define row_B   RB5_bit
#define row_C   RB6_bit
#define row_D   RB7_bit
```

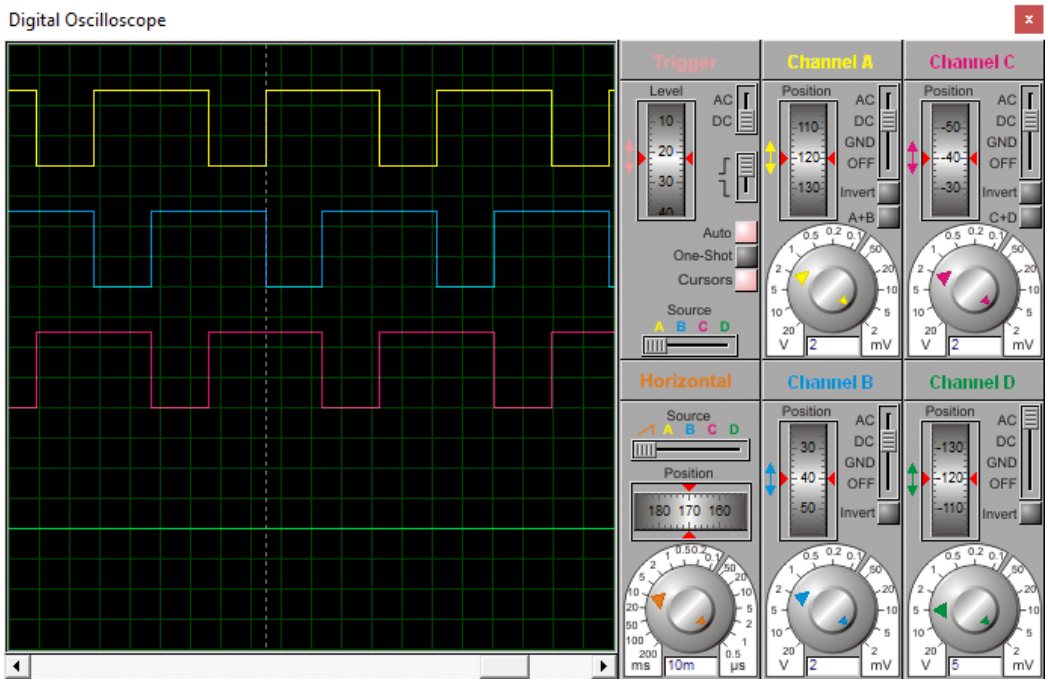
```
void pulse(char number);                    //Função para pulsar o led1

char control = 0x01;

void interrupt()
{
   if(T0IF_bit)                             //Houve estouro do Timer0?
   {                                        //Sim...

    T0IF_bit = 0x00;                        //Limpa a flag
    TMR0     = 0x6C;                        //Reinicia o timer0

    if(col_1 && control == 0x01)            //Coluna 1 em nível high? Control igual 1?
    {                                       //Sim...
      control = 0x02;
      col_1 = 0x00;                         //Apenas a coluna 1 em nível baixo
      col_2 = 0x01;
      col_3 = 0x01;

      if(!row_A)      pulse(1);
      else if(!row_B) pulse(4);
      else if(!row_C) pulse(7);
      else if(!row_D) pulse(11);

    }
    else if(col_2 && control == 0x02)       //Coluna 2 em nível high? Control igual 2?
    {                                       //Sim...
      control = 0x03;
      col_1 = 0x01;                         //Apenas a coluna 2 em nível baixo
      col_2 = 0x00;
      col_3 = 0x01;

      if(!row_A)      pulse(2);
      else if(!row_B) pulse(5);
      else if(!row_C) pulse(8);
      else if(!row_D) pulse(10);

    }
    else if(col_3 && control == 0x03)       //Coluna 3 em nível high? Control igual 3?
    {                                       //Sim...
      control = 0x01;
      col_1 = 0x01;                         //Apenas a coluna 3 em nível baixo
      col_2 = 0x01;
      col_3 = 0x00;

      if(!row_A)      pulse(3);
      else if(!row_B) pulse(6);
      else if(!row_C) pulse(9);
      else if(!row_D) pulse(12);

    }
```

```
    }

} //end interrupt


// --- Função Principal
void main()
{
    CMCON      = 0x07;                    //Desabilita os comparadores
    OPTION_REG = 0x86;                    //Timer0 incrementa com ciclo de instrução, prescaler 1:128
    GIE_bit    = 0x01;                    //Habilita interrupção global
    PEIE_bit   = 0x01;                    //Habilita interrupção por periféricos
    T0IE_bit   = 0x01;                    //Habilita interrução do Timer0

    TMR0       = 0x6C;                    //Inicia o timer0

    TRISA = 0x03;                         //Entrada em RA0 e RA1
    TRISB = 0xF0;                         //Nibble mais significativo do PORTB será entrada
    PORTA = 0x03;                         //RA0 e RA1 iniciam em high
    PORTB = 0xFF;                         //Nibble mais significativo inicia em high.


    while(1)                              //Loop Infinito
    {


    } //end while

} //end main


void pulse(char number)
{
  char i;                                 //variável de iterações

  for(i=0;i<number;i++)
  {
    led1 = 0x01;
    delay_ms(200);
    led1 = 0x00;
    delay_ms(200);

  }
}
```



Neste exemplo, o circuito mostra o funcionamento do teclado matricial e quando pressionado o número correspondente no teclado, o LED1 pisca este número de vezes. Por exemplo, se pressionado a tecla correspondente ao número 5, o LED1 irá piscar 5 vezes. A varredura do circuito é feita a cada estouro do TIMER0 acionando pela condição lógica “se” e comparando com a variável de controle “control” para ligar e desligar cada coluna por vez. Diferente do exemplo utilizado pelo MikroC, a lógica é outra, funcionando com nível alto nas linhas, e quando pressionado alguma tecla, identifica nível baixo nas portas do PIC, por isso a necessidade de utilizar resistores de *pull up*. Já a função pulse() determina o número de vezes que a saída referente ao LED1 deve ficar alternando entre nível alto e baixo, ou seja, quantas vezes o led irá piscar. Os valores referentes a “* 0 #” do teclado foram arbitrados como respectivamente: “11 10 12”. Portanto, é possível modificar qualquer valor no teclado através do código.

O osciloscópio conectado nas saídas do PIC, referente às colunas, mostra o tempo que as colunas ficam em nível baixo em determinado período.



Referências:

http://www.embarcados.com.br/
https://www.youtube.com/watch?v=ZKJ4wM3690I&list=PLZ8dBTV2_5HQTv6DRKZTp9L0iRReXis0q&index=25