4 آشنایی با LCD کاراکتری و صفحه کلید ماتریسی

در این فصل، نمایشگر LCD و صفحه‌کلید ماتریسی بررسی می شود. صفحه کلید به روش سرکشی خوانده می‌شود و با توجه به وقت‌گیر بودن این پروسه از وقفه‌ها استفاده خواهد شد.

4.1 معرفی LCD کاراکتری

یک عدد LCD کاراکتری از نوع 16x2 (دارای 16 ستون و 2 ردیف) با نور پس زمینه‌ی به رنگ سبز یا آبی تحت بلوکی با عنوان 16x2 Character LCD به منظور نمایش جملات و اطلاعات دلخواه در پکیج آموزشی قرار داده شده است.

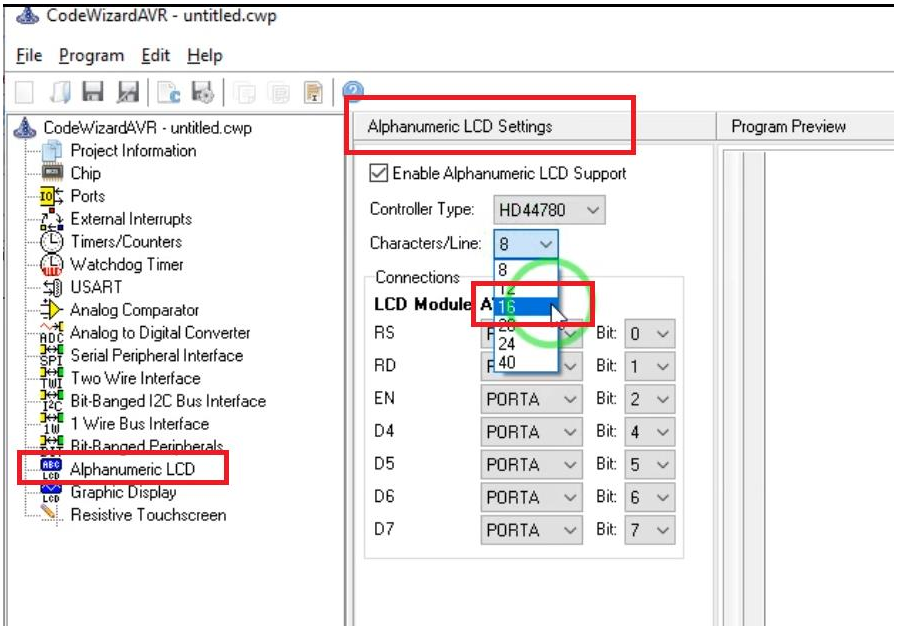
ارتباط ریزپردازنده و LCD را می‌توان به دو صورت برقرار کرد:

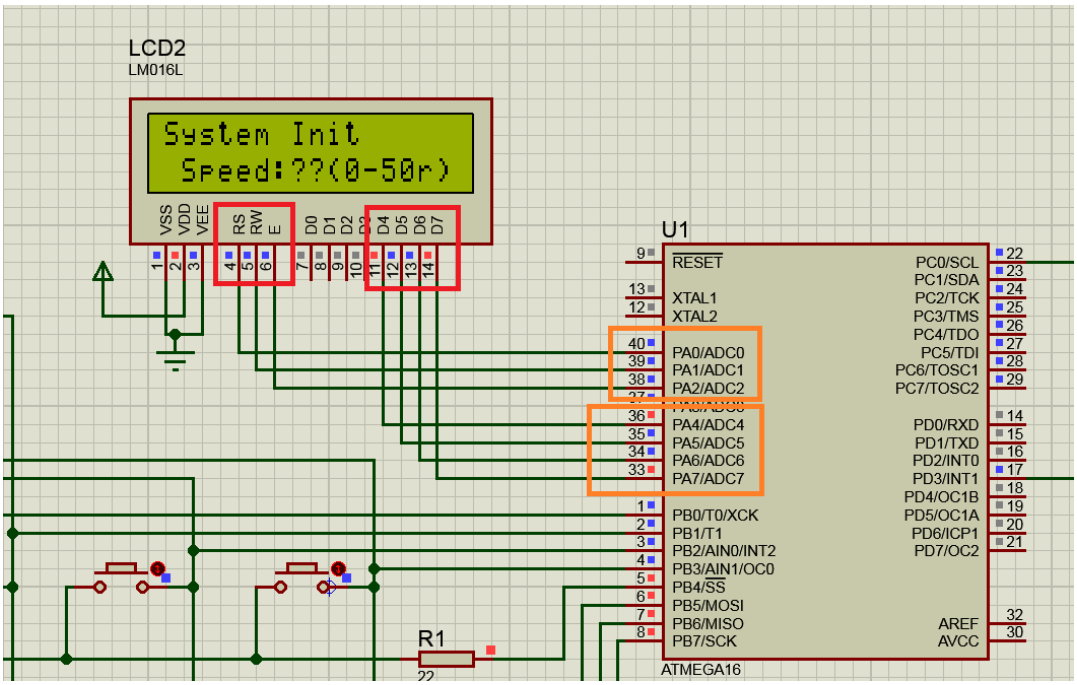
ارتباط 4 سیمه : سریال

ارتباط 8 سیمه : موازی

ارتباط سریال:

تعداد خط های کمتری از میکرو، درگیر میشوند و به راحتی به وسیله ی code wizard در code vision میتوانیم برنامه نویسی کنیم. (در code vision فقط حالت سریال پشتیبانی میشود)





اضافه شدن هدرفایل زیر به برنامه

```
// Alphanumeric LCD functions
#include <alcd.h>

// Declare your global variables here
```

تعریف اولیه ی lcd

```
// Alphanumeric LCD initialization
// Connections are specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTA Bit 0
// RD - PORTA Bit 1
// EN - PORTA Bit 2
// D4 - PORTA Bit 4
// D5 - PORTA Bit 5
// D6 - PORTA Bit 6
// D7 - PORTA Bit 7
// Characters/line: 16
lcd_init(16);
```

پایه هایی که به lcd متصل میشوند را باید به عنوان خروجی تعریف کنیم.

توی تابع lcd_init() خودش میاد پورت های مورد نظر را خروجی تعریف میکند => نیازی نیست ما کاری کنیم.

اگه وارد فایل alcd.h بشیم، میتوانیم زیربرنامه هایی که قابل استفاده برای کارکردن با lcd است را ببینیم.

C:\cvavr\inc\alcd.h

Notes | a.c | alcd.h *

```
13  #define _LCDX_INCLUDED_
14
15  void _lcd_write_data(unsigned char data);
16  /* read a byte from the LCD character generator or display RAM */
17  unsigned char lcd_read_byte(unsigned char addr);
18  /* write a byte to the LCD character generator or display RAM */
19  void lcd_write_byte(unsigned char addr, unsigned char data);
20  // set the LCD display position  x=0..39 y=0..3
21
22  void lcd_gotoxy(unsigned char x, unsigned char y);
23  // clear the LCD
24  void lcd_clear(void);
25  void lcd_putchar(char c);
26  // write the string str located in SRAM to the LCD
27  void lcd_puts(char *str);
28  // write the string str located in FLASH to the LCD
29  void lcd_putsf(char flash *str);
30  // write the string str located in EEPROM to the LCD
31  void lcd_putse(char eeprom *str);
32  // initialize the LCD controller
```

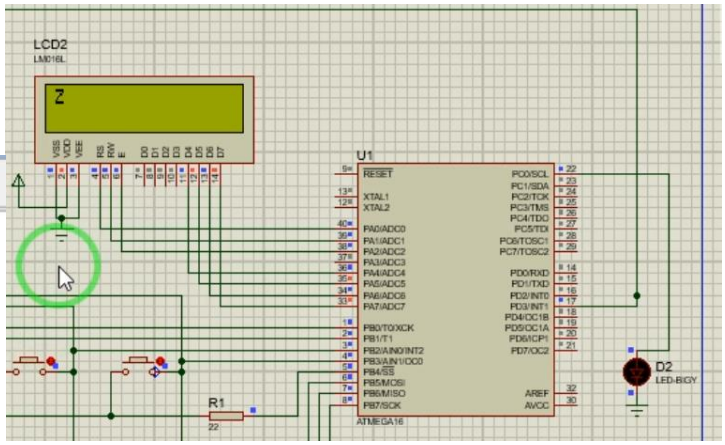
```
// initialize the LCD controller
void lcd_init(unsigned char lcd_columns);
```

ورودی lcd_init: تعداد کاراکترهایی که روی هر خط نمایش داده میشود.

برنامه ی خیلی ساده:نمایش یک کاراکتر روی ال سی دی

Notes | a.c | alcd.h

```
67  // D5 - PORTA Bit 5
68  // D6 - PORTA Bit 6
69  // D7 - PORTA Bit 7
70  // Characters/line: 16
71  lcd_init(16);
72  lcd_putchar('Z');
```



نمایش یک عدد روی lcd : باید عدد مورد نظر را داخل یک متغیر که رشته است بریزیم و به کمک lcd_puts() نمایش بدیم.

استفاده از دستور sprint()

اینکلود کردن هدرفایل stdio.h

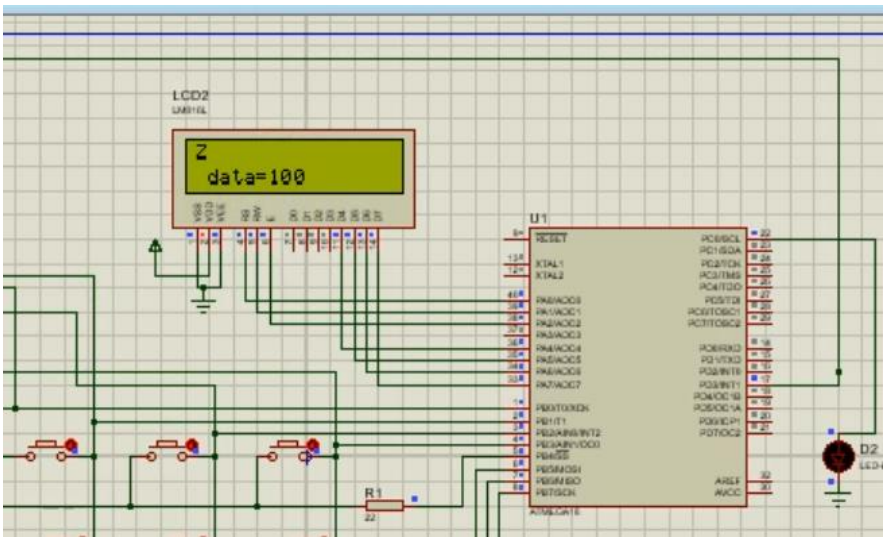
C:\cvavr\BIN\work14002\session3\lcd\a.c

Notes | a.c | alcd.h

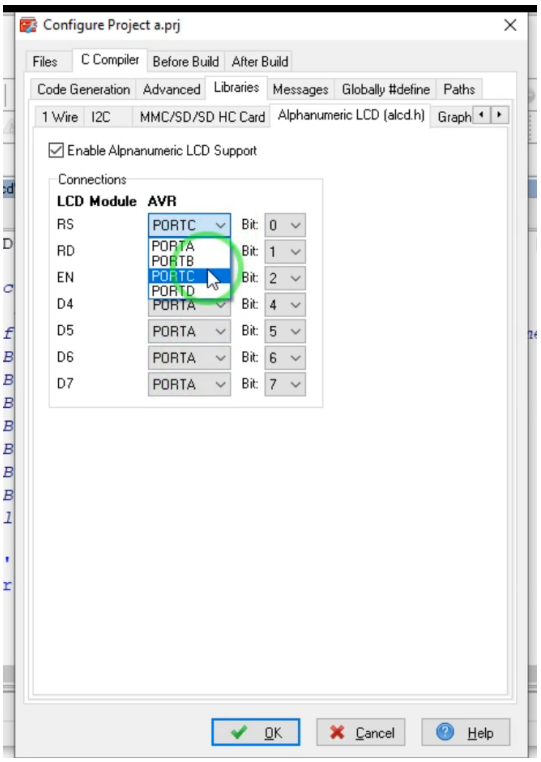
```
22  ****************************************
23
24  #include <mega16.h>
25
26  // Alphanumeric LCD functions
27  #include <alcd.h>
28  #include <stdio.h>
29  // Declare your global variables here
```

```
void main(void)
{
// Declare your local variables here
 char scr[20];
```

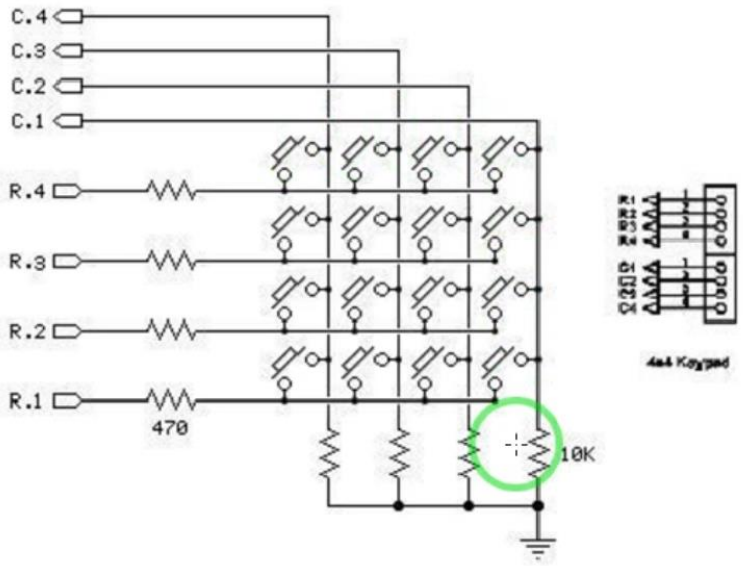
```
71  lcd_init(16);
72  lcd_putchar('Z');
73  sprintf(scr,"\r\n data=%d",100);
74  lcd_puts(scr);
75
76
77  while (1)
78        {
79        // Place your code here
80
81        }
82  }
83
```



تغییر دادن پورت یا بیت های پورتی که ال سی دی بش وصل شده بعد از ساخت پروژه:

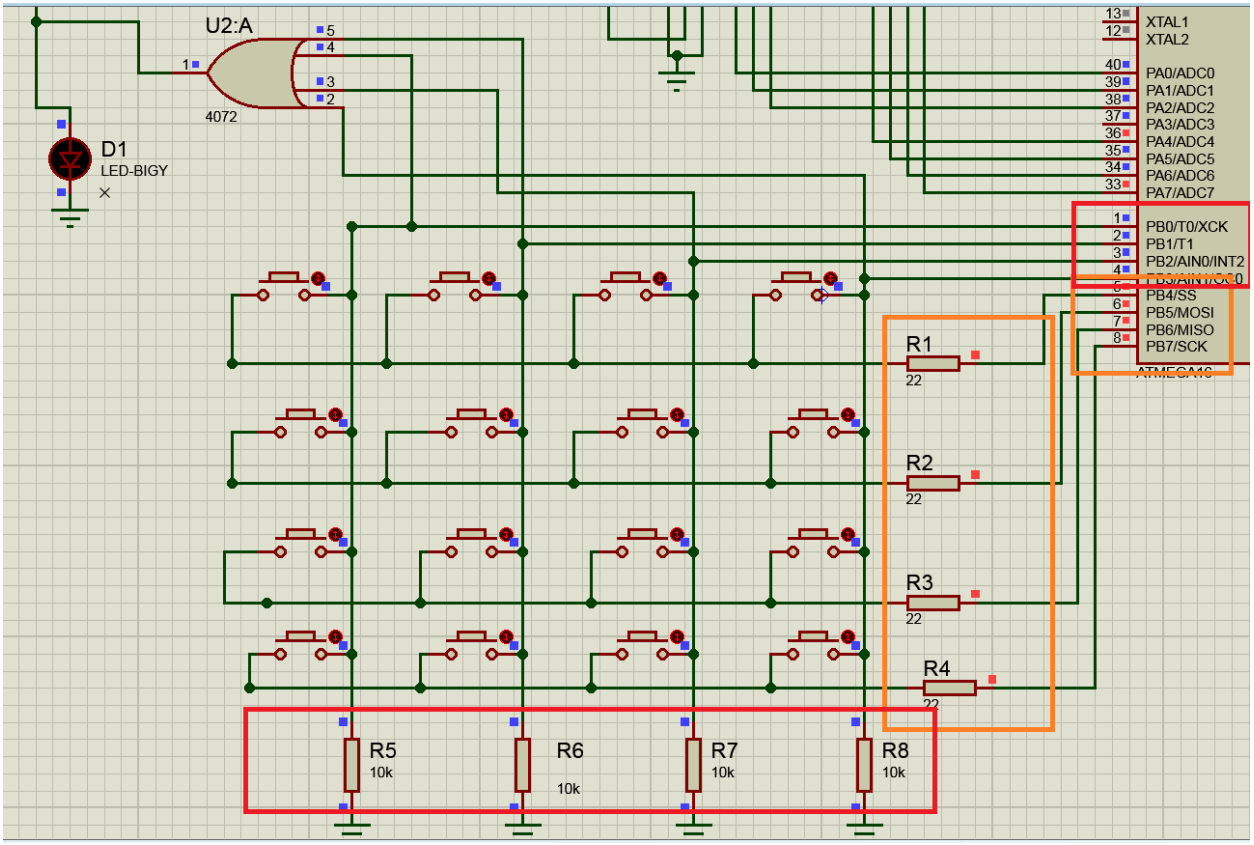


Keypad



شکل 4-4: ساختار صفحه کلید ماتریسی استفاده شده در برد آموزشی

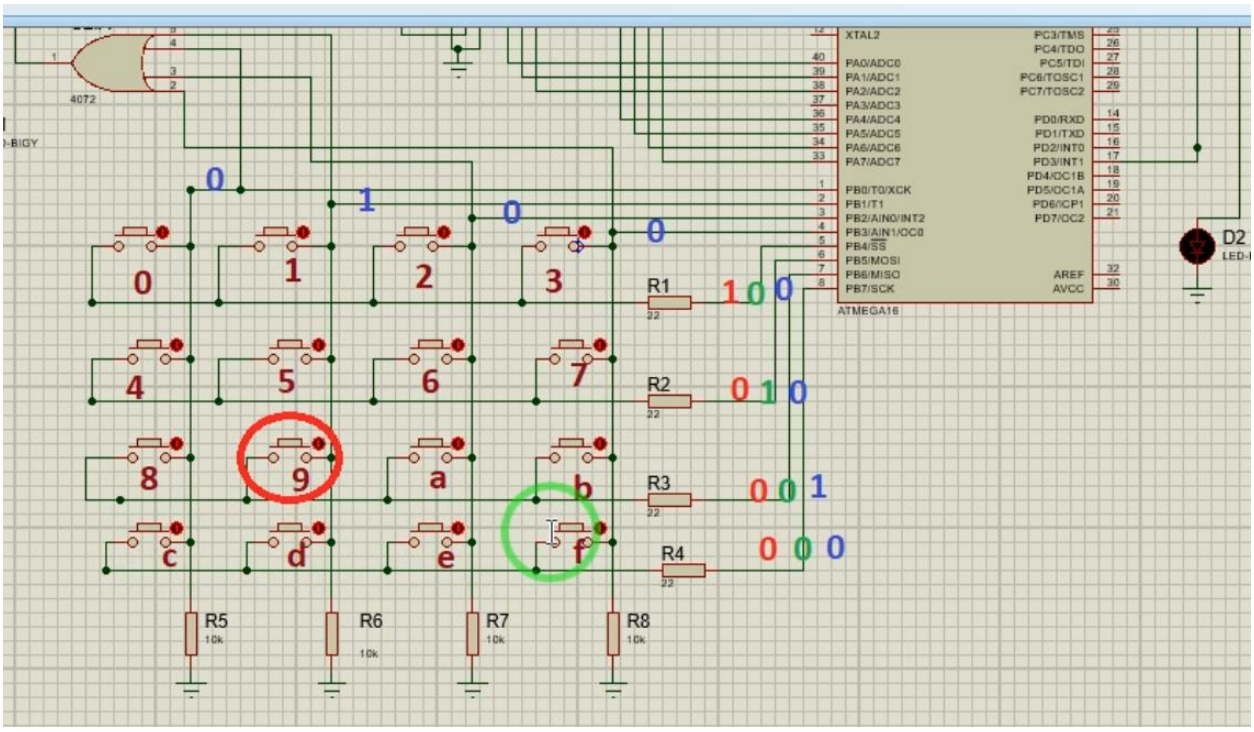
Keypad شامل ۱۶تا کلید پوش باتن است. که شامل ۴ تا ردیف و ۴ تا ستون است.



ردیف ها که از مقاومت های R1 تا R4 هستند => به بیت های 4 تا 7 از پورت B وصل شده اند.
ستون ها از مقاومت های R5 تا R8 هستند => به بیت های 0 تا 3 از پورت B وصل شده اند.

ردیف ها به صورت خروجی => عدد یک را قرار میدیم.

ستون ها به صورت ورودی => تا زمانی که کلیدی فشرده نشه => ستون ها یا ورودی ها، صفر هستند.



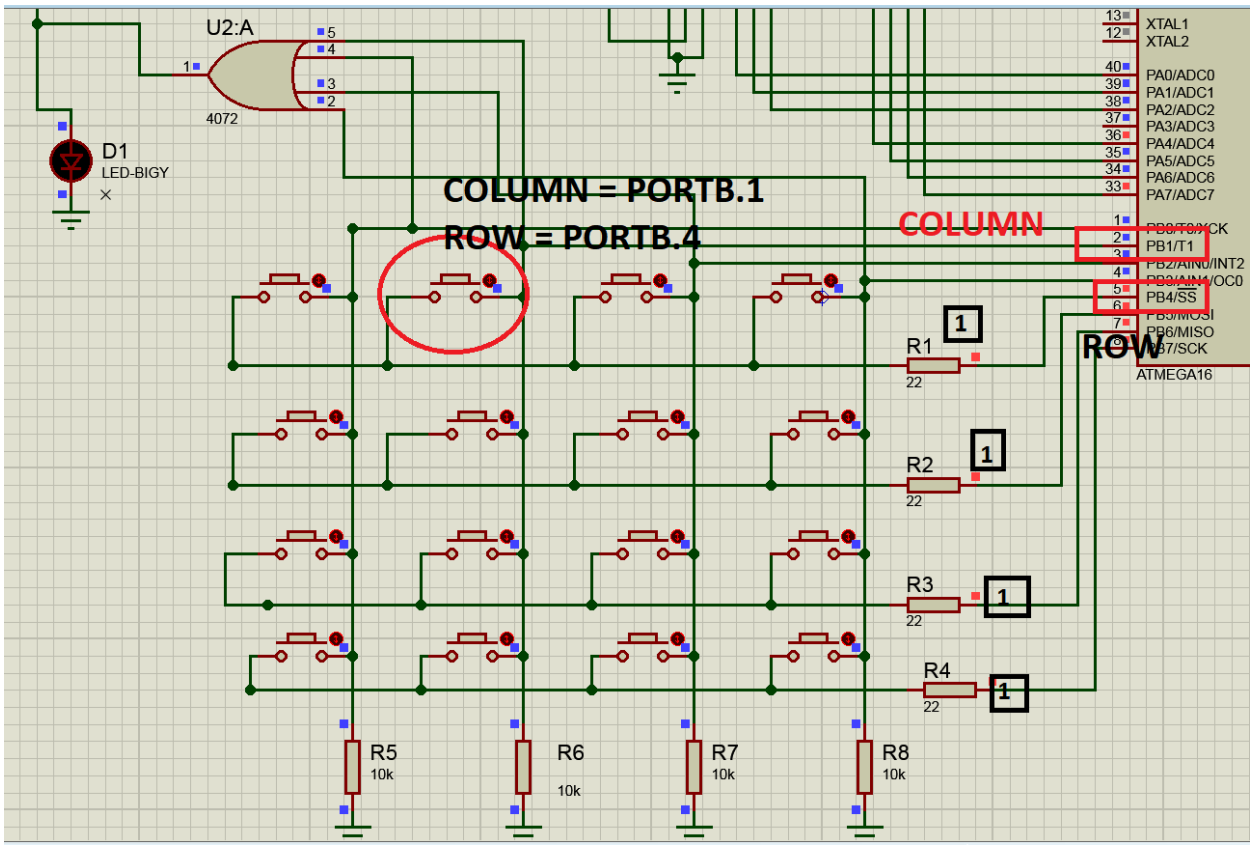
اگه کلیدی که دورش دایره کشیده را فشار بدیم => توی ماتریسی که داریم میریم شماره خانه ی ۹ را میبینیم که مقدارش چیه

پوش باتن در ردیف ۳ و ستون ۳ از راست فشرده شده

شان داده شده، نگاشت می کند.

```
char data_key[]={
'7','8','9','a',
'4','5','6','b',
'1','2','3','c',
'*','0','#','d'};
```

مثلا اگه کلیدی در موقعیت ۱۵م فشرده شد => کاراکتر d نمایش داده میشود.



قبل از اجرای زیربرنامه ی keypad

باید ۱. پورت هایی که به کی پد متصل هستند را به عنوان خروجی درنظر بگیریم.

۲. یک مقداردهی اولیه کنیم

یعنی ۴ بیت کم ارزش که ورودی است را صفر بگذاریم و ۴ بیت پرارزش که خروجی است را یک بگذاریم.

شناسایی موقعیت کلید فشرده شده:

```
char keypad2(void)
{
    char key=100;
    for (r=0;r<4;r++)
    {
    PORTB=row[r]; //row= 0x10,0x20,0x40,0x80

    c=20;
    delay_ms(10);
    if (PINB.0==1) c=0;
    if (PINB.1==1) c=1;
    if (PINB.2==1) c=2;
    if (PINB.3==1) c=3;


        if (!(c==20)){
           key=(r*4)+c;
           PORTB=0xf0;
           while (PINB.0==1) {}
           while (PINB.1==1) {}
           while (PINB.2==1) {}
           while (PINB.3==1) {}
        }
     PORTB=0xf0;
    }
    return key;

}
```

متناسب با یک کردن هرکدام از ردیف ها، داده ای را روی پورت قرار میده

0001 0000 , 0010 0000 , 0100 0000 , 1000 0000

شناسایی ستون

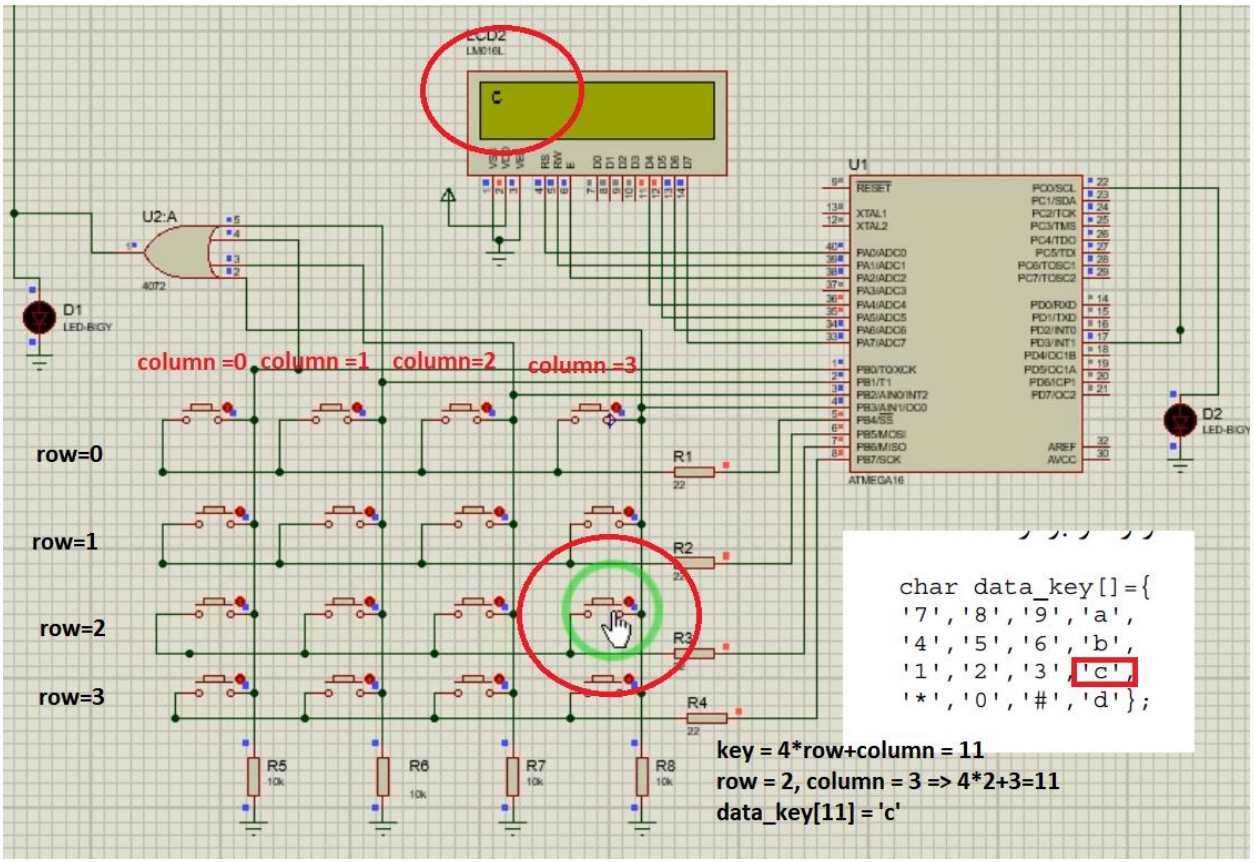
برای ردیف ها، با ۴ بیت پرارزش کار میکنه و هر ردیف، یه بیت از بیت های پرارزش را تغییر میده.

صبر میکنه تا کلید رها بشه

ردیف را در ۴ ضرب به علاوه ی شماره ی ستون میکنه تا موقعیت کلید فشرده شده را سیو کنه

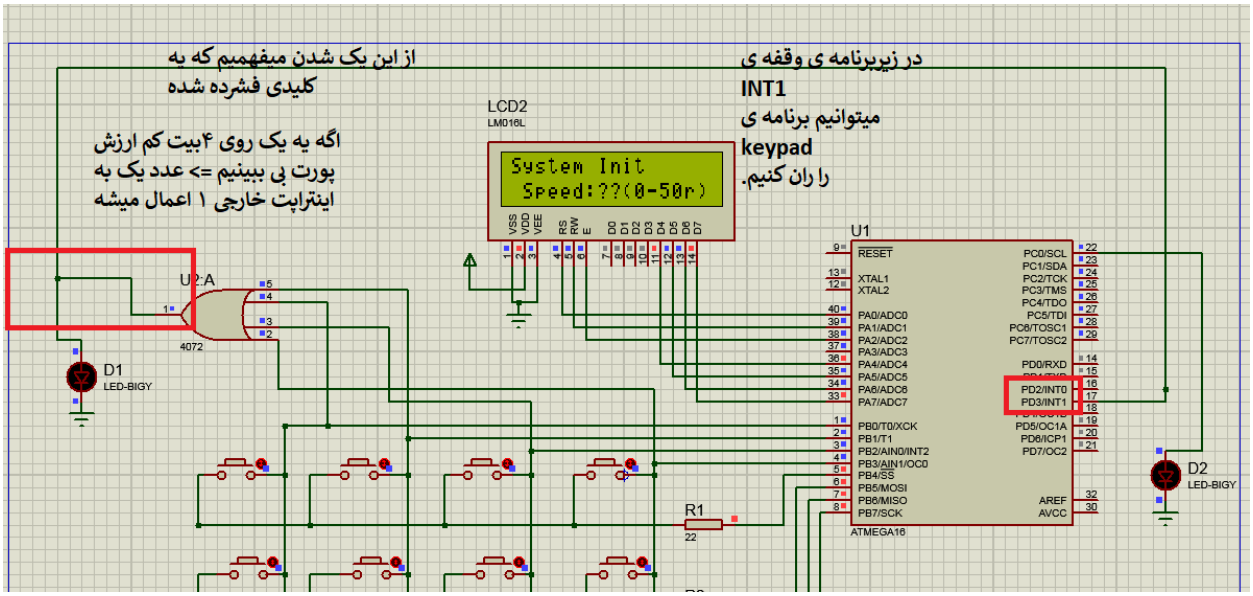
این برنامه ، کلید را برگشت میزنه که کلید همان موقعیت پیدا شده است

بعدش از ماتریسی که در برنامه داریم، داده ی مورد نظر را پیدا میکنیم.

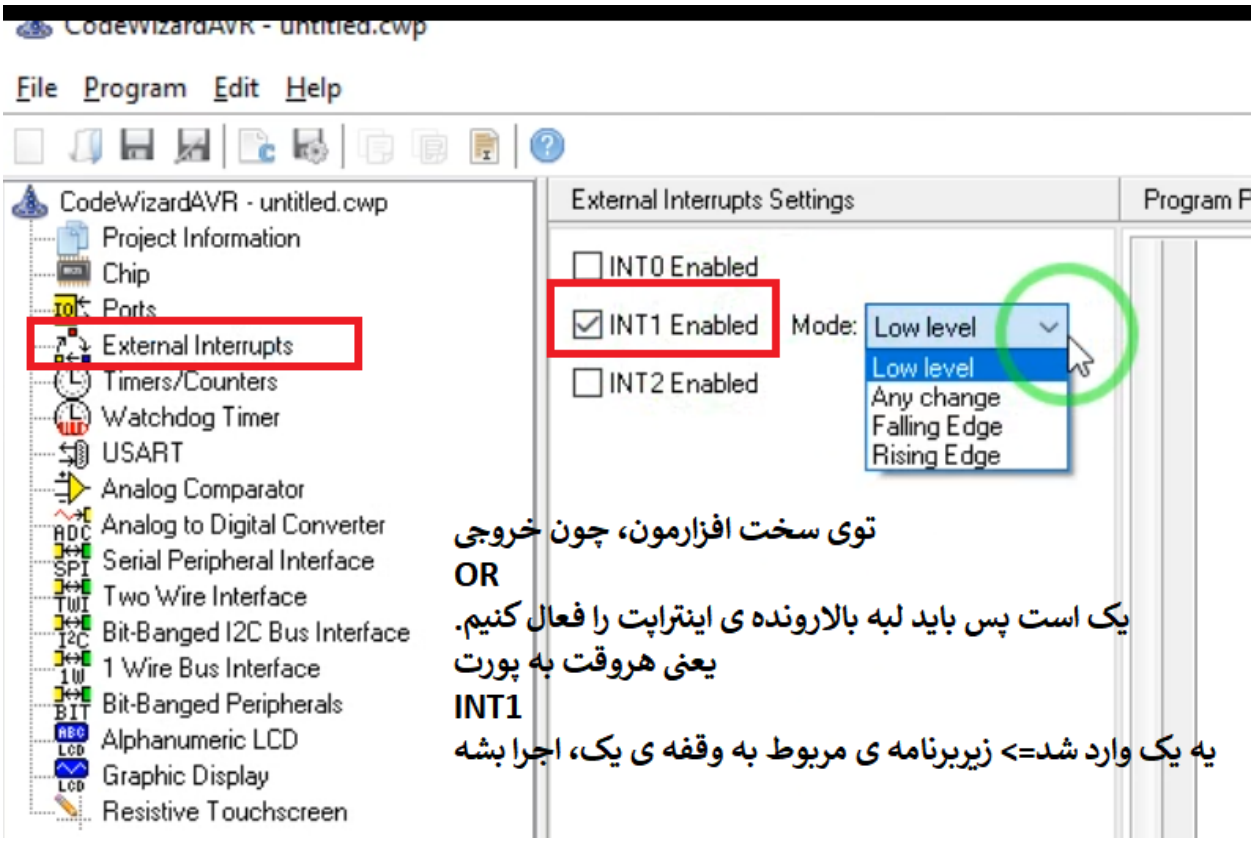


با این نوع نوشتن برنامه، میکروپروسسر همه ی وقتش را روی اسکن این صفحه کلید (keypad) میکنه و نمیتونه کار دیگه ای کنار این انجام بده => داره از روش pulling استفاده میکنه و منتظر است که کاربر یه پوش باتن را از صفحه فشار بده.

یه راه بهتر هست که بشه بهتر از قابلیت های میکرو استفاده کرد و اون استفاده از وقفه هاست.



فعال کردن وقفه های خارجی



توی سخت افزارمون، چون خروجی
OR
یک است پس باید لبه بالارونده ی اینتراپت را فعال کنیم.
یعنی هروقت به پورت
INT1
یه یک وارد شد=> زیربرنامه ی مربوط به وقفه ی یک، اجرا بشه

تنظیمات مربوط به LCD را هم انجام میدیم و بعدش فایل را ذخیره میکنیم.

زیربرنامه ی مربوط به interrupt1

```
// External Interrupt 1 service routine
interrupt [EXT_INT1] void ext_int1_isr(void)
{
// Place your code here

}
```

مقداردهی وقفه های خارجی

```
// External Interrupt(s) initialization
// INT0: Off
// INT1: On
// INT1 Mode: Rising Edge
// INT2: Off
GICR|=(1<<INT1) | (0<<INT0) | (0<<INT2);
MCUCR=(1<<ISC11) | (1<<ISC10) | (0<<ISC01) | (0<<ISC00);
MCUCSR=(0<<ISC2);
GIFR=(1<<INTF1) | (0<<INTF0) | (0<<INTF2);
```

فعال شدن وقفه ی جهانی

```
// Global enable interrupts
#asm("sei")
```

برای راه اندازی این وقفه‌ها ۴ رجیستر در فضای SFR تعبیه شده است که عبارت‌اند از: GICR , GIFR , MCUCR و MCUCSR.

نکته: رجیسترهای نام برده دارای بیت‌های غیر مرتبط با وقفه خارجی نیز هستند که ما در این آموزش با آنها کاری نداریم.

رجیستر GICR (General Interrupt Control Register)

برای فعال‌سازی هرکدام از وقفه‌های ۰ ، ۱ یا ۲ باید از این رجیستر استفاده کرد.

7	6	5	4	3	2	1	0	
INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
R/W	R/W	R/W	R	R	R	R/W	R/W	

در انتهای این رجیستر سه بیت INT۰ , INT۱ و INT۲ قرار دارند که با یک کردن هرکدام از آنها وقفه مربوط فعال می‌شود. البته باید دقت کرد که بیت وقفه عمومی قبل تر راجب آن توضیح دادیم هم فعال باشد.

رجیستر GIFR (General Interrupt Flag Register)

هرگاه وقفه خارجی رخ دهد، بیت متناظر با آن وقفه یک شده و CPU از این طریق متوجه رخ دادن وقفه می‌شود.

7	6	5	4	3	2	1	0	
INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
R/W	R/W	R/W	R	R	R	R	R	

همانطور که در شکل بالا می‌بینید، سه بیت INTF۰ , INTF۱ و INTF۲ که به آنها بیت‌های Flag یا پرچم نیز گفته می‌شود، با رخ دادن وقفه ۱ شده و CPU را مطلع می‌سازند. سپس CPU عملیات تعیین شده را انجام داده و در پایان به صورت اتوماتیک، همان بیت را ۰ می‌کند.

رجیستر MCUCR (MCU Control Register)

این رجیستر نحوه چگونگی رخ دادن وقفه را معلوم می‌کند. مثلا وقفه با لبه بالارونده باشد یا لبه پایین رونده و ... .

7	6	5	4	3	2	1	0	
SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

7	6	5	4	3	2	1	0	
SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

۴ بیت کم ارزش این رجیستر متعلق به وقفه‌های خارجی ۰ و ۱ هستند. برای تنظیم این ۴ بیت به شکل زیر توجه کنید.

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

دو بیت اول مربوط به وقفه شماره ۰ و دو بیت دوم مربوط به وقفه شماره ۱ است. هرکدام چهار حالت می‌توانند داشته باشند:

۱. اگر ۰۰ مقداردهی شوند، سطح ۰ منطقی باعث ایجاد وقفه می‌شود.

۲. اگر ۰۱ مقداردهی شوند، هر دو لبه‌ی پایین رونده و بالارونده باعث ایجاد وقفه می‌شود.

۳. اگر ۱۰ مقداردهی شوند، لبه پایین رونده وقفه ایجاد می‌کند.

۴. اگر ۱۱ مقداردهی شوند، لبه بالا رونده وقفه ایجاد می‌کند.

رجیستر MCUCSR (MCU Control and Status Register)

همانند رجیستر MCUCR است. با این تفاوت که نحوه رخ دادن وقفه خارجی ۲ را تنظیم می‌کند.

7	6	5	4	3	2	1	0	
JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

وقفه شماره ۲ تنها یک بیت تنظیم به نام ISC۲ دارد. اگر این بیت ۰ شود، وقفه بصورت پایین رونده و اگر ۱ شود وقفه بصورت بالارونده عمل می‌کند.

نکته: توصیه می‌شود رجیستر MCUCR و MCUCSR قبل از GICR و GIFR مقداردهی شود.

```
GICR|=(1<<INT1) | (1<<INT0) | (1<<INT2);

MCUCR=(1<<ISC11) | (1<<ISC10) | (1<<ISC01) | (1<<ISC00);

MCUCSR=(1<<ISC2);

GIFR=(1<<INTF1) | (1<<INTF0) | (1<<INTF2);
```

مثلا اگه وقفه ها اینطوری مقداردهی شوند:
رجیستر GICR (General Interrupt Control Register)
برای فعال کردن وقفه های خارجی است => با یک کردن هرکدام، وقفه ی موردنظر فعال میشود => در این مثال همه ی وقفه ها فعال میشوند.

رجیستر MCUCR

نحوه ی چگونگی رخ دادن وقفه برای وقفه های 0,1
ISC00,ISC01 برای وقفه ی صفر به کار میره که اگه هردو بیت یک باشه => وقفه ی صفر، با لبه ی بالارونده ی کلاک (بیت ۱)، فعال میشود.

بیت های ISC10,ISC11 برای وقفه ی یک به کار میروند => چون هردو یک هستند یعنی وقفه ی صفر، با لبه ی بالارونده ی کلاک(بیت ۱)، فعال میشود.

رجیستر MCUCSR
نحوه رخ دادن وقفه خارجی 2 را تنظیم می کند.
وقفه شماره 2 تنها یک بیت تنظیم به نام ISC2 دارد. اگر این بیت 0 شود، وقفه بصورت پایین رونده و اگر 1 شود وقفه بصورت بالارونده عمل می کند.

رجیستر GIFR (General Interrupt Flag Register)
هرگاه وقفه خارجی رخ دهد، بیت متناظر با آن وقفه یک شده و CPU از این طریق متوجه رخ دادن وقفه می‌شود.