وزارت علوم ، تحقیقات و فناوری
دانشگاه جهرم
دانشکده فنی و مهندسی

آزمایشگاه ریزپردازنده

(با رویکرد برنامه‌نویسی به زبان C و استفاده از نرم‌افزار CodeVisionAVR)

گردآورنده :

محمّدعلی شفیعیان

فهرست

آزمایشگاه ریزپردازنده

آزمایش شماره ۱

آشنایی با محیط برنامه‌نویسی، نرم‌افزار شبیه‌ساز و مجموعه آموزشی میکروکنترلر *AVR*

۱-۱ آشنایی با محیط برنامه‌نویسی *CodeVisionAVR*

نرم‌افزار *CodeVisionAVR*، یک نرم افزار تخصصی برای رشته‌های برق و کامپیوتر می‌باشد. در واقع این نرم‌افزار یک کامپایلر برای زبان برنامه نویسی *C* می‌باشد که برای برنامه نویسی میکروکنترلرهای *AVR* از آن استفاده می‌شود. این برنامه، محیط برنامه‌نویسی و کامپایل کردن برنامه نوشته شده برای برنامه‌ریزی میکروکنترلر را فراهم می‌کند.

بسیاری از افراد حتی کسانی که رشته کامپیوتر می باشند با این نرم افزار به‌خوبی آشنا می‌باشند. آخرین نسخه این برنامه قدرت بسیار بیشتری پیدا کرده است و از طرفی مشکلات قبلی آن برطرف شده است. این برنامه در تمامی نسخه های ویندوز قابل نصب است.

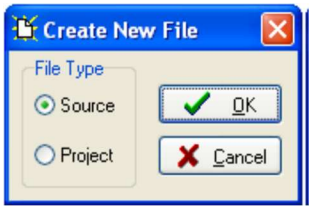
۱-۱-۱ ایجاد یک پروژه جدید در *CodeVisionAVR*

در نرم‌افزار *Codevision* به دو طریق می‌توان برنامه دلخواه را ایجاد کرد که در ادامه به بررسی هر دو روش پرداخته خواهد شد:

۱- با ایجاد یک صفحه پروژه و نوشتن کل برنامه

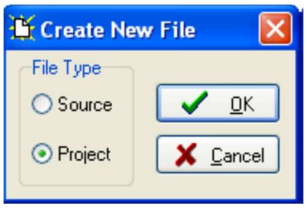
۲- با استفاده از *Codewizard*

در روش اول، ابتدا برنامه *CodeVisionAVR* را اجرا کرده سپس از منوی *File* گزینه *New* را کلیک می‌کنیم. در پنجره ظاهرشده گزینه *Source* را انتخاب کرده و صفحه ایجاد شده را *Save* می‌کنیم.



شکل ۱ : ایجاد یک *source* جدید

دوباره با کلیک بر روی گزینه *New* یک *Project* ایجاد می‌کنیم.

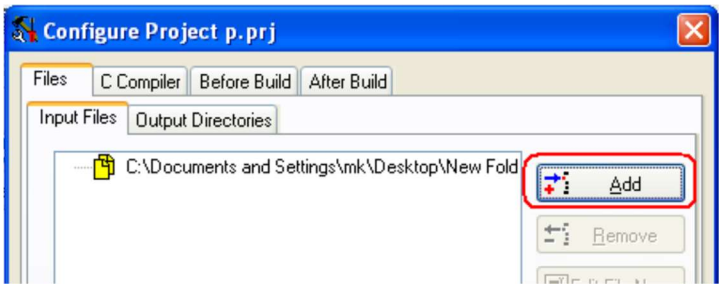


شکل ۲: ایجاد یک *project* جدید

در روش اول، چون نمی خواهیم از *Codewizard* استفاده کنیم، در پنجره *Confirm* که در شکل ۳ نشان‌داده شده است، گزینه *No* را انتخاب می‌کنیم.

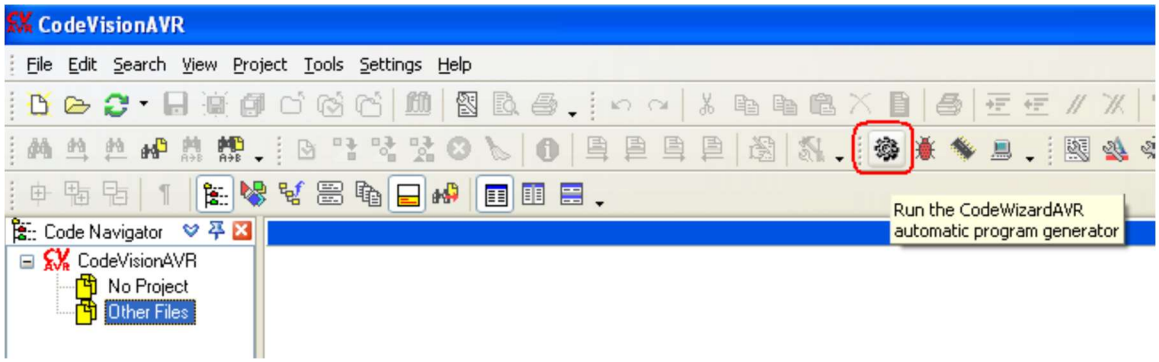
شکل ۱-۳: پنجره *Confirm*

پس از *Save* این فایل در همان مسیر، در پنجره باز شده بر روی گزینه *Add* کلیک کرده و فایل *Source* ایجاد شده با پسوند *C* را به پروژه اضافه می کنیم. با استفاده از این پنجره *IC* مورد نظر، فرکانس کاری، امکانات جانبی و غیره را تعیین می‌کنیم.



شکل ۴: اضافه کردن فایل *source* به فایل پروژه

در روش دوم که بهترین روش می‌باشد، پس از باز کردن نرم‌افزار، آیکن *CodeWizard* را کلیک می‌کنیم.

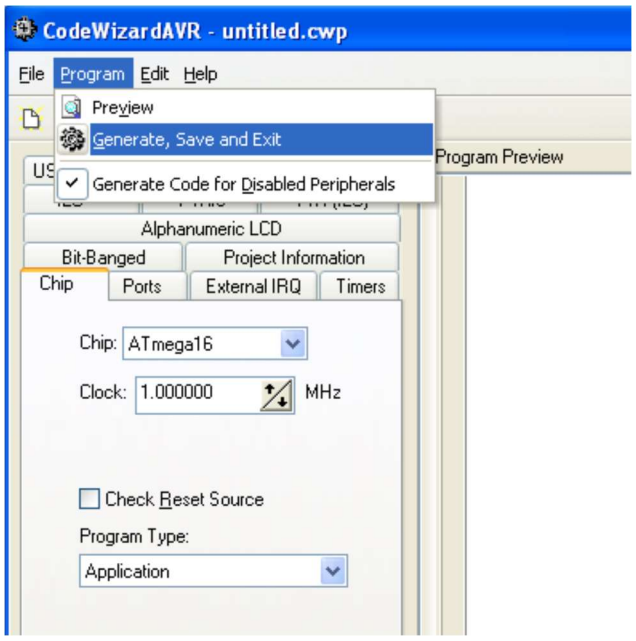


شکل ۵: استفاده از *CodeWizard*

در پنجره باز شده چون قصد استفاده از آی سی *Xmega* را نداریم گزینه اول را انتخاب می‌کنیم.

شکل ۶ انتخاب نوع *IC*

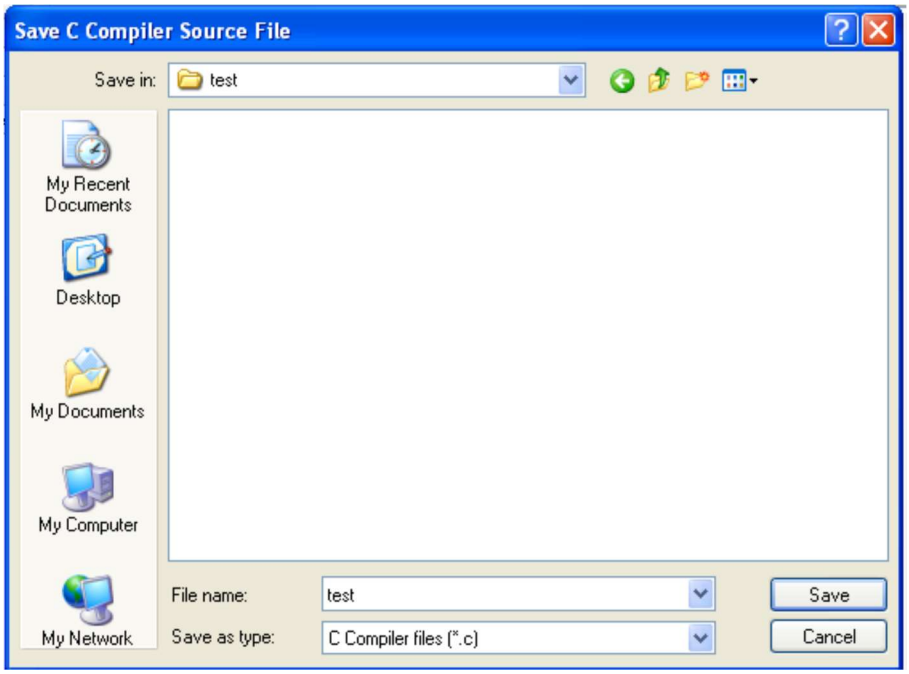
با انتخاب هر یک از آی‌سی‌های *AVR* امکانات آن به سربرگ‌ها افزوده می‌شود و برای فعال کردن هریک از امکانات از قبیل *LCD* کاراکتری، پورت سریال و ... کافی است به قسمت مربوطه برویم و تیک فعال‌سازی آن را اتخاب کنیم. همچنین یکی از روش های آشنایی با امکانات آی‌سی‌های *AVR* ، استفاده از *CodeWizard* است. پس از انتخاب امکانات مورد نظر و انجام تنظیمات مربوطه برای ساخت فایل پروژه، از منوی *Program* گزینه *Generate, Save and Exit* را کلیک می‌کنیم.



شکل ۶: ذخیره تنظیمات ایجادشده

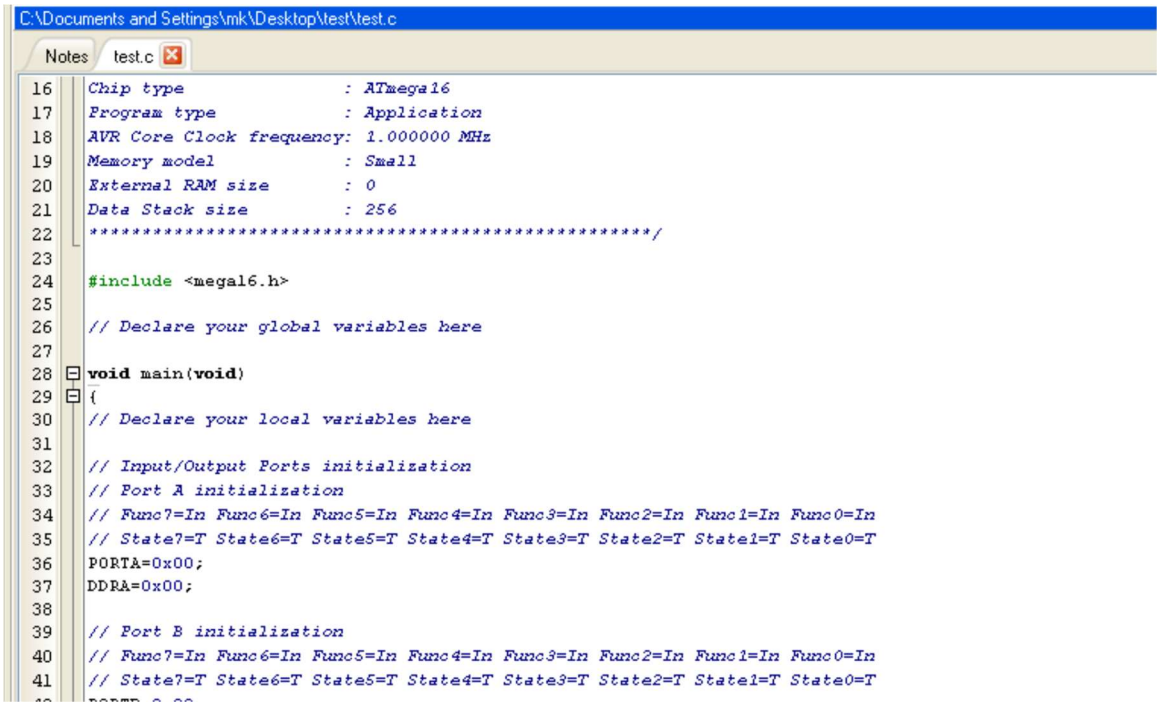
نرم افزار سه فایل برای پروژه، فایل *C* و فایل تنظیمات *Codewizard* ایجاد می کند و ما باید سه بار نام دلخواه را در آن وارد کنیم که بهتر است نام سه فایل مانند هم باشد.

** **نکته:** لازم است کلیه فایل‌هایی که می‌خواهیم ایجاد کنیم در یک پوشه مخصوص باشد تا از پراکندگی و اشتباه در جابجایی پروژه جلوگیری شود.



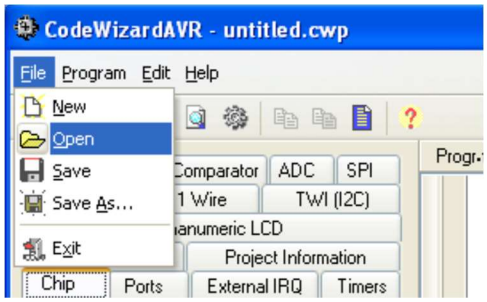
شکل ۷: ذخیره همه فایل‌ها در یک پوشه

پس از *Save* کردن مشاهده می‌کنید که تمام رجیسترها، کتابخانه‌های مربوطه، وقفه ها، تابع اصلی و حتی حلقه `while(1)` نوشته شده و شما فقط باید بدنه اصلی برنامه را به آن اضافه کنید.



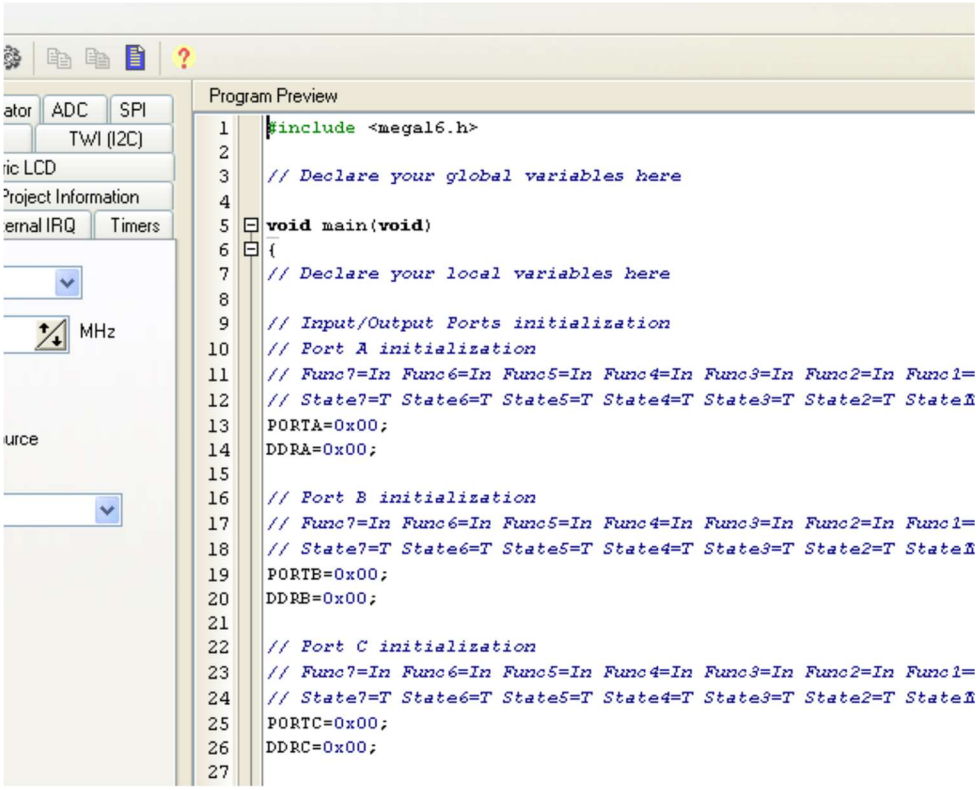
شکل ۸: محیط برنامه‌نویسی

اگر بخواهیم در خلال برنامه تغییراتی در رجیسترها اعمال کنیم و برای این کار به *Codewizard* نیاز داشته باشیم به این صورت عمل می‌کنیم که پنجره *Codewizard* را باز کرده و در آن از منو *File* گزینه *Open* را کلیک می‌کنیم.



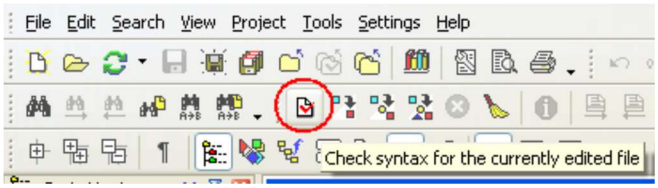
شکل ۹: مراجعه دوباره به *CodeWizard*

فایل *Codewizard* ایجاد شده برای برنامه با پسوند *cwp* را باز می‌کنیم. در این حالت مشاهده می‌کنید که تمام تنظیمات انجام شده *Load* می‌شود. پس از تغییر تنظیمات از منو فایل گزینه *Save* را زده و سپس از منوی *Program* گزینه *Program preview* را انتخاب می‌کنیم. برنامه جدید طبق تنظیمات انجام شده در سمت راست صفحه ظاهر می‌شود. در اینجا رجیسترهایی را که تغییر کرده‌اند را کپی کرده و به جای مقادیر قبلی *Paste* می‌کنیم.



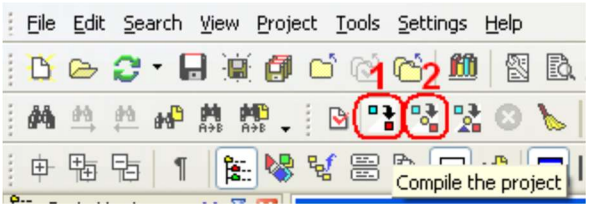
شکل ۱۰: مشاهده *Program Preview*

حین نوشتن برنامه اگر بخواهیم از وجود خطاها آگاه شویم گزینه *Check syntax...* را کلیلک می‌کنیم. این گزینه مواردی را که مقایر با شکل انشایی دستورات است، به ما گزارش می‌دهد.



شکل ۱۱: استفاده از *Check syntax*

پس از نوشتن برنامه، گزینه *Compile the project* سپس *Build the project* را کلیک می‌کنیم.



شکل ۱۲: کامپایل کردن برنامه نوشته‌شده

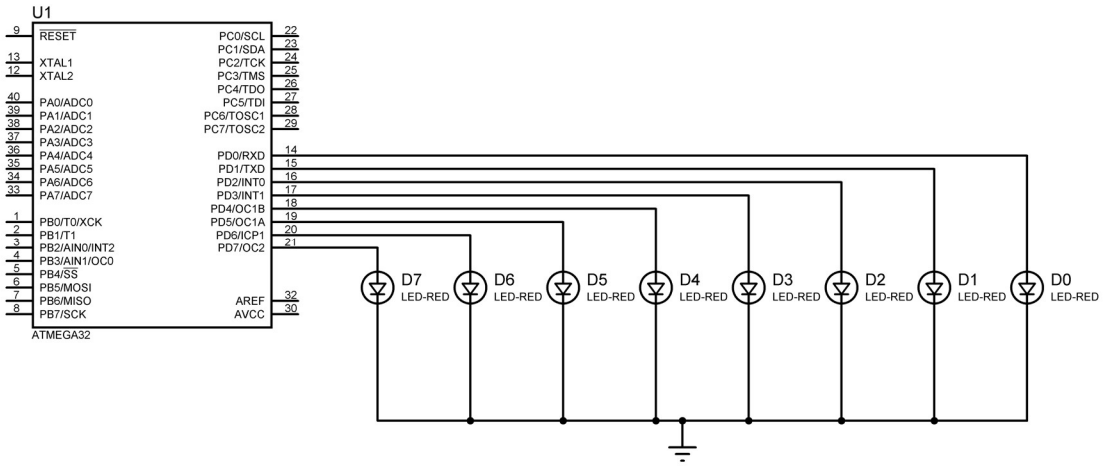
آزمایش ۱-۱: فلاشر ساده

برنامه زیر را در محیط نرم‌افزار *CodeVisionAVR* نوشته و پس از بررسی درستی برنامه نوشته شده، آن را کامپایل کرده و در محیط نرم‌افزار *ISIS Proteus* مدار نشان داده شده در شکل ۱۳ را شبیه‌سازی نمایید. سپس به‌کمک *Programmer* برنامه خود را به *IC* میکروکنترلر *AVR* منتقل کرده و با استفاده از مجموعه آموزشی موجود در آزمایشگاه، مدار نشان داده شده در شکل ۱۳ را پیاده‌سازی کنید.

```
#include <mega32.h>
#include <delay.h>
#define xtal 8000000

int i;
void main (void)
{
    DDRD = 0xFF;
    while(1)
    {
        for(i = 1; i <= 128; i = i*2)
        {
            PORTD = i;
            delay_ms(100);
        }

        for(i = 64; i > 1; i = i/2)
        {
            PORTD = i;
            delay_ms(100);
        }
    }
}
```



شکل ۱۳: شمای مداری آزمایش ۱-۱

آزمایش ۲-۲

برنامه‌ای بنویسید که در مدار شکل ۱۳، *LED*ها با الگوی نشان داده شده در جدول ۱ روشن و خاموش شوند. مدت زمان تأخیر بین هر دو لحظه متوالی را ۱۰۰ میلی ثانیه در نظر بگیرید. برنامه را به گونه‌ای بنویسید که این عمل تا بینهایت تکرار شود. برنامه خود را با استفاده از یکی از دستورالعمل‌های حلقه که فکر می کنید مناسب باشد، بنویسید.

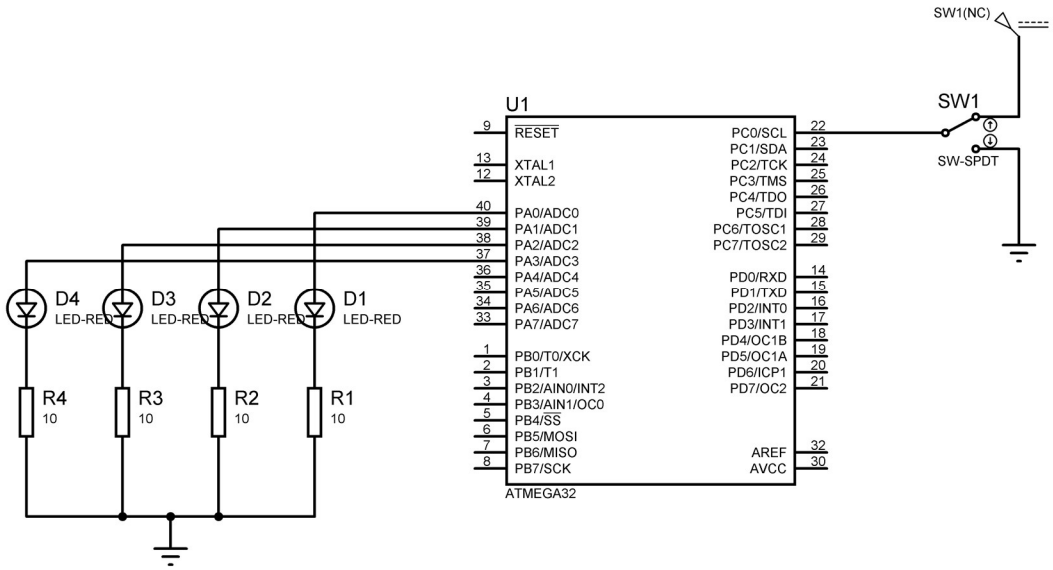
جدول ۱: الگوی روشن و خاموش شدن *LED*ها

	D1	*D2*	*D3*	*D4*	*D5*	*D6*	*D7*	*D8*
لحظه ۱	***ON***	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	***ON***
لحظه ۲	*OFF*	***ON***	*OFF*	*OFF*	*OFF*	*OFF*	***ON***	*OFF*
لحظه ۳	*OFF*	*OFF*	***ON***	*OFF*	*OFF*	***ON***	*OFF*	*OFF*
لحظه ۴	*OFF*	*OFF*	*OFF*	***ON***	***ON***	*OFF*	*OFF*	*OFF*
لحظه ۵	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*
لحظه ۶	*OFF*	*OFF*	*OFF*	***ON***	***ON***	*OFF*	*OFF*	*OFF*
لحظه ۷	*OFF*	*OFF*	***ON***	*OFF*	*OFF*	***ON***	*OFF*	*OFF*
لحظه ۸	*OFF*	***ON***	*OFF*	*OFF*	*OFF*	*OFF*	***ON***	*OFF*
لحظه ۹	***ON***	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	*OFF*	***ON***

پرسش:

۱ـ برنامه‌ای بنویسید که در مدار شکل ۱۵، اگر کلید متصل به *PortC.0* در وضعیت یک منطقی (متصل به ۵ ولت باشد)، آنگاه معادل باینری اعداد ۰ تا ۱۵ به صورت شمارش از صفر تا ۱۵ بر روی *LED*های متصل به *PORT A* ارسال شود و اگر در وضعیت صفر منطقی (متصل به زمین) باشد، عملی انجام نشود. این برنامه را با

استفاده از حلقه‌ها بنویسید و آن را به‌گونه‌ای بنویسید که عمل شمارش تا بینهایت تکرار شود. همچنین زمان تأخیر بین هر دو شمارش متوالی را ۱۰۰ میلی‌ثانیه در نظر بگیرید.



شکل ۱۵: شمای مداری مربوط به پرسش ۱

آزمایشگاه ریزپردازنده

آزمایش شماره ۲

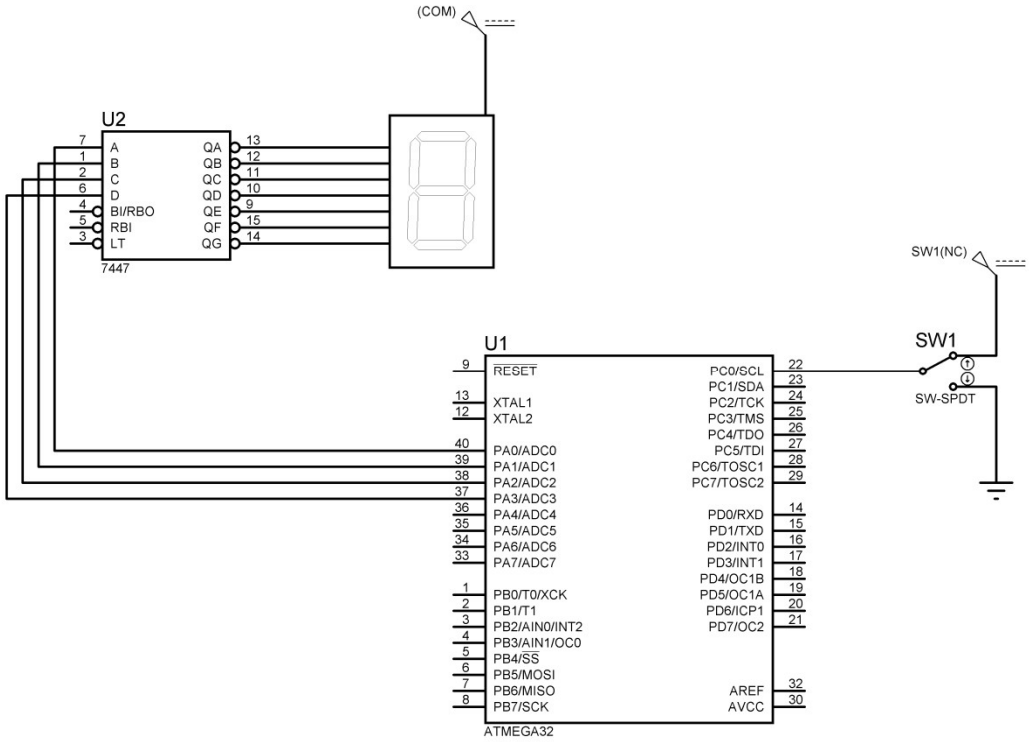
نحوه استفاده از *7-Segment* و ارتباط آن با میکروکنترلر

آزمایش ۲–۱

برنامه ای بنویسید که وضعیت پایه صفر از *Port C* را بررسی کند و در صورت یک بودن آن، شمارش را صفر تا ۹ و در صورت صفر بودن، شمارش را از ۹ تا صفر انجام دهد. سپس حاصل را بر روی یک *7-Segment* که توسط یک *IC* دیکدر ۷۴۴۷ به *Port A* متصل شده است، نمایش دهد.

توجه : برای شبیه سازی از یک *7-Segmant* آند مشترک استفاده کنید.

توجه : *IC* دیکدر ۷۴۴۷ یک دیکدر *BCD* به *7-Segment* است. در این *IC*، عدد *BCD* مورد نظر به ورودی های *A*، *B*، *C* و *D* اعمال می شود که در آن *A* بیت کم ارزش است و خروجی متناظر با آن در پایه های *a* تا *g* قرار می گیرد. در عمل، پایه های خروجی توسط مقاومت های ۱۵۰ Ω به *7-Segment* متصل می شوند. شمای مداری این بخش از آزمایش در شکل ۱ نشان داده شده است.



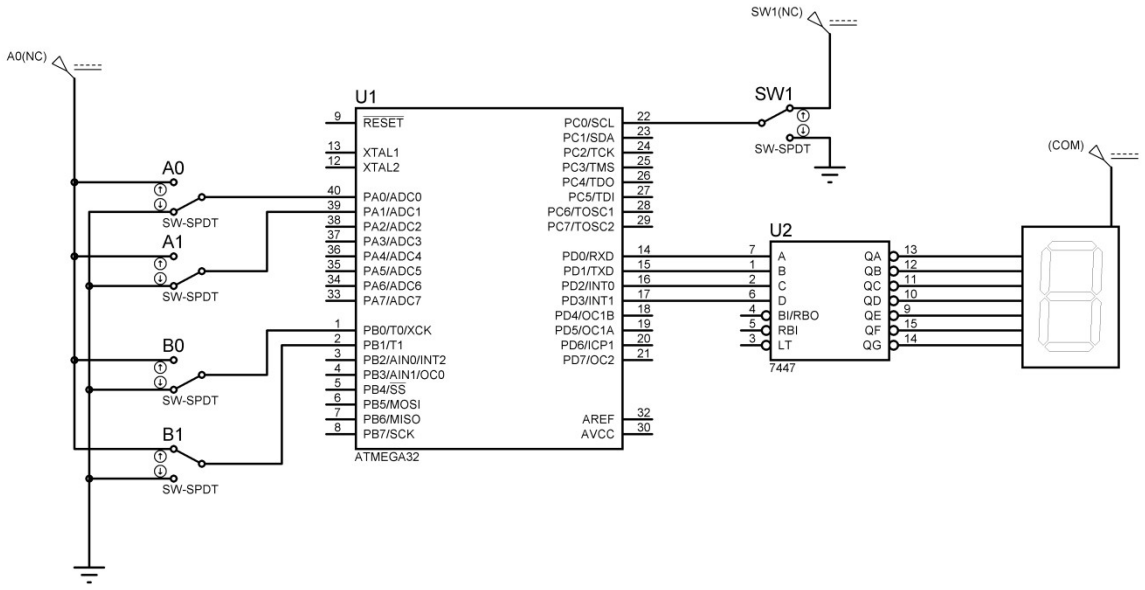
شکل ۱ : شمای مداری آزمایش ۲–۱

آزمایش ۲–۲

برنامه ای بنویسید که داده ورودی به پایه های صفر و یک از *Port A* و داده ورودی به پایه های صفر و یک از *Port B* را برحسب وضعیت پایه صفر از *Port C* با یکدیگر جمع یا در هم ضرب نماید؛ به این صورت که اگر پایه صفر از *Port C* یک باشد دو عدد را با هم جمع و اگر صفر باشد دو عدد را در هم ضرب کند. بدیهی است که دو عدد ورودی دو بیتی بوده و بیت کم ارزش آنها به پایه های صفر از *Port B* و *Port C* متصل است. سپس نتیجه عمل جمع یا ضرب بر روی یک *7-Segment* که توسط یک *IC* دیکدر ۷۴۴۷ به *Port D* متصل شده است، نمایش داده شود.

توجه : برنامه را به گونه ای بنویسید که اگر در هر لحظه از اجرای شبیه سازی وضعیت *Port C.0* یا اعداد ورودی به *Port A* و *Port B* تغییر کند، متناسب با آن، خروجی نمایش داده شده نیز تغییر کند.

شمای مداری این بخش از آزمایش در شکل ۲ نشان داده شده است.



شکل ۲ : شمای مداری آزمایش ۲–۲

پرسش :

۱ـ با توجه به آزمایش ۲–۲ برنامه ای بنویسید که با توجه به وضعیت *Port C.0* دو عدد ۴ بیتی ورودی به *Port A* و *Port B* را با هم جمع یا در هم ضرب نماید و نتیجه را (که ممکن است عددی دو رقمی باشد) بر روی دو عدد *7-Segment* متصل به *Port D* نمایش دهد. همچنین اگر حاصل ضرب دو عدد بزرگتر از ۹۹ بود، بر روی دو *7-Segment* عدد یا حرفی نمایش داده نشود.

آزمایشگاه ریزپردازنده

آزمایش شماره ۳

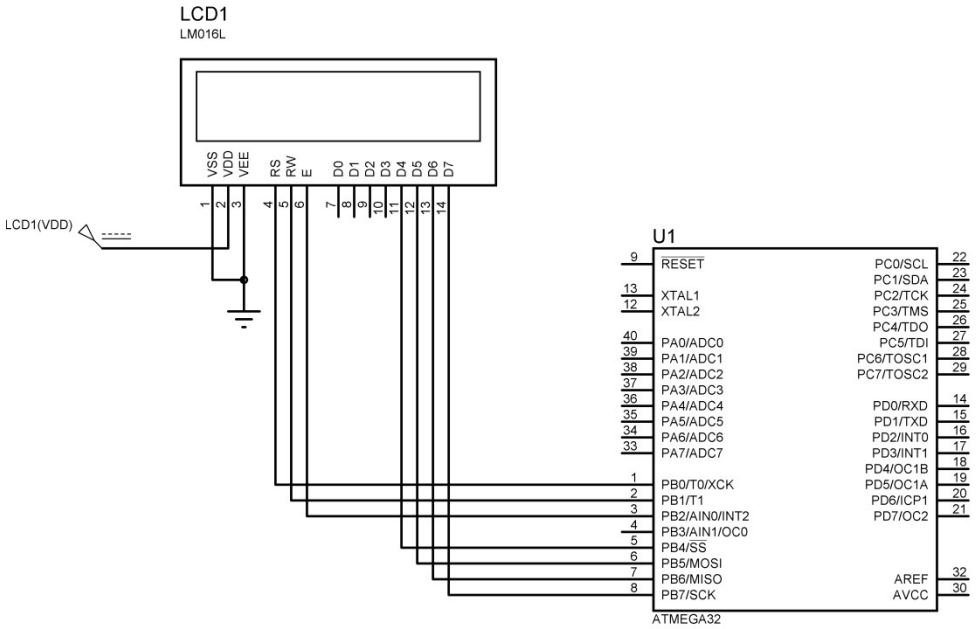
اتصال *LCD* به میکروکنترلرهای *AVR*

در سال های اخیر، برای نمایش اطلاعات میکروکنترلرها بیشتر از *LCD* استفاده می شود. دلیل آن، بهای پایین و امکان نمایش حروف، رقم و کاراکترهای گرافیکی است.

اطلاعات بر روی *LCD* ممکن است در یک، دو یا چهار سطر نوشته شود. این اطلاعات را می توان به صورت ۸ بیتی یا ۴ بیتی برای *LCD* ارسال نمود و با فرمان هایی که برای آن پیش بینی شده است، در هر لحظه می توان آن را پاک نمود، محل نمایش اطلاعات را تغییر داد، حروف را از طرف راست به چپ یا برعکس نوشت و بالأخره حروف یا مکان نما را روشن یا خاموش نمود.

آزمایش ۳-۱

در این آزمایش می خواهیم با استفاده از میکروکنترلر *ATmega32* برنامه ای بنویسیم که توسط آن، پیغام *Microcontroller Course* بر روی یک *LCD* نمایش داده شود. شمای مداری این آزمایش در شکل ۱ نشان داده شده است.



شکل ۱ : شمای مداری آزمایش ۳-۱

برای یادآوری و آشنایی بیشتر با دستورات *LCD*، برنامه این آزمایش در ادامه آورده شده است :

```
#include <mega32.h>
#include <stdio.h>
#include <delay.h>

#asm
.equ __lcd_port=0x18;PORTB
#endasm
#include <lcd.h>

void main (void){
    PORTB = 0x00;
    DDRB = 0x00;

    lcd_init(16);
    lcd_clear();
    lcd_gotoxy(0,0);
    lcd_putsf("microcontroller”);
    lcd_gotoxy(5,1);
    lcd_putsf("course”);
}
```

این برنامه را به دقت بررسی نموده و آزمایش را انجام دهید.

آزمایش ۳–۲

در همان مدار نشان داده شده در شکل ۱ برنامه ای بنویسید که رشته *Microcontroller Course* را به طور متناوب و به صورت حرف به حرف و با تأخیر زمانی اندکی بین نمایش حروف، بر روی *LCD* نمایش داده دهد.

آزمایش ۳–۳

برنامه ای بنویسید که در هر لحظه، داده متصل به *PortD* را بخواند و بر روی *LCD* نمایش دهد.

پرسش :

۱ـ برنامه ای بنویسید که یک رشته (مانند نام و نام خانوادگی خودتان) را به صورت متحرک بر روی *LCD* نمایش دهد به این صورت که رشته مذکور از سمت راست *LCD* وارد شود و از راست به چپ *LCD* را طی کند و از سمت چپ خارج شود.

آزمایشگاه ریزپردازنده

آزمایش شماره ۴

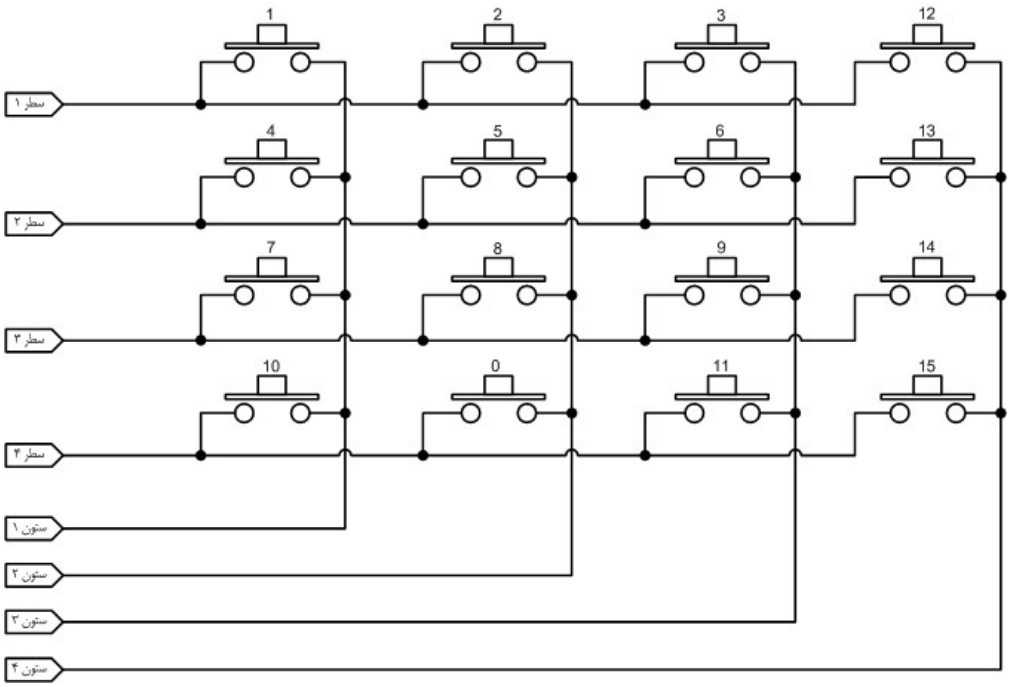
اتصال صفحه کلید به میکروکنترلرهای *AVR*

مقدمه

یکی از وسایلی که برای وارد کردن اطلاعات در میکروکنترلرها به کار می رود، صفحه کلید می باشد. ساختار صفحه کلید به صورت ماتریسی از سطر و ستون می باشد که با فشار هر کلید، یک سطر به یک ستون متصل می شود و در غیر این صورت، اتصالی بین سطر و ستون وجود ندارد.

برای خواندن از صفحه کلید توسط میکروکنترلر، معمولاً از روش اسکن صفحه کلید برای تشخیص کلید فشرده شده استفاده می شود. همانگونه که در شکل ۱ نشان داده شده است می توان ستون ها را با یک مقاومت به سطح ولتاژ *Vcc* متصل نمود تا میکروکنترلر هنگامی که کلیدی فشرده نشده است، سطح منطقی *High* را بخواند. سطرها با توجه به الگوی چرخشی (... ←1110←1101←1011← 0111 ...) اسکن می شوند. زمانی میکرو کنترلر می تواند در پایه یکی از ستون ها صفری تشخیص دهد که کلید فشرده شده، آن ستون را به سطر صفر شده وصل کند و با توجه به شماره سطر و ستون صفر شده، می توان کلید فشرده شده را تشخیص داد.

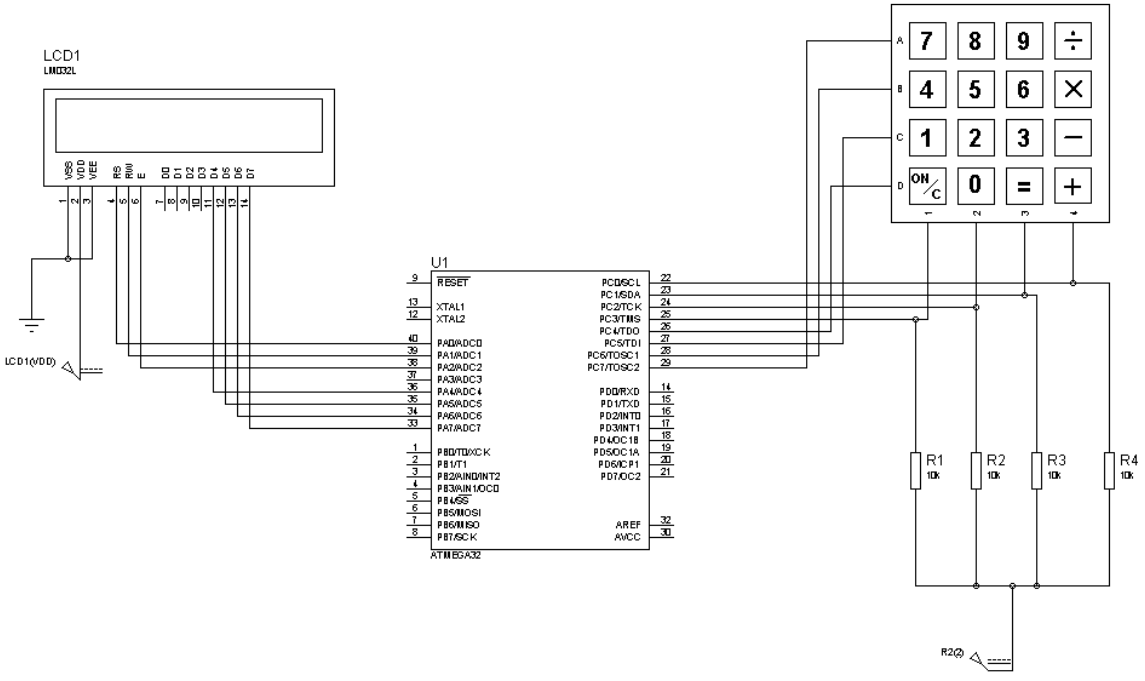
برای روشن شدن روش بالا، فرض کنید سطرهای یک صفحه کلید ۴×۴ به چهار بیت کم ارزش *PortC* و ستون های آن به چهار بیت پر ارزش *PortC* متصل شده اند. همچنین ستون ها توسط مقاومت هایی به *Vcc* وصل شده اند. حال اگر سطرها را صفر کنیم و هر یک از کلیدهای یک سطر را فشار دهیم، اتصال بین یک سطر و ستون برقرار می شود و ستون مربوطه نیز صفر می شود. به عنوان مثال، در شکل ۱ اگر سطر یک را برابر صفر کرده باشیم، در این صورت اگر هر یک از کلیدهای ۱، ۲، ۳ یا ۱۲ را فشار دهیم، یکی از ستون های ۱ تا ۴ نیز مساوی صفر می شود؛ بنابراین اگر سطرها یا چهار بیت پر ارزش *PortC* یعنی *PC4* تا *PC7* را صفر کنیم، لذا با فشار یک کلید، ستون ۱ برابر با صفر و بقیه ستون ها مساوی ۱ می شوند یعنی مقدار ستون های *PC0* تا *PC3* برابر با *0111* می شود به این ترتیب کد کلیدهای سطر یک برابر با *00000111* باینری یا *0x07* در مبنای شانزده می باشد. حال با توجه به ستون صفر شده می توان تشخیص داد که کدام یک از کلیدهای سطر یک فشار داده شده است.



شکل ۱ : صفحه کلید ۴×۴

آزمایش ۴-۱

برنامه ای بنویسید که کلید فشرده شده در یک صفحه کلید ۴×۴ متصل به *PortC* را بر روی یک *LCD* متصل به *PortA* نمایش دهد. شمای مداری این آزمایش در شکل ۲ نشان داده شده است.



شکل ۲ : شمای مداری آزمایش ۴-۱

برنامه ای که می توان برای اسکن صفحه کلید به کار برد، در ادامه آورده شده است :

```
#include <mega32.h>
#include <stdio.h>
#include <delay.h>

unsigned char scan_key(void);
unsigned char code[4][4]={ {7,4,1,10},{8,5,2,0},{9,6,3,11},{12,13,14,15} };
.
.
.
.
.
.
.
.
.
unsigned char scan_key(void)
{
        unsigned char i,data,num_key,temp;
        num_key=0xff;
        temp=0x70;
        for(i=0;i<4;i++){
                PORTC=temp;
                delay_ms(5);
                data=PINC & 0x0f;
                if(data==0x07)
                        num_key=code[0][i];
                if(data==0x0B)
                        num_key=code[1][i];
                if(data==0x0D)
                        num_key=code[2][i];
                if(data==0x0E)
                        num_key=code[3][i];
                temp= ((temp>>=1)  | 0x80) & 0xF0 ;

        }
        return num_key;
}
```

این برنامه را به دقت بررسی نموده و از آن به عنوان یک تابع در برنامه خود استفاده کنید به این صورت که مقدار بازگردانده شده توسط تابع مذکور را به عنوان شماره کلید فشرده شده به کار ببرید.

پرسش:

۱ـ همین برنامه را برای یک صفحه کلید ۳×۴ بنویسید.

۲ـ با استفاده از صفحه کلید ۴×۴ برنامه ای بنویسید که دو عدد را گرفته و با توجه به کلید فشرده شده در ستون چهارم (یعنی کلیدهای تقسیم، ضرب، تفریق و جمع) عمل متناظر با هر کلید را انجام داده و با فشردن کلید = نتیجه محاسبه را بر روی *LCD* نمایش دهد. همچنین اگر کلید *ON/C* فشار داده شود، صفحه نمایش *LCD* پاک شود.

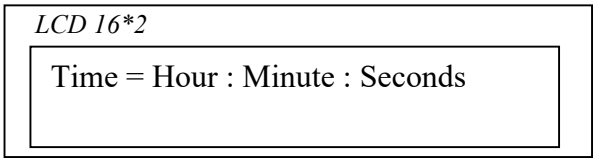
آزمایشگاه ریزپردازنده

آزمایش شماره ۵

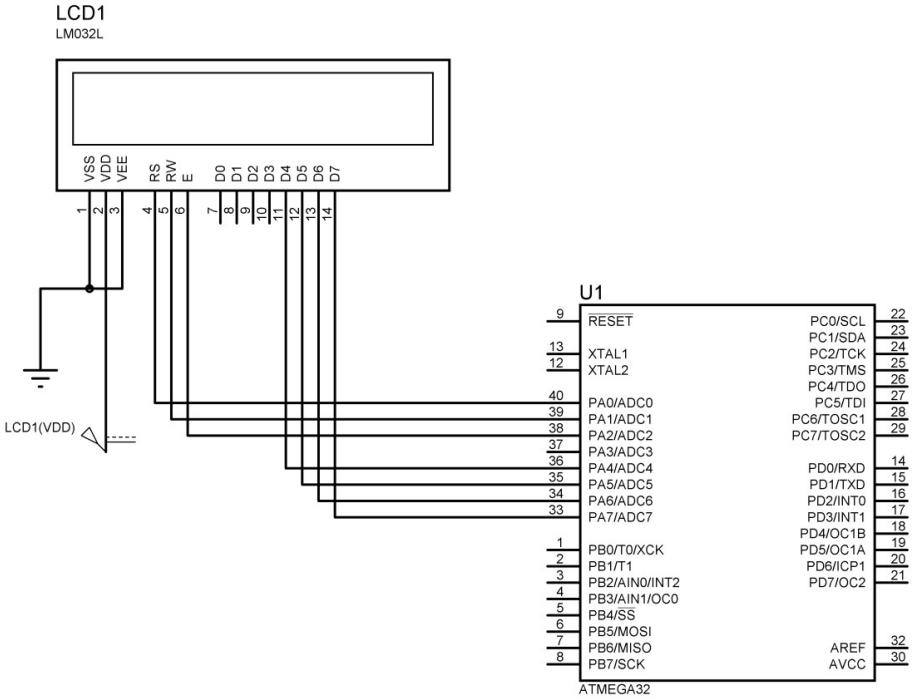
ساعت دیجیتال به کمک میکروکنترلر *AVR*

آزمایش ۵–۱

برنامه ای بنویسید که زمان به صورت ساعت، دقیقه و ثانیه بر روی یک *LCD* متصل به *PortA* نمایش داده شود و با شمارش و افزایش یک واحدی ثانیه شمار، دقیقه و ساعت نیز *update* شوند. برنامه را به گونه ای بنویسید که زمان به صورت زیر بر روی *LCD* نمایش داده شود :



برای ایجاد تأخیر، از دستور *delay_ms()* استفاده کنید. شمای مداری این آزمایش در شکل ۱ نشان داده شده است.



شکل ۱ : شمای مداری آزمایش ۵–۱

آزمایش ۵–۲

برنامه آزمایش ۵–۱ را به گونه ای بنویسید که در سطر دوم LCD شماره روز و ماه نیز نمایش داده شود.

پرسش ها :

۱ـ برنامه ساعت دیجیتال را به گونه ای بنویسید که قبل از شروع به کار ساعت، تنظیم اولیه ساعت، دقیقه، ثانیه، روز و ماه توسط یک صفحه کلید انجام شود.

آزمایشگاه ریزپردازنده ها

آزمایش شماره ۶

آشنایی با تایمر و کانتر در میکروکنترلر *AVR*

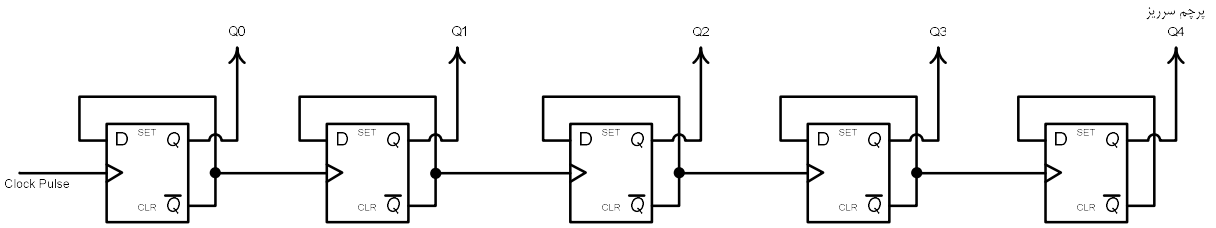
مقدمه

تایمر/کانتر یکی از بخش های مهم میکروکنترلرها می باشد. در بیشتر مواقع لازم که تعدادی وقایع خارجی (با سرعت بالا) شمارش شود و یا گاهی لازم است که در یک زمان خاص و دقیق، کاری صورت گیرد. تنها توسط تایمر/کانتر ها می توان این کارهای دقیق و با سرعت بالا را انجام داد.

میکروکنترلرهای *AVR* حداکثر دارای شش عدد تایمر کانتر هشت بیتی و شانزده بیتی هستند. برخی از آنها دارای عملکرد ساده و برخی دیگر دارای امکانات بیشتر نظیر تولید موج *PWM* ، حالت مقایسه *CTC* ، حالت تسخیر، عملکرد غیر همزمان و ... می باشند.

۱ ـ تئوری عملکرد تایمر/کانتر

تایمر و کانترها از تعدادی فلیپ فلاپ که به صورت پشت سر هم قرار گرفته و به صورت یک شمارنده آسنکرون عمل می کنند، تشکیل شده اند (شکل ۱).



شکل ۱ : شمارنده آسنکرون

در این شمارنده پایه های *Q3-Q0* به عنوان خروجی استفاده می شود. با فرض وارد نمودن مقدار *0000* و با اعمال پالس ساعت، شمارنده شروع به شمارش می نماید و حداکثر تا مقدار *1111* بالا می رود. پس از این حالت و با اعمال پالس ساعت بعدی، بیت سرریز (*Q4*) فعال می شود که نشان دهنده این است که شمارنده به حداکثر مقدار خود رسیده است.

هنگامی که پالس ساعت اعمالی به این شمارنده یک مقدار مشخص داشته باشد، می توان با در نظر گرفتن مدت زمان شمارش از مقدار *xxxx* تا *1111* و سپس سرریز شدن شمارنده، زمان های معینی را ایجاد نمود. در این حالت شمارنده به صورت تایمر عمل می کند. به عنوان نمونه برای شکل ۱ اگر پالس ساعت اعمالی دارای فرکانس *Hz* ۲ باشد و شمارنده نیز با مقدار پیش فرض *0000* در نظر گرفته شود، حداکثر ۸ ثانیه طول می کشد تا شمارنده سرریز شود.

هنگامی که منبع پالس ساعت اعمالی از یک منبع منظم و معین نباشد (منبع خارجی)، در این صورت می توان از شمارنده برای شمارش وقایع خارجی نظیر عبور تعداد قطعات از جلوی یک سنسور نوری استفاده نمود که در این حالت شمارنده به صورت کانتر به کار گرفته می شود.

۲ ـ تایمر / کانتر صفر

تایمر کانتر صفر در *AVR* ها را می توان در سه نوع عملکرد زیر دسته بندی کرد :

۱ ـ نوع عملکرد ساده هشت بیتی : این مدل فقط در سری *At90S , ATTiny* به کار گرفته شده است.

۲ ـ نوع عملکرد پیشرفته هشت بیتی : این مدل در سری های *ATmega* (به غیر از *ATmega8* و *ATmega163* که از مدل ساده ۸ بیتی استفاده می کنند) به کار گرفته شده است.

۳ ـ نوع عملکرد پیشرفته شانزده بیتی : این مدل فقط در *AVR* های سری *ATTiny2313* و *ATTiny13* به کار گرفته است.

۳ ـ تایمر کانتر صفر در حالت هشت بیتی پیشرفته

حالت عملکرد هشت بیتی در سری های *ATmega* (به غیر از *ATmega8* و *ATmega163*) قرار دارد. از خصوصیات تایمر/کانتر در این مدل می توان به موارد زیر اشاره کرد :

۱ ـ تایمر / کانتر در حالت عادی

۲ ـ تایمر / کانتر در حالت مقایسه *CTC*

۳ ـ تایمر / کانتر در حالت *PWM* سریع (تک شیب)

۴ ـ تایمر / کانتر در حالت *PWM* تصحیح فاز (دو شیب)

۴ ـ رجیستر های تایمر / کانتر صفر در حالت عملکرد هشت بیتی پیشرفته

۴-۱ رجیستر مقایسه خروجی (*OCR0*)

این رجیستر هشت بیتی، خواندنی و نوشتنی بوده و به طور مستقیم با مقدار شمارنده *TCNT0* مقایسه می شود. از تطابق این دو برای تولید وقفه خروجی یا تولید یک شکل موج روی پایه *OC0* می توان استفاده نمود.

Bit	7	6	5	4	3	2	1	0
	OCR0[*7:0*]							
Read/Write	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Initial Value	0	0	0	0	0	0	0	0

۴-۲ رجیستر تایمر کانتر صفر *TCNT0*

این رجیستر هشت بیتی امکان دسترسی مستقیم برای خواندن و نوشتن در شمارنده را فراهم می کند. این رجیستر، هم خواندنی است و هم نوشتنی. به هنگام خواندن، مقدار شمارش شده را از خروجی شمارنده برمی گرداند و به هنگام نوشتن، مقدار جدید را به ورودی شمارنده انتقال می دهد.

Bit	7	6	5	4	3	2	1	0
	TCNT0[*7:0*]							
Read/Write	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Initial Value	0	0	0	0	0	0	0	0

۴-۳ رجیستر کنترلی تایمر/کانتر صفر *TCCR0*

این رجیستر دارای هشت بیت کنترلی برای انتخاب پالس ساعت، حالت خروجی هنگام تطابق مقایسه، عملکرد های *PWM*، و بیت مقایسه خروجی می باشد.

با استفاده از رجیستر *TCCR0* می توان فرکانس پالس ساعت تایمر را انتخاب کرد که برای این کار از بیت های انتخاب پالس ساعت (*CS00*، *CS01*، *CS02*) استفاده می شود.

Bit	7	6	5	4	3	2	1	0
	FOC0	*WGM00*	*COM01*	*COM00*	*WGM01*	*CS02*	*CS01*	*CS00*
Read/Write	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Initial Value	0	0	0	0	0	0	0	0

ـ بیت های انتخاب پالس ساعت (*CS00*، *CS01* و *CS02*)

این سه بیت برای انتخاب فرکانس پالس ساعت تایمر صفر مورد استفاه قرار می گیرند. جدول ۱ و ۲ انتخاب پالس ساعت برای میکروکنترلرهای *ATmega128*،*ATmega64* و *ATmega32* را نشان می دهد. تایمر در زمانی که هیچ منبع پالس ساعتی انتخاب نشود، غیرفعال است.

جدول ۱ : انتخاب پالس ساعت برای میکروکنترلرهای *ATmega64* و *ATmega128*

CS02	*CS01*	*CS00*	*Description*
0	*0*	*0*	*No Clock Source (Timer/Counter Stopped)*
0	*0*	*1*	*$CLK_{I/O}$ (No Prescaling)*
0	*1*	*0*	*$CLK_{I/O}/8$ (From Prescaler)*
0	*1*	*1*	*$CLK_{I/O}/32$ (From Prescaler)*
1	*0*	*0*	*$CLK_{I/O}/64$ (From Prescaler)*
1	*0*	*1*	*$CLK_{I/O}/128$ (From Prescaler)*
1	*1*	*0*	*$CLK_{I/O}/256$ (From Prescaler)*
1	*1*	*1*	*$CLK_{I/O}/1024$ (From Prescaler)*

جدول ۲ : انتخاب پالس ساعت برای میکروکنترلرهای *ATmega32*

CS02	*CS01*	*CS00*	*Description*
0	*0*	*0*	*No Clock Source (Timer/Counter Stopped)*
0	*0*	*1*	*$CLK_{I/O}$ (No Prescaling)*
0	*1*	*0*	*$CLK_{I/O}/8$ (From Prescaler)*
0	*1*	*1*	*$CLK_{I/O}/64$ (From Prescaler)*
1	*0*	*0*	*$CLK_{I/O}/256$ (From Prescaler)*
1	*0*	*1*	*$CLK_{I/O}/1024$ (From Prescaler)*
1	*1*	*0*	*External clock source on T0 pin. Clock on falling edge.*
1	*1*	*1*	*External clock source on T0 pin. Clock on rising edge.*

ـ بیت های حالت خروجی تطابق مقایسه (*COM01:0*)

این بیت ها رفتار پایه مقایسه خروجی (*OC0*) را کنترل می کنند. زمانی که این بیت ها هر دو صفر باشند، پایه *OC0* که یکی از پایه های یک *Port* است، به صورت *I/O* معمولی استفاده می شود. اما اگر یکی یا هر دوی این بیت ها یک شوند، پایه *OC0* به خروجی واحد تولید شکل موج متصل خواهد شد و دیگر نمی توان از آن به عنوان *I/O* معمولی استفاده نمود (جدول ۴).

ـ بیت های *WGM01* و *WGM00*

توسط این دو بیت می توان حالت های عملکرد پشتیبانی شده تایمر/کانتر را عبارتند از : عملکرد عادی، عملکرد مقایسه ای، عملکرد با *PWM* سریع و عملکرد با *PWM* تصحیح فاز را انتخاب نمود (جدول ۳).

جدول ۳ : انتخاب مد تولید موج

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0 at	TOV0 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	TOP
3	1	1	Fast PWM	0xFF	TOP	TOP

ـ بیت مقایسه خروجی (*FOC0*)

این بیت زمانی فعال می شود که بیت های *WGM*، یک حالت غیر *PWM* (عملکرد عادی و مقایسه) را انتخاب کرده باشند. در هنگام نوشتن یکِ منطقی در این بیت، یک تطابق مقایسه فوری روی واحد تولید شکل موج رخ می دهد و پایه *OC0* را با توجه به تنظیم بیت های *COM01* و *COM00* تغییر می دهد.

در زمان یک کردن *FOC0* پرچم مقایسه خروجی (*OCF0*) فعال خواهد شد و وقفه ای نیز تولید نمی شود. این بیت در کل، تأثیری بر هیچ رجیستر و پرچمی نمی گذارد و فقط وضعیت پایه *OC0* را با توجه به تنظیمات آن تغییر می دهد.

۴-۴ رجیستر پرچم وقفه تایمر کانتر صفر *TIFR*

از این رجیستر برای تشخیص یک سرریز یا تطابق در مقایسه در تایمر/کانتر صفر رخ داده باشد.

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

ـ بیت پرچم سرریز تایمر/کانتر صفر (*TOV0*)

بیت *TOV0* زمانی فعال (یک) می شود که یک سرریز در تایمر/کانتر صفر به وقع بپیوندد. برای پاک کردن این بیت باید در آن یک نوشت. البته در زمان اجرای بردار وقفه، *TOV0* توسط سخت افزار پاک می شود. در حالت *PWM* تصحیح فاز، *TOV0* زمانی فعال می شود که جهت شمارش در *$00* عوض شود.

ـ بیت پرچم مقایسه خروجی صفر (*OCF0*)

بیت *OCR0* زمانی فعال (یک) می شود که یک تطابق مقایسه مابین تایمر/کانتر صفر (*TCNT0*) و داده درون رجیستر *OCR0* به وقوع بپیوندد. برای پاک کردن این بیت باید در آن یک نوشت. البته در زمان اجرای بردار وقفه، *OCR0* توسط سخت افزار پاک می شود.

۵-۴ رجیستر پوشش وقفه تایمر/کانتر صفر (*TIMSK*)

Bit	7	6	5	4	3	2	1	0
	OCIE2	*TOIE2*	*TICIE1*	*OCIE1A*	*OCIE1B*	*TOIE1*	*OCIE0*	*TOIE0*
Read/Write	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Initial Value	0	0	0	0	0	0	0	0

بیت فعالساز وقفه سرریز تایمر/کانتر صفر (*TOIE0*)

زمانی که بیت *TOIE0* یک شود و همچنین بیت *I* در *SREG* نیز فعال باشد وقفه سرریز تایمر/کانتر صفر فعال می شود. هنگامی که بیت *TOV0* در رجیستر پرچم وقفه (*TIFR*) فعال شود، یک وقفه درخواست خواهد شد.

ـ بیت فعالساز وقفه تطابق مقایسه خروجی تایمر/کانتر صفر

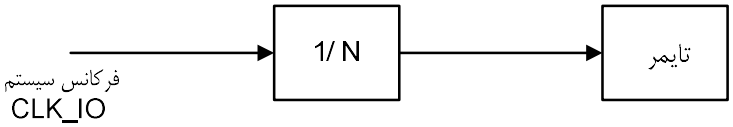
وقتی *OCIE0* یک شده و بیت *I* (فعالساز وقفه سراسری) در *SREG* نیز فعال باشد، وقفه تطابق مقایسه تایمر/کانتر صفر فعال می شود. هنگامی که بیت *OCF0* در رجیستر پرچم وقفه (*TIFR*) فعال شود، یک وقفه درخواست خواهد شد.

۵ ــ نحوه محاسبه زمان بندی برای تایمرها

برای محاسبه زمان بندی مورد نظر در تایمرها باید مراحل زیر انجام پذیرد :

۱ ـ فرکانس داخلی پالس ساعت سیستم را در نظر گرفت.

۲ ـ ضریب تقسیم پالس ساعت سیستم برای تولید پالس ساعت تایمر توسط بیت های *CS00*، *CS01* و *CS02* را تعیین نمود (شکل ۲).



شکل ۲ : تعیین ضریب تقسیم پالس ساعت تایمر

۳ ـ مقدار مطلوب را رجیستر *TCNTx* قرار داد (*x* شماره تایمر مورد نظر است).

مثال ۱ : با فرض نوسان ساز داخلی *MHz* ۱ و ضریب تقسیم *N*=۳۲ و *TCNT0=0AH* مقدار زمان تا فعال شدن پرچم سرریز را محاسبه کنید.

فرکانس پالس ساعت تایمر = $\frac{1\ MHz}{32} = 31.25\ KHz$

مدت زمان یک شمارش = $\frac{1}{32.25\ KHz} = 32\ \mu s$

تعداد پالس برای سرریز شدن تایمر = $256 - TCNT0 = 256 - 10 = 246$

مدت زمان شمارش تایمر = $246 \times 32\ \mu s = 7.872\ ms$

مثال ۲ : با فرض استفاده از کریستال خارجی *MHz* ۱۶ برای تولید مدت زمان *ms* ۱۰ چه عددی را باید در تایمر یک (*TCNT1*) قرار داد ؟

با فرض انتخاب ضریب تقسیم ۶۴ (*N*=۶۴) داریم :

فرکانس پالس ساعت تایمر = $\frac{16\ MHz}{64} = 250\ KHz$

مدت زمان یک شمارش = $\frac{1}{250\ KHz} = 4\ \mu s$

تعداد پالس لازم برای مدت *ms* ۱۰ = $\frac{10\ ms}{4\ \mu s} = 2500$

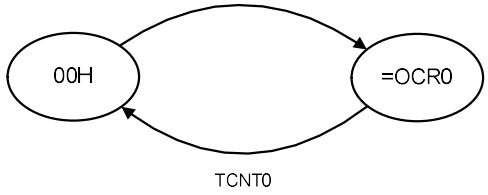
عدد لازم برای *TCNT1* = $65536 - 2500 = 63036 = F63CH$

۶ ـ تایمر/کانتر صفر در حالت عادی

ساده ترین حالت عملکرد تایمر/کانتر، حالت عادی می باشد. جهت شمارش، رو به بالا (افزایشی) بوده و پس از رسیدن مقدار رجیستر *TCNT0* به *TOP* دوباره از مقدار *BOTTOM* شروع به شمارش نموده و پرچم سرریز *TOV0* فعال خواهد شد. مقدار *TOP* برابر حداکثر مقدار شمارش در یک چرخه است و *BOTTOM* مقداری است که با رسیدن شمارنده به آن، شمارش صفر می شود.

۷ ـ عملکرد تایمر/کانتر صفر در حالت مقایسه (*CTC*)

در حالت مقایسه، رجیستر *TCNT0* به طور دایم با رجیستر *OCR0* مقایسه و در صورت تطابق، رجیستر *TCNT0* برابر با صفر می شود (شکل ۳).



شکل ۳ : نمودار شمارش تایمر/کانتر صفر در حالت *CTC*

از نتیجه مقایسه می توان برای تولید شکل موج روی پایه مقایسه خروجی *OC0* استفاده کرد (قسمت اول آزمایش). در هنگام تطبیق مقایسه در صورت فعال بودن وقفه و تولید پرچم *OC0* می توان یک وقفه مقایسه را ایجاد نمود و از روال وقفه برای به روز رسانی مقدار رجیستر *OCR0* (*TOP*) استفاده نمود. به هر حال تغییر رجیستر *OC0* (*TOP*) به یک مقدار جدید در زمانی که تایمر در حال شمارش است باید با احتیاط انجام شود، زیرا حالت *CTC* دارای بافر مضاعف نمی باشد. حالت های متفاوت بیت های *COM00* و *COM01* در جدول ۴ آورده شده است.

جدول ۴ : تعیین حالت خروجی تطابق مقایسه (*CTC*)

COM01	***COM00***	***Description***
0	*0*	*Normal port operation, OC0 disconnected.*
0	*1*	*Toggle OC0 on compare match*
1	*0*	*Clear OC0 on compare match*
1	*1*	*Set OC0 on compare match*

همانگونه که از جدول ۲ مشاهده می شود، چهار حالت به وجود می آید که عبارتند از :

۱ ـ پایه *OC0* که پایه ای از یک *Port* می باشد به صورت *I/O* معمولی استفاده می شود.

۲ ـ پایه *OC0* با خروجی مقایسه منطبق شده و در هر بار تطابق، خروجی تغییر وضعیت می دهد (*Toggle*).

۳ ـ پایه *OC0* با خروجی مقایسه منطبق شده و در زمان تطابق، این پایه را صفر می کند.

۴ ـ پایه *OC0* با خروجی مقایسه منطبق شده و در زمان تطابق، این پایه را یک می کند.

نکته : در حالت های ۲، ۳ و ۴ باید پین *OC0* به صورت خروجی تعریف شود. (با نوشتن یک در *DDRx*).

۸ ـ عملکرد تایمر/کانتر صفر درحالت *PWM* سریع (تک شیب)

واژه *PWM* مخفف عبارت *Pulse Width Modulation* یا مدولاسیون پهنای پالس می باشد. در این مدولاسیون پهنای پالس تولیدی را می توان تحت کنترل داشت، به طوری که این پهنا در برخی مواقع می تواند متأثر از مقدار دامنه یک موج دیگر باشد. از کاربردهای *PWM* می توان به کنترل دور موتورهای *AC*، *DC* و منابع تغذیه سوییچینگ و ... اشاره نمود.

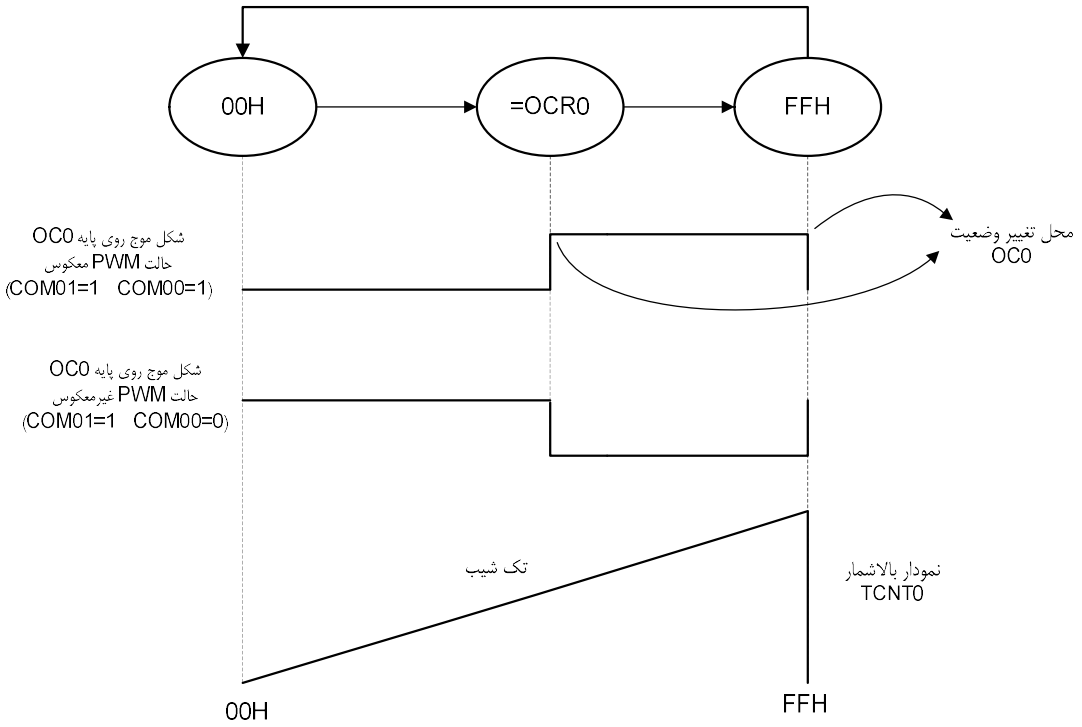
جهت ورود و دسترسی به مد عملکرد *PWM* سریع باید بیت های *WGM00=1* و *WGM01=1* قرار داد. همانگونه که درباره حالت مقایسه *CTC* توضیح داده شد، رجیستر *OCR0* به طور دایم با رجیستر *TCNT0* مقایسه می شود و پایه *OC0* با توجه به تنظیمات مربوطه، تغییر وضعیت می دهد. عملکرد تایمر در حالت *PWM* سریع همانند حالت مقایسه می باشد با این تفاوت که در زمان تطابق بین *OCR0* و *TCNT0* رجیستر *TCNT0* با مقدار *00H* پر نمی شود، بلکه شمارش خود را تا حداکثر مقدار (*FFH*) ادامه داده و پس از سرریز شدن تایمر، رجیستر *TCNT0* با مقدار *00H* پر می شود و شمارش ادامه پیدا می کند. رجیستر *OCR0* در حالت *PWM* دارای بافر مضاعف می باشد (شکل ۴).

باید توجه داشت که در حالت *PWM* پایه *OC0* در دو حالت زیر تغییر وضعیت می دهد :

۱ ـ در حالت تطابق

۲ ـ در زمان سرریز تایمر

در صورتی که در حالت *CTC* فقط در زمان تطابق، وضعیت پایه *OC0* تغییر می کند.



شکل ۴ : نمودار عملکر تایمر/کانتر صفر در حالت *PWM* سریع

در زمانی که تایمر/کانتر صفر در حالت *PWM* سریع قرار می گیرد، بیت های *COM00* و *COM01* عملکرد متفاوتی با حالت مقایسه (*CTC*) پیدا می کنند. این حالت ها در جدول ۵ آورده شده است.

جدول ۵ : حالت خروجی مقایسه در *PWM* سریع

COM01	***COM00***	***Description***
0	*0*	*Normal port operation, OC0 disconnected.*
0	*1*	*Reserved*
1	*0*	*Clear OC0 on compare match, set CO0 at TOP*
1	*1*	*Set OC0 on compare match, clear CO0 at TOP*

در صورتی که وقفه سرریز تایمر فعال باشد(وقفه سراسری فعال)، با هر بار فعال شدن *TOV0* می توان در زیربرنامه روال سرویس وقفه (*ISR*) رجیستر *OCR0* را *Update* نمود تا پهنای *PWM* به مقدار دلخواه تنظیم شود.

فرکانس *PWM* خروجی را می توان از رابطه زیر محاسبه کرد :

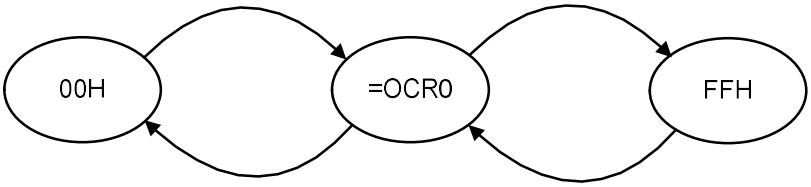
$$f_{PWM} = \frac{f_{clk\_I/O}}{N*255} \qquad (۱)$$

که در آن N بیانگر ضریب تقسیم پالس ساعت سیستم برای تولید پالس ساعت تایمر می باشد و برابر یکی از اعداد ۱، ۸، ۳۲، ۶۴، ۱۲۸، ۲۵۶ و ۱۰۲۴ است.

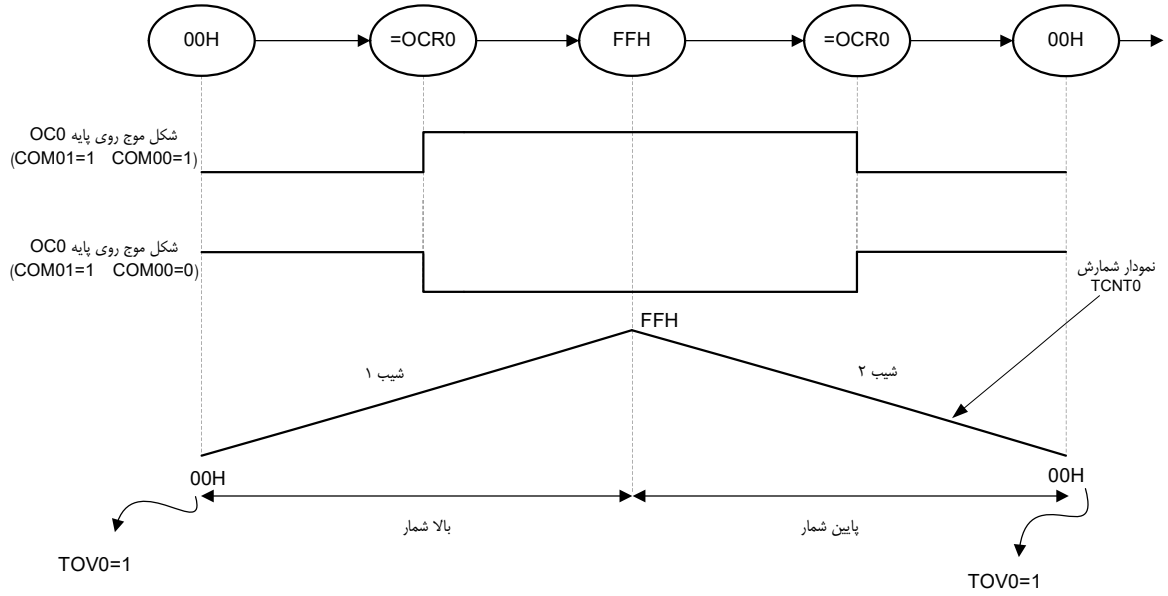
۸ ـ عملکرد تایمر/کانتر صفر درحالت *PWM* تصحیح فاز (دو شیب)

حالت *PWM* تصحیح فاز (*WGM00=1* و *WGM01=0*) امکان تولید شکل موج *PWM* با تفکیک پذیری بالا را مهیا می سازد. در *PWM* تصحیح فاز مکرراً از *00H* (*BOTTOM*) تا *FFH* (*TOP*) و سپس از *FFH* تا *00H* شمارش می نماید. در هنگام شمارش به سمت بالا مقایسه گر، رجیستر *TCNT0* را با *OCR0* مقایسه می کند و هنگام برابر شدن، یک تغییر وضعیت روی پایه *OC0* ایجاد می نماید و هنگام شمارش به سمت پایین، نیز همین عمل مجدداً تکرار می شود.

حالت عملکرد *PWM* به روش تصحیح فاز دارای فرکانس تولیدی پایین تری نسبت به *PWM* سریع می باشد. با این وجود با توجه به متقارن بودن عملکرد *PWM* تصحیح فاز، از آن بیشتر در کنترل دور موتور استفاده می شود. شکل های ۵ و ۶ نحوه عملکرد تایمر را در *PWM* تصحیح فاز نشان می دهند.



شکل ۵ : نمودار شمارش تایمر در حالت *PWM* تصحیح فاز



شکل ۶ : عملکرد تایمر در حالت *PWM* تصحیح فاز

یک نکته مهم این است که پرچم سرریز تایمر صفر (*TOV0*) زمانی فعال می شود که رجیستر *TCNT0* برابر صفر باشد نه *FFH*. پس باید توجه داشت که در زمان شروع به کار تایمر، اگر *TCNT0=0* باشد، پرچم سرریز فعال خواهد شد.

فرکانس موج *PWM* در حالت تصحیح فاز طبق معادله زیر قابل محاسبه است :

$$f_{PWM} = \frac{f_{clk\_I/O}}{N*510} \qquad (۲)$$

که در آن N بیانگر ضریب تقسیم پالس ساعت سیستم برای تولید پالس ساعت تایمر می باشد و برابر یکی از اعداد ۱، ۸، ۳۲، ۶۴، ۱۲۸، ۲۵۶ و ۱۰۲۴ است.

برای حالتی که تایمر/کانتر صفر در حالت *PWM* تصحیح فاز قرار می گیرد، بیت های *COM00* و *COM01* عملکرد متفاوتی پیدا می کنند که در جدول ۶ آورده شده است.

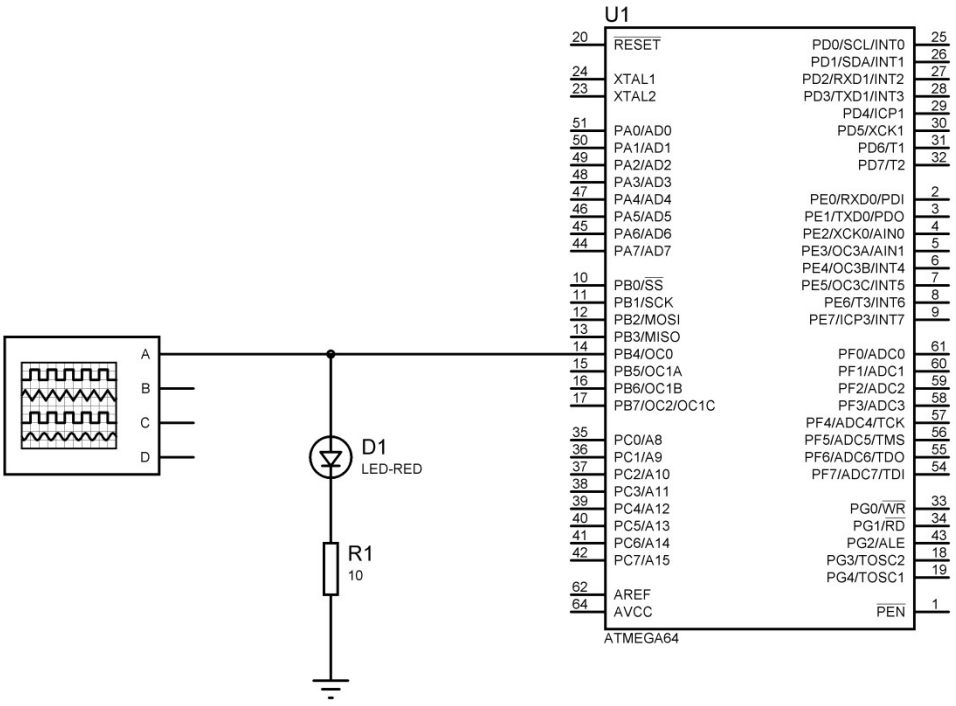
جدول ۶ : حالت خروجی مقایسه در *PWM* تصحیح فاز

COM01	*COM00*	*Description*
0	*0*	*Normal port operation, OC0 disconnected.*
0	*1*	*Reserved*
1	*0*	*Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.*
1	*1*	*Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.*

شرح آزمایش

ـ قسمت اول :

ابتدا مدار شکل ۲ رابیندید.



شکل ۲ : مدار قسمت اول آزمایش

۲ ـ با استفاده از برنامه زیر می توان هر *ms* ۱ پایه *OC0* (*PORTB.4* در *ATmega64*) را مکمل کرد. فرکانس پالس ساعت سیستم برابر با *MHz* ۸ فرض شده است.

```
#include <mega64.h>
void main(void){
        DDRB=0x10;
        TCNT0=0x00;
        OCR0=0x7C;
        TCCR0=0x1C;
        while (1);
}
```

در برنامه بالا مراحل زیر انجام می شود :

۱ ـ تعیین بیت چهارم *PortB* به عنوان خروجی جهت فعال شدن *OC0* به صورت خروجی.

۲ ـ مقداردهی اولیه در رجیستر تایمر صفر (*TCNT0*) با مقدار *00H* .

۳ ـ مقداردهی رجیستر مقایسه خروجی با *7CH* (۱۲۴).

۴ ـ تنظیم فرکانس پالس ساعت تایمر صفر روی *KHz* ۱۲۵ (با فرض فرکانس پالس ساعت سیستم *MHz* ۸) و قرار دادن تایمر صفر در حالت عملکرد مقایسه و تنظیم عملکرد شکل موج روی پایه *OC0* به صورت مکمل (*Toggle*) و شروع شمارش تایمر.

۵ ـ پایه *OC0* (*PortB.4*) به صورت خودکار پس از *ms* ۱ مکمل می شود.

برای محاسبه زمان تایمر می توان از فرمول زیر استفاده نمود :

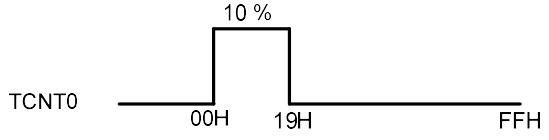
$$Time = \frac{N(1+OCR\ )}{fclk\_IO} \qquad (۳)$$

که در آن N ضریب تقسیم (۱، ۸، ۳۲، ۶۴، ۱۲۸، ۲۵۶، ۱۰۲۴) و *fclk_IO* فرکانس پالس ساعت سیستم می باشد.

ـ قسمت دوم :

در این قسمت، می خواهیم برنامه ای را بنویسیم که یک موج *PWM* با عرض پالس ٪ ۱۰ را روی پایه *OC0* ایجاد نماید. (با استفاده از کریستال *MHz* ۱۶ خارجی)

برای به دست آوردن پهنای پالس ٪ ۱۰ می توان به صورت زیر عمل نمود :



شکل ۳ : ایجاد پهنای پالس ٪ ۱۰

بنابراین داریم :

$$Duty\ Cycle = \frac{Time\ On}{Time\ On + Ti\quad Off} \times 100 \qquad (۴)$$

و

$$\frac{255}{OCR0} = \frac{100\ \%}{10\ \%} = OCR0 = 25.5 \ \rightarrow \ \ OCR0 = 25 \qquad (۵)$$

البته در این روش، یک خطا در پهنای پالس ایجاد می شود که می توان از آن صرفنظر نمود. برنامه زیر را می توان برای ایجاد چنین شکل موجی به کار برد.

```
#include <mega64.h>
void main(){
    DDRB=0x10;
    TCNT0=0x00;
    OCR0=25;
    TCCR0=0x6A;
    while(1);
}
```

مدار این قسمت از آزمایش، همانند قسمت قبل می باشد.

قسمت سوم :

با استفاده از تایمر صفر برنامه ای بنویسد که اعداد صفر تا ۹ را به ترتیب بر روی یک *7segment* و با فاصله زمانی ۰/۲۵ ثانیه نشان دهد. فرکانس پالس ساعت را برابر با *MHz* ۱ و N را برابر با ۱۰۲۴ در نظر بگیرید.

پرسش :

۱ ـ در برنامه قسمت دوم آزمایش، منظور از خطای ذکر شده چیست و چگونه می توان آن را برطرف کرد ؟ (برنامه آن را بنویسید).

۲ـ منظور از بافر مضاعف چیست ؟

۳ ـ در برنامه ساعت دیجیتال در آزمایش ۵ برای ایجاد تأخیر به جای استفاده از دستور *delay_ms()* از تایمر استفاده کنید.

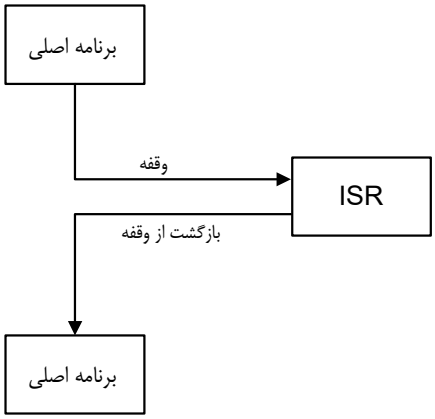
آزمایشگاه ریزپردازنده ها

آزمایش شماره ۷

آشنایی با وقفه و سازمان وقفه در میکروکنترلر AVR

مقدمه

وقفه، مکانیزمی از میکروکنترلر است که میکروکنترلر را برای پاسخ گویی به برخی از وقایع لحظه ای فعال می کند. وقفه ها نقش مهمی در میکروکنترلرها ایفا می کنند. در زمان رخداد یک وقفه، برنامه در حال اجرا متوقف شده و زیربرنامه وقفه شروع به اجرای برنامه خود می کند و پس از پایان زیربرنامه موردنظر، اجرای برنامه اصلی ادامه می یابد. برنامه ای را که CPU در زمان رخداد یک وقفه به آن رجوع می کند، روال سرویس وقفه (ISR) می نامند. شکل ۱ نحوه عملکرد میکروکنترلر را به هنگام رخداد وقفه نشان می دهد.



شکل ۱ : عملکرد میکروکنترلر هنگام رخداد وقفه ها

۱ ـ مراحل اجرای یک وقفه

میکروکنترلر به محض دریافت یک وقفه، مراحل زیر را انجام می دهد :

۱ ـ دستوری که در حال اجرا می باشد را پایان داده و آدرس دستورالعمل بعدی را در پشته ذخیره می نماید.

۲ ـ رجوع به جدول بردار وقفه (محلی از حافظه) و به دست آوردن آدرس سرویس وقفه (ISR) و پرش به آن آدرس.

۳ ـ شروع مرحله اجرای ISR تا رسیدن به دستور بازگشت از وقفه ($RETI$).

۴ ـ برداشتن آدرس از پشته و قرار دادن آن در شمارنده برنامه (PC) و اجرای ادامه برنامه.

۲ ـ سازمان وقفه در *AVR*

در میکروکنترلرهای *AVR* منابع وقفه با توجه به نوع میکروکنترلر متفاوت است. به عنوان نمونه در میکروکنترلر *ATmega8* تعداد ۹ منبع وقفه و در *ATmega32* تعداد ۲۱ منبع وقفه وجود دارد.

۲-۱ بردار وقفه

هنگام رخداد یک وقفه، آدرسی که در شمارنده برنامه قرار می گیرد را بردار وقفه می نامند. این آدرس، همان آدرس شروع *ISR* مربوط به منبع وقفه است. در میکروکنترلرهای *AVR* برخلاف دیگر خانواده میکروکنترلرها نظیر ۸۰۵۱ که بردارهای وقفه در یک مکان ثابت و مشخص در ابتدای حافظه برنامه قرار داشتند، این قابلیت به کاربر داده شده است که مکان بردار وقفه را بین دو بخش حافظه برنامه کاربردی و *BOOT* حرکت دهد. پس با توجه به این موضوع می توان میکروکنترلرهای *AVR* را به صورت زیر تقسیم بندی نمود :

الف) *AVR* بدون حافظه *BOOT* :

در این نوع میکروکنترلرها، فقط حافظه کاربردی وجود دارد، مانند سری های *ATTiny* و *AT80s*. در این *AVR*ها بردارهای وقفه در یک مکان ثابت و مشخص در ابتدای حافظه برنامه قرار داده شده که در هنگام وقوع وقفه، *CPU* به آن محل از حافظه برنامه رجوع کرده و از آنجا شروع به اجرای برنامه می کند. اگر وقفه ای استفاده نشود، خانه مربوط به هر یک از وقفه ها می تواند به صورت خانه معمولی حافظه برنامه در نظر گرفته شود.

ب) *AVR* با حافظه *BOOT* :

این تقسیم بندی برای میکرو کنترلرهای سری *ATmega* بوده (به غیر از *ATmega103*، *ATmega603* و *ATmega48*) که هم حافظه *BOOT* و هم حافظه کاربردی را پشتیبانی می نمایند.

۳ ـ رجیستر کنترل وقفه عمومی (*GICR*)

این ثبات در شکل ۲ نشان داده شده است.

Bit	7	6	5	4	3	2	1	0
	INT1	*INT0*	*INT2*	-	-	-	*IVSEL*	*IVCE*
Read/Write	*R/W*	*R/W*	*R/W*	*R*	*R*	*R*	*R/W*	*R/W*
Initial Value	0	0	0	0	0	0	0	0

شکل ۲ : رجیستر *GICR*

دو بیت از بیت های مهم این ثبات عبارتند از :

ـ بیت انتخاب بردار وقفه (*IVSEL*)

از این بیت برای انتخاب بردار وقفه استفاده می شود. زمانی که *IVSEL=0* باشد، بردار وقفه در آغاز حافظه کاربردی و اگر *IVSEL=1* باشد، بردار وقفه در آغاز حافظه *BOOT* قرار می گیرد.

ـ بیت فعال ساز تعیین بردار وقفه (*IVCE*)

برای تغییر *IVSEL*، ابتدا باید بیت *IVCE* را یک کرد و سپس مقدار دلخواه را در *IVSEL* قرار داد. بیت *IVCE* به طور خودکار و به صورت سخت افزاری پس از گذشت چهار سیکل صفر می شود.

نکته : بیت های *IVSEL* و *IVCE* در میکروکنترلرهای *ATmega64*، *ATmega169* و *ATmega128* در رجیستر *MCUCR* قرار دارند.

برای جلوگیری از تغییرات غیرعادی در جدول بردار وقفه، مراحل زیر را دنبال نمایید :

۱ ـ فعال کردن بیت *IVCE*

۲ ـ با فعال کردن *IVCE* به مدت چهار سیکل فرصت دارید تا در *IVSEL* مقدار دلخواه خود را بنویسید. پس از این مدت به طور خودکار *IVCE* صفر می شود. در این مدت، وقفه ها به صورت خودکار غیرفعال خواهند بود.

۴ ـ وقفه های خارجی (*External Interrupts*)

میکروکنترلرهای *AVR*، علاوه بر وقفه های داخلی مانند وقفه تایمرها، سریال، مقایسه کننده آنالوگ و ... از وقفه های خارجی نیز پشتیبانی می کنند. این وقفه ها با توجه به نوع میکروکنترلرها می توانند از یک (در میکروکنترلر *ATtiny13*) تا هشت (در میکروکنترلرهای *ATmega103* و *ATmega128*) وقفه خارجی باشند. میکروکنترلر *ATmega32* از سه وقفه خارجی (*INT0*، *INT1* و *INT2*) پشتیبانی می کند.

وقفه های خارجی توسط پین های *INT0-7* راه اندازی می شوند. حتی اگر پین های *INT0-7* به صورت خروجی پیکربندی شده باشند، وقفه ها راه اندازی خواهند شد. وقفه های خارجی می توانند با یک لبه پایین رونده یا بالا رونده و یا یک سطح پایین فعال شوند. یکی از خصوصیات وقفه ها تشخیص غیرهمزمان آنها می باشد که از این خصوصیت می توان برای بیدار کردن *CPU* از بخشی از مدهای *sleep* به غیر از مد *Idle* (در این مد، پالس ساعت *I/O* متوقف است) استفاده نمود.

۵ ـ فعال کردن وقفه ها

۵-۱ فعال نمودن وقفه سراسری

پیش از به کارگیری وقفه باید بیت فعالساز وقفه عمومی (*I*) را فعال نمود. این بیت در رجیستر وضعیت (*SREG*) قرار دارد.

Bit	7	6	5	4	3	2	1	0
	I	*T*	*H*	*S*	*V*	*N*	*Z*	*C*
Read/Write	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Initial Value	0	0	0	0	0	0	0	0

I : بیت فعالساز وقفه عمومی

همانگونه که گفته شد، خانواده *AVR* تا هشت (از ۰ تا ۷) وقفه خارجی را پشتیبانی می کند. بنابراین رجیسترهای داخلی مربوط به کنترل و وضعیت هر وقفه در هر میکروکنترلر ممکن است متفاوت با دیگری باشد. در ادامه، به طور نمونه به بررسی رجیسترهای مربوط به وقفه *ATmega32* و *ATmeg128* خواهیم پرداخت که به ترتیب ۳ و ۸ منبع وقفه خارجی را پشتیبانی می کنند. البته اساس کار وقفه در تمام سری های *AVR* یکسان بوده و تنها تفاوت موجود، فقط در تعداد وقفه ها ونام رجیسترها خلاصه می شود.

۵-۲ رجیستر کنترل (*MCUCR*)

رجیستر کنترل *MCU* (*Master Control Unit*) شامل بیت های کنترل وقفه و عملکردهای *AVR* در حالت کلی می باشد.

Bit	7	6	5	4	3	2	1	0
	SE	*SM2*	*SM1*	*SM0*	*ISC11*	*ISC10*	*ISC01*	*ISC00*
Read/Write	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Initial Value	0	0	0	0	0	0	0	0

ـ بیت های *ISC00* و *ISC01*

از بیت های *ISC00* و *ISC01* جهت تعیین نوع فعالسازی وقفه خارجی روی لبه بالا رونده، لبه پایین رونده، حساس به سطح و یا حساس به هر تغییری بر روی پین *INT0* استفاده می شود، حتی اگر پین های مربوطه به صورت خروجی پیکربندی شده باشند (جدول ۱).

جدول ۱ : وضعیت های مختلف بیت های *ISC00* و *ISC01*

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.*

مقدار موجود روی پایه *INT0* قبل از تشخیص لبه ها، نمونه برداری می شوند. اگر وقفه لبه یا *Toggle* انتخاب شده باشد، پالس هایی که مدت زمان آنها بیش از سیکل پالس ساعت طول بکشد، وقفه را تولید خواهد کرد و در پالس های کوتاه، ضمانتی برای اجرای وقفه وجود ندارد. همچنین اگر وقفه سطح انتخاب شود، این سطح باید تا زمان اجرای دستورالعمل جاری تحت تولید وقفه پایین نگه داشته شود.

ـ بیت های *ISC10* و *ISC11*

وظیفه این بیت ها همانند بیت های *ISC00* و *ISC01* اما برای *INT1* است.

ـ بیت های *SM2-0*

از این بیت ها برای انتخاب مد *sleep* استفاده می شود.

ـ بیت *SE*

از این بیت برای فعال کردن مد *sleep* استفاده می شود.

نکته : رجیسترهای *EICRA* و *EICRB* در *ATmega128* همان وظیفه رجیستر *MCUCR* در *ATmega32* را برعهده دارند.

Bit	7	6	5	4	3	2	1	0
EICRA	ISC31	ISC30	ISC21	ISC21	ISC11	ISC10	ISC01	ISC00
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
EICRB	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

۳-۵ رجیستر وقفه و کنترل *AVR* (*MCUCSR*)

Bit	7	6	5	4	3	2	1	0
	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	See Bit Description				

ـ بیت *ISC2*

از این بیت برای تعین نوع فعالسازی وقفه خارجی ۲ در *ATmega32* استفاده می شود.

نکته : وقفه *INT2* در *Atmega32* فقط با لبه فعال خواهد شد.

اگر در *ISC2* صفر نوشته شود، *INT2* توسط یک لبه پایین رونده فعال و اگر در *ISC2* یک نوشته شود، *INT2* توسط یک لبه بالارونده فعال می شود.

پالس های تولیدکننده وقفه باید بیش از *ns* ۵۰۰ طول بکشد و در پالس های کوتاه تر از آن، ضمانتی برای تولید وقفه وجود ندارد.

نکته : هنگام تغییر بیت *ISC2* ممکن است وقفه ای رخ دهد؛ بنابراین پیشنهاد می شود برای جلوگیری از این حالت، ابتدا بیت *INT2* در رجیستر فعالساز وقفه (*GICR*) را غیرفعال کنید. سپس بیت *ISC2* را تغییر دهید و مجدداً بیت *INT2* در رجیستر *GICR* را فعال نمایید.

۵-۴ رجیستر کنترل وقفه عمومی (*GICR*)

به کمک این رجیستر می توان وقفه های خارجی را در *ATmwga32* فعال یا غیرفعال نمود.

Bit	7	6	5	4	3	2	1	0
	INT1	*INT0*	*INT2*	-	-	-	*IVSEL*	*IVCE*
Read/Write	*R/W*	*R/W*	*R/W*	*R*	*R*	*R*	*R/W*	*R/W*
Initial Value	0	0	0	0	0	0	0	0

بیت *INT0* : بیت فعالساز وقفه خارجی صفر

بیت *INT1* : بیت فعالساز وقفه خارجی یک

بیت *INT2* : بیت فعالساز وقفه خارجی دو

نکته : رجیستر *EIMSK* در *ATmega128* همان وظیفه *GICR* در *Atmega32* را برعهده دارد.

Bit	7	6	5	4	3	2	1	0
EIMSK	*INT7*	*INT6*	*INT5*	*INT4*	*INT3*	*INT2*	*INT1*	*INT0*
Read/Write	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Initial Value	0	0	0	0	0	0	0	0

۵-۵ رجیستر پرچم وقفه عمومی (*GIFR*)

در زمان وقوع وقفه، پرچم مربوط به آن در این رجیستر فعال می شود.

Bit	7	6	5	4	3	2	1	0
	INTF1	*INTF0*	*INTF2*	-	-	-	-	-
Read/Write	*R/W*	*R/W*	*R/W*	*R*	*R*	*R*	*R*	*R*

Initial Value	0	0	0	0	0	0	0	0

ـ بیت *INTF1* (پرچم وقفه خارجی ۱)

زمانی که یک تغییر سطح روی پین *INT1* رخ دهد، بیت *INTF1* یک خواهد شد و اگر بیت *I* در *SREG* و بیت *INT1* در *GICR* یک باشند، *CPU* به محل بردار وقفه پرش خواهد کرد. این پرچم در حین اجرای زیرروال وقفه پاک می شود. زمانی که *INT1* به صورت یک وقفه سطح پیکربندی شده باشد، به صورت خودکار صفر می شود.

ـ بیت *INTF2* (پرچم وقفه خارجی ۲)

عملکرد این بیت همانند *INTF1* است.

نکته : رجیستر *EIFR* در *ATmega128* همان وظیفه *GIFR* در *ATmega32* را بر عهده دارد.

Bit	7	6	5	4	3	2	1	0
EIFR	*INTF7*	*INTF6*	*INTF5*	*INTF4*	*INTF3*	*INTF2*	*INTF1*	*INTF0*
Read/Write	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*	*R/W*
Initial Value	0	0	0	0	0	0	0	0

در *AVR*ها امکان تعیین اولویت وقفه وجود ندارد و هر وقفه ای که در آدرس پایین تری قرار دارد، از اولویت بالاتری برخوردار می باشد.

۶ ـ برنامه نویسی وقفه ها در *Codevision*

برای استفاده از وقفه در *Codevision* از کلمه کلیدی *interrupt* به فرم زیر کمک می گیریم :

```
interrupt [Vector number] void int_expresion (void) {
    برنامه سرویس وقفه
}
```

عبارت *Vector number* شمارنده بردار وقفه مورد نظر بوده و از روی جدول بردار وقفه به دست می آید؛ به عنوان نمونه [۱] برای بردار *Reset* و [۲] برای بردار *INT0* و ... . در جدول ۲ بردارهای وقفه برای میکروکنترلر *ATmega32* آورده شده است.

عبارت *int_expresion* نام تابع سرویس دهنده وقفه بوده که می تواند توسط برنامه نویس نوشته شود.

نکته : قبل از به کارگیری این تابع باید رجیسترهای مربوط به وقفه را تنظیم نموده تا وقفه ها فعال شوند. همچنین تابع وقفه چیزی را باز نمی گرداند.

جدول ۲ : بردارهای وقفه برای *ATmega32*

Vector No.	***Program Address***	***Source***	***Interrupt Description***
1	*$000*	*RESET*	*External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset*
2	*$002*	*INT0*	*External Interrupt Request 0*
3	*$004*	*INT1*	*External Interrupt Request 1*
4	*$006*	*INT2*	*External Interrupt Request 2*
5	*$008*	*TIMER2 COMP*	*Timer/Counter2 Compare Match*
6	*$00A*	*TIMER2 OVF*	*Timer/Counter2 Overflow*
7	*$00C*	*TIMER1 CAPT*	*Timer/Counter1 Capture Event*
8	*$00E*	*TIMER1 COMPA*	*Timer/Counter1 Compare Match A*
9	*$010*	*TIMER1 COMPB*	*Timer/Counter1 Compare Match B*
10	*$012*	*TIMER1 OVF*	*Timer/Counter1 Overflow*
11	*$014*	*TIMER0 COMP*	*Timer/Counter0 Compare Match*
12	*$016*	*TIMER0 OVF*	*Timer/Counter0 Overflow*
13	*$018*	*SPI, STC*	*Serial Transfer Complete*
14	*$01A*	*USART, RXC*	*USART, Rx Complete*
15	*$01C*	*USART, UDRE*	*USART Data Register Empty*
16	*$01E*	*USART, TXC*	*USART, Tx Complete*
17	*$020*	*ADC*	*ADC Conversion Complete*
18	*$022*	*EE_RDY*	*EEPROM Ready*
19	*$024*	*ANA_COMP*	*Analog Comparator*
20	*$026*	*TWI*	*Two-wire Serial Interface*
21	*$028*	*SPM_RDY*	*Store Program Memory Ready*

شرح آزمایش

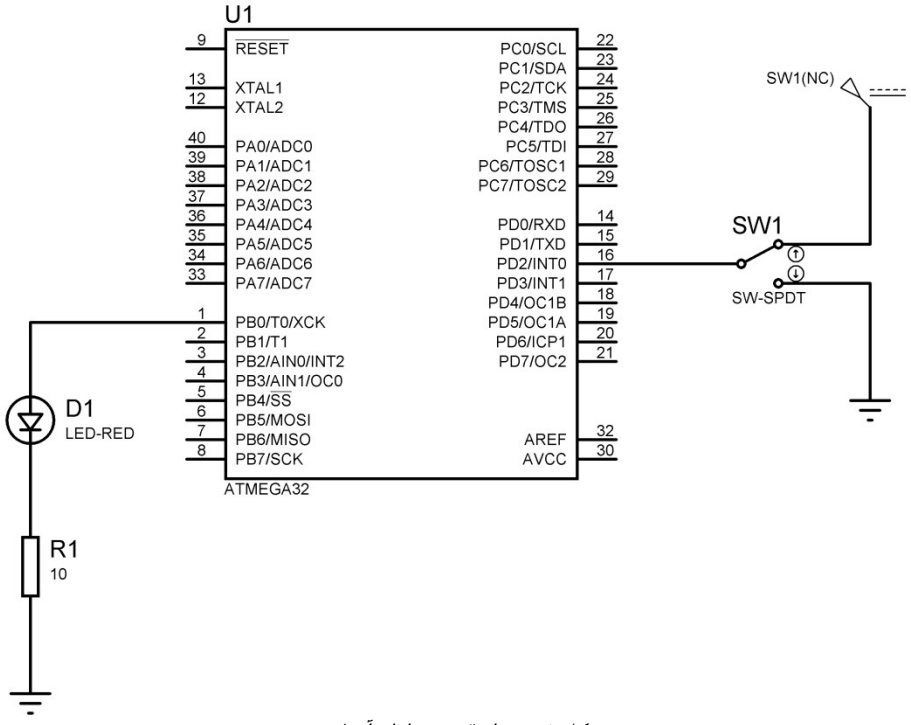
ـ قسمت اول :

می خواهیم برنامه ای را بنویسیم که هنگام وقوع وقفه خارجی صفر، *LED* متصل به *PortB.0* به مدت ۱ ثانیه روشن و ۲ ثانیه خاموش شود. توجه شود که از میکروکنترلر *ATmega32* استفاده نموده ایم. برای این منظور می توان برنامه زیر را نوشت :

```
#include <mega32.h>
#include <delay.h>
#define LED PORTB.0

void main(){
      DDRB=0xFF;              PORTB به صورت خروجی
      PORTB=0x00;
      GICR=0x40;              فعال نمودن وقفه خارجی صفر
      MCUCR=0x03;             فعال شدن وقفه با لبه بالارونده
      GIFR=0x40;
      #asm("sei");            فعال کردن وقفه سراسری
      while(1);               در انتظار وقفه
}
//-----------------------------
interrupt [2] void ext_int0 (void)
{
       #asm("cli");           غیر فعال کردن وقفه سراسری
       LED=1;
       delay_ms(1000);
       LED=0;
       delay_ms(2000);
       #asm("sei");           فعال کردن وقفه سراسری
}
```

این برنامه را به دقت بررسی نمایید و با بستن مدار نشان داده شده در شکل ۳ طرز کار آن را مشاهده و درستی برنامه را تحقیق نمایید.



شکل ۳ : مدار قسمت اول آزمایش

ـ قسمت دوم :

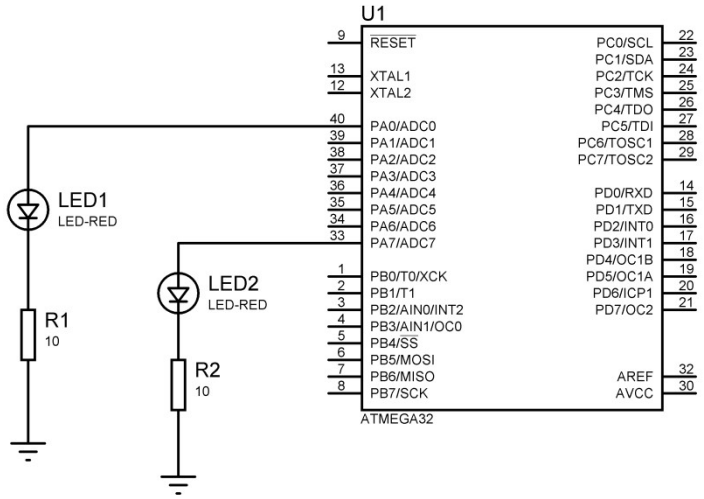
در این قسمت، می خواهیم یک وقفه داخلی را ایجاد کنیم. برای این منظور فرض می کنیم که تایمر صفر در حالت عملکرد *CTC* فعال شده باشد. می خواهیم هرگاه در مقایسه بین عدد موجود در رجیستر *TCNT0* و رجیستر *OC0* مطابقت وجود داشت، یک وقفه از نوع تایمر صفر فعال شود و *LED* متصل به *PortB.7* را برای *ms* ۱ روشن و دوباره خاموش نماید. باید توجه داشت که برای استفاده از وقفه تایمر صفر در حالت عملکرد *CTC* حتماً بایستی بیت *OCIE0* در رجیستر *TIMSK* یک شود. برنامه این قسمت از آزمایش در ادامه آورده شده است.

```
#include <mega32.h>
#include <delay.h>
#define xtal 8000000
#define LED1 PORTA.0
#define LED2 PORTA.7

void main(void){
      DDRA=0xFF;
      PORTA=0x00;
      TIMSK=0x02;
      LED1=1;
```

```
        TCNT0=0x00;
        OCR0=0xFF;
        TCCR0=0x1C;
        #asm("sei");
        while (1);
}
//---------------------------------
interrupt [11] void tim0_comp(void)
{
        #asm("cli");
         LED1=0;
         LED2=1;
         delay_ms(100);
         LED2=0;
         LED1=1;
         delay_ms(100);
         #asm("sei");
}
```

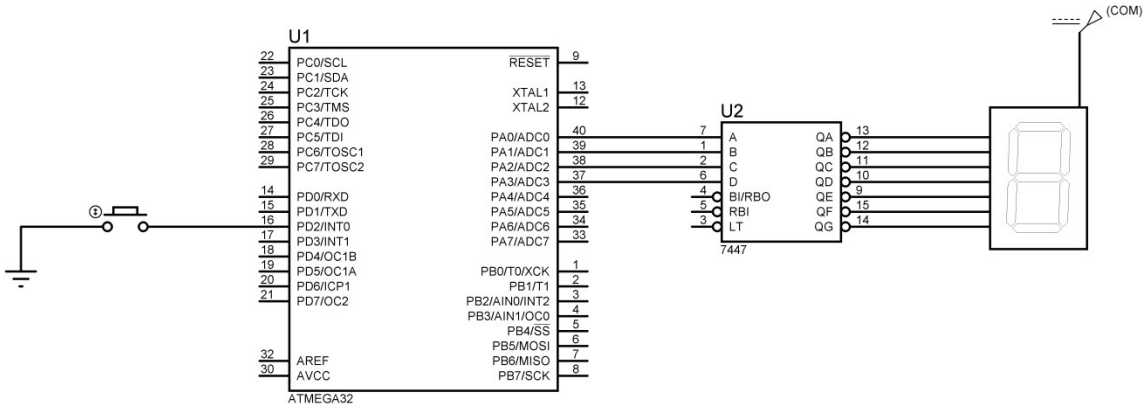
این برنامه را نیز به دقت مورد بررسی قرار داده و با بستن مدار نشان داده شده در شکل ۴ طرز کار آن را مشاهده و درستی برنامه را تحقیق نمایید.



شکل ۴ : مدار قسمت دوم آزمایش

ـ قسمت سوم :

برنامه ای بنویسد که اعداد صفر تا ۹ را به ترتیب بر روی یک *7segment* نشان دهد، اما هرگاه سوییچ متصل به *PortD.2* فشار داده شد، شمارش آن برعکس شود و از پایان شمارش معکوس، به کار عادی شمارش از ۰ تا ۹ بازگردد.



شکل ۵ : مدار قسمت سوم آزمایش

پرسش :

۱ ـ برنامه ای بنویسید که یک سیگنال مربعی را دریافت کند و فرکانس آن را محاسبه و بر روی یک *LCD* نمایش دهد.

راهنمایی : در این پرسش، بایستی از تایمر به عنوان شمارنده (کانتر) استفاده شود. عملکرد این مدار فرکانس متر به این صورت است که توسط تایمر صفر، زمانی به مدت یک ثانیه تولید می شود و همزمان با آن، تایمر یک نیز شروع به شمارش تعداد پالس های اعمالی بر روی پایه *T1* می کند (در صورت استفاده از *ATmega16* منظور از پایه *T1* همان پایه *PB1* است). پس از طی زمان یک ثانیه، مقدار شمارش شده در رجیستر تایمر یک، معادل فرکانس موج روی پایه *T1* است و می توان آن را بر روی *LCD* نمایش داد. می توان از هر لبه پایین رونده مانند یک وقفه خارجی استفاده کرد.

آزمایشگاه ریزپردازنده ها

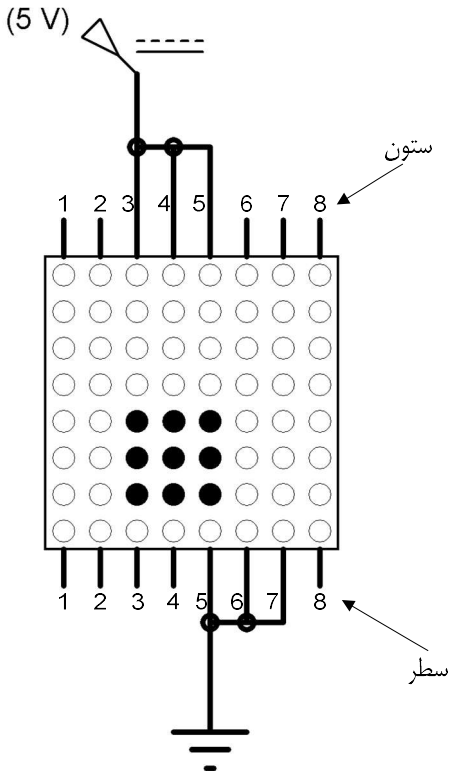
آزمایش شماره ۸

آشنایی با اصول اسکن *Dot Matrix* و نمایش اطلاعات بر روی آن

(تابلو روان)

مقدمه

از *Dot Matrix* برای ساخت تابلو روان استفاده می شود. برای نمایش تصویر بر روی *Dot Matrix*ها از اصول اسکن سطری پیروی می شود؛ به این ترتیب که هر *LED* را که بخواهیم روشن کنیم باید ستونی که آن *LED* در آن قرار دارد را یکِ منطقی (۵ ولت) و سطری که آن *LED* در آن قرار دارد را صفر (زمین) نماییم. یک نمونه از چنین ساختاری در شکل ۱ نشان داده شده است. می توان دید که با یک کردن ستون های ۳، ۴ و ۵ و صفر کردن سطرهای ۵، ۶ و ۷ یک دسته از *LED*ها در سطر و ستون های متناظر روشن شده اند.



شکل ۱ : نحوه اسکن شدن سطرها و ستون ها در *Dot Matrix*

شرح آزمایش

ـ قسمت اول :

می خواهیم برنامه ای را بنویسیم که توسط آن، حرف *A* بر روی یک *Dot Matrix* نمایش داده شود. این برنامه در ادامه آورده شده است.

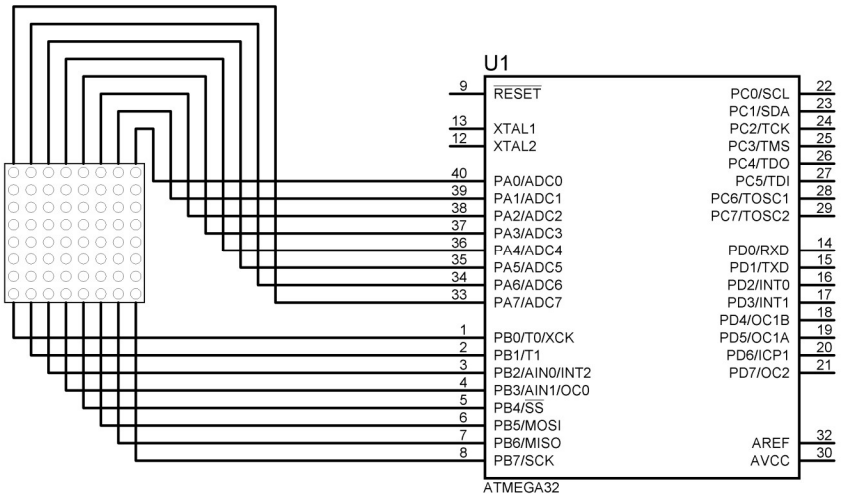
```
#include <mega32.h>
#include <delay.h>
#define xtal 8000000

unsigned char k;
flash unsigned char arr[8] = {0x18, 0x3C, 0x66, 0x66, 0x7E, 0x66, 0x66, 0x00};

void main(void)
{
    DDRA = 0xFF;
    DDRB = 0xFF;

    while(1){
        for(k=0;k<=7;k++){
            PORTA = arr[k];
            PORTB = ~(1<<k);
            delay_us(100);
            PORTB = 0xFF;
        }
    }
}
```

این برنامه را به دقت بررسی نمایید و با بستن مدار نشان داده شده در شکل ۲ طرز کار آن را مشاهده و درستی برنامه را تحقیق نمایید.



شکل ۲ : مدار قسمت اول آزمایش

ـ قسمت دوم :

با توجه به قسمت اول آزمایش، برنامه را به گونه ای تغییر دهید که بر روی *Dot Matrix* حروف *K* و *Z* و اعداد *3* و *6* و حروف فارسی آ و پ نمایش داده شود.

ـ قسمت سوم :

برنامه ای بنویسید که اعداد صفر تا ۹ (فارسی) را با یک فاصله زمانی دلخواه، بر روی یک *Dot Matrix* نمایش دهد.

پرسش :

۱ ـ به کمک چندین *Dot Matrix* در کنار یکدیگر، یک تابلو روان طراحی کنید که هر رشته دلخواه مانند نام خودتان را نمایش دهد. برای سادگی، رشته ها را انگلیسی در نظر بگیرید و برنامه آن را به گونه ای بنویسید که رشته ها از چپ به راست حرکت کنند.

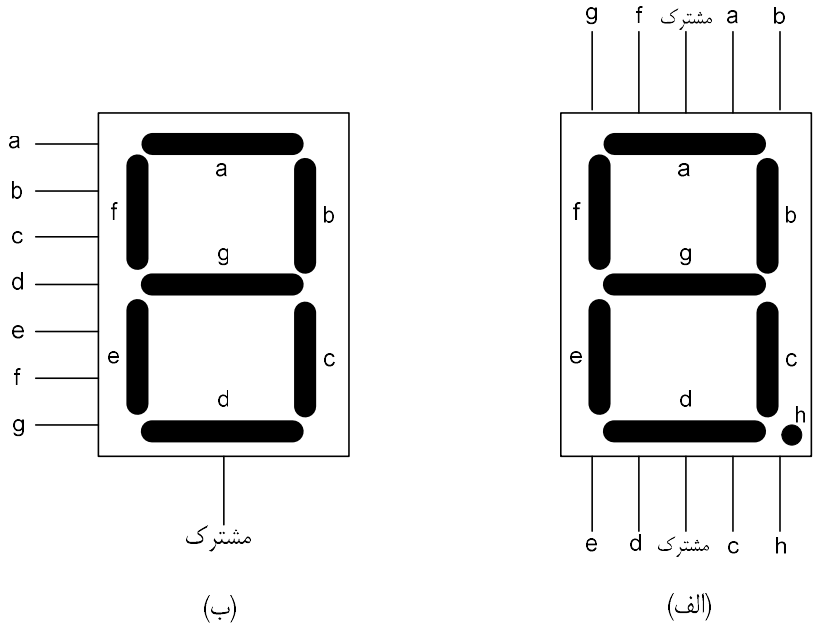
آزمایشگاه ریزپردازنده ها

آزمایش شماره ۹

پیاده سازی چراغ راهنمایی به کمک میکروکنترلر *AVR*

مقدمه

در این آزمایش، هدف، پیاده سازی یک چراغ راهنمایی به صورت ساده می باشد. برای نمایش اعداد از *7-segment* کاتد مشترک استفاده شده است. در این نوع *7-segment* پایه مشترک را به زمین متصل کرده و برای روشن کردن هر یک از *LED*های درون *7-segment* پایه متناظر با آن *LED* را به یکِ منطقی (۵ ولت) متصل می نماییم. ارتباط بین پایه ها و *LED* ها در شکل ۱ نشان داده شده است.



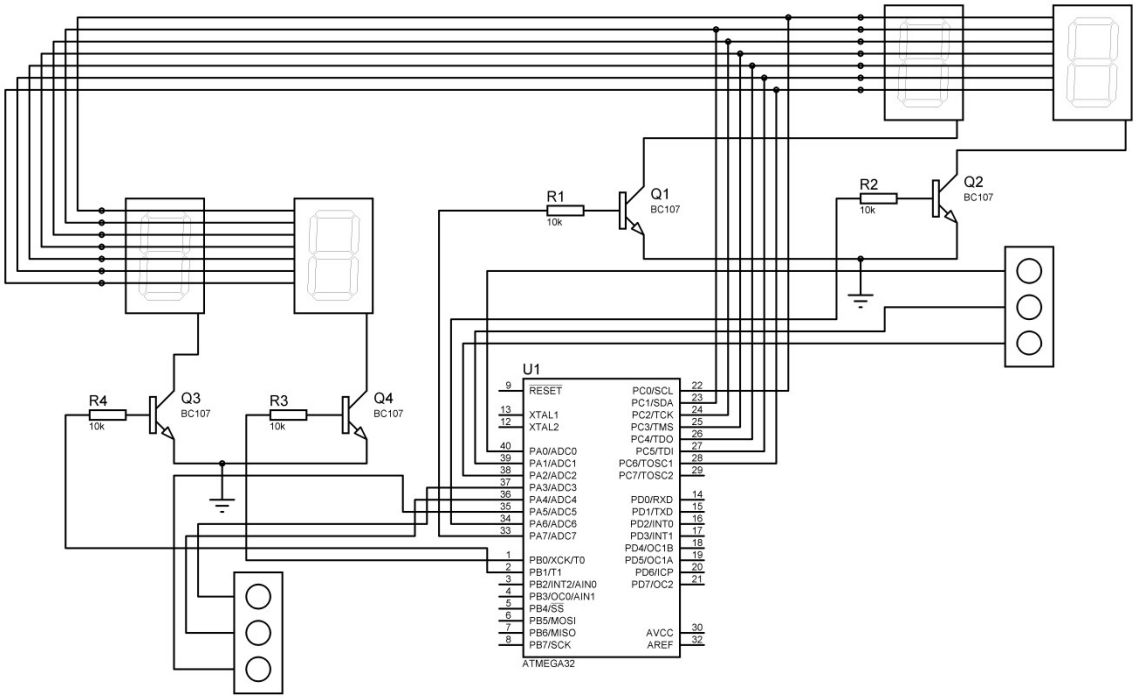
شکل ۱ : ارتباط بین پایه ها و *LED* ها در یک *7-segment*

شرح آزمایش

برای مدار نشان داده شده در شکل ۲ برنامه ای بنویسید که یک چراغ راهنمایی را پیاده سازی نماید؛ به این صورت که هر گاه *7-segment* با رنگ سبز از ۹ به صفر شمارش می کند، چراغ راهنمایی مربوط به آن، سبز باشد و *7-segment* با رنگ قرمز در کنار آن، خاموش باشد، همچنین در این حالت بایستی چراغ راهنمایی دیگر قرمز باشد و *7-segment* قرمز رنگ مربوط به آن چراغ راهنمایی از ۹ به صفر شمارش نموده و *7-segment* سبز خاموش باشد. پس از پایان شمارش بایستی جای آنها با یکدیگر عوض شود. البته برنامه را به گونه ای بنویسید که

با پایان یافتن شمارش (از ۹ به صفر)، برای یک مدت زمانی کوتاه (مثلاً ۲ ثانیه) *7-segment* های در حال شمارش در عدد صفر بمانند و چراغ راهنمایی که می خواهد به رنگ قرمز درآید، رنگ زرد را نشان دهد.

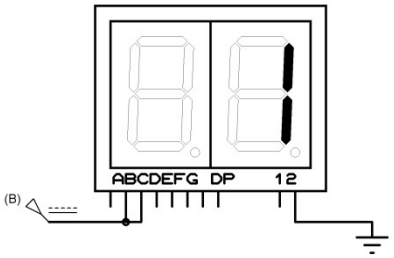
توجه : این مدار به گونه ای طراحی شده است که اگر *Base* هر یک از ترانزیستورهای *BC107* به منطق یک (۵ ولت) متصل شود، *7-segment* متصل به آن فعال می شود.



شکل ۲ : مدار به کار رفته برای پیاده سازی چراغ راهنمایی

پرسش :

۱ ـ برنامه این آزمایش را برای حالتی بنویسید که از *7-segment* های نشان داده شده در شکل ۳ استفاده می شود. بدیهی است که در این حالت می توان شمارش را برای اعداد بزرگتر از ۹ نیز انجام داد. به عنوان نمونه شمارش را از ۳۰ به صفر انجام دهید. این نوع *7-segment* نیز مشابه آنچه در این آزمایش به کار رفت، می باشد. برای فعال کردن *7-segment* سمت چپ باید پایه یک، *7-segment* سمت راست باید پایه ۲ و برای فعال کردن هر دو باید هر دو پایه ۱ و ۲ را به **زمین** متصل نمود. پایه های دیگر همانند یک *7-segment* کاتد مشترک معمولی است. البته نوع آند مشترک آن نیز وجود دارد که اتصال پایه های آن به ۵ ولت و زمین برعکس حالتی است که در بالا شرح داده شد.



شکل ۳ : *7-segment* به کار رفته برای پرسش