

برنامه نویسی

میکروکنترلر

AVR

به زبان C

جواد راستی

مقدمه

برای آشنایی با معماری و ساختمان داخلی یک میکروکنترلر، بهترین زبان برنامه نویسی زبان سطح پایین اسمبلی است. سطح این زبان از زبان ماشین بالاتر است و کاربر را از کار با کدهای پیچیده ماشین معاف می کند؛ اما پیاده سازی برنامه های مفصل و پیچیده به زبان اسمبلی کاری دشوار و وقتگیر است که نمونه آن را می توان در پیاده سازی شرطها مشاهده کرد.

مزایای استفاده از زبانهای سطح بالا برای برنامه نویسی میکروکنترلرها عبارتند از :

- سادگی و صرف زمان کمتر برای نوشتن برنامه های بزرگ و پیچیده
- سهولت تغییر و به روز کردن برنامه ها
- قابل فهم تر بودن برنامه های زبانهای سطح بالا نسبت به برنامه های زبان اسمبلی
- قابلیت استفاده از توابع از پیش آماده شده کامپایلر و کتابخانه های موجود
- قابلیت استفاده از کد نوشته شده برای یک میکروکنترلر در میکروکنترلرهای دیگر با

تغییرات ناچیز

امروزه برای برنامه نویسی میکروکنترلرها اغلب از زبانهای سطح بالا استفاده می شود که زبان C، یکی از زبان های متداول در این زمینه است. برنامه های زبانهای سطح بالا توسط کامپایلر به زبان ماشین قابل فهم میکروکنترلر تبدیل می شوند؛ هرچند تلاشهای زیادی برای بهینه کردن کامپایلرها در روش تبدیل دستورات سطح بالا به دستورات ماشین انجام می شود، اما فایل های زبان ماشین تولید شده توسط کامپایلرها نسبت به فایل های زبان ماشین تولید شده توسط اسمبلرها به نحو قابل ملاحظه ای بزرگتر و کندتر است. به بیان دیگر اگر برای انجام یک عملیات واحد، یک برنامه به زبان سطح بالا نوشته و توسط کامپایلر به زبان ماشین تبدیل کنیم و برنامه ای دیگر به زبان اسمبلی نوشته و با اسمبلر کد زبان ماشین معادل آن را به دست آوریم، حجم فایل زبان ماشین اول بین ۱/۵ تا دو برابر حجم فایل زبان ماشین دوم بوده و اجرای آن نیز به همین نسبت بیشتر به طول می انجامد.

در کل می توان گفت برای کاربردهای معمولی که حجم کد ماشین و زمان اجرای برنامه چندان مهم نیست، استفاده از زبان سطح بالا برای برنامه نویسی میکروکنترلر نسبت به زبان اسمبلی ساده تر، سریعتر و کاراتر است.

در این نوشته ابتدا به مقدمات برنامه نویسی میکروکنترلر AVR با زبان C خواهیم پرداخت و سپس با ارائه تعدادی مثال کاربردی، ویژگیهای مختلف زبان C را بررسی خواهیم کرد. برای مطالعه مطالبی که در ادامه می آید، آشنایی با ساختارهای زبان C ضروری است.

متداول ترین نرم افزار برای برنامه نویسی AVR به زبان C، نرم افزار CodeVision است که دارای کتابخانه های کاربردی برای راه اندازی دستگاههای جانبی مختلف می باشد و قسمت اعظم کدهای مورد نیاز یک پروژه را به صورت خود کار تولید می کند. برای برنامه نویسی های حرفه ای از این نرم افزار استفاده می شود؛ اما برای مقاصد آموزشی مناسب نیست؛ چون فاقد شبیه ساز و اشکالزدا می باشد و نیز به دلیل تولید خود کار کدها، کاربر را از یادگیری عمقی مفاهیم برنامه نویسی میکروکنترلر محروم می کند. محیط برنامه نویسی IAR نیز از نسخه های کارآمد، اما گران قیمت برنامه نویسی میکروکنترلر AVR است.

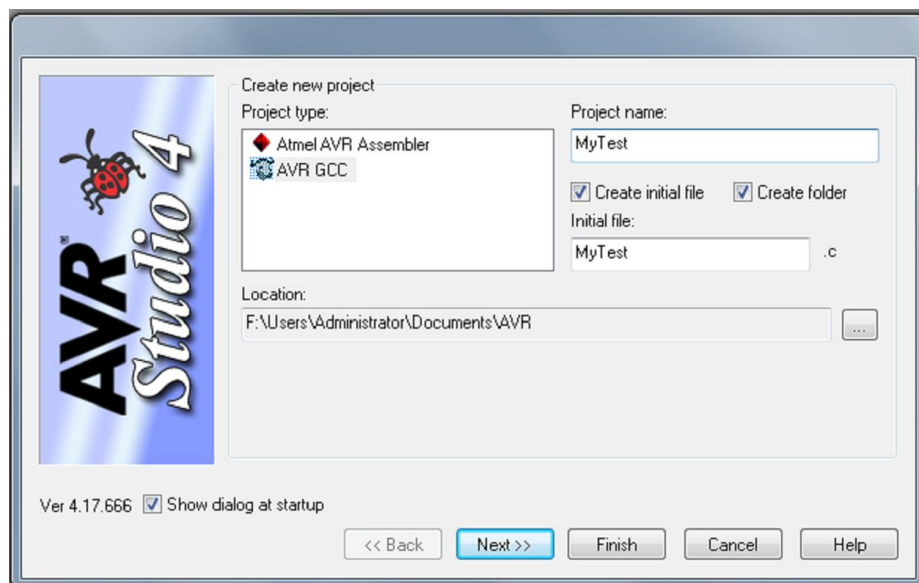
در این مطلب برای نوشتن برنامه های C، ترجمه، شبیه سازی و اشکالزدایی آنها از محیط مجتمع رایگان و کارآمد AVRStudio استفاده می کنیم. فراموش نکنید که برای این کار علاوه بر نرم افزار فوق باید WinAVR را نیز نصب کنید. استفاده از کتابخانه های avr-libc می تواند کمک خوبی در نوشتن پروژه های حرفه ای زبان C برای شما باشد. برای اطلاعات بیشتر به آدرس ذیل مراجعه کنید:

<http://www.nongnu.org/avr-libc/>

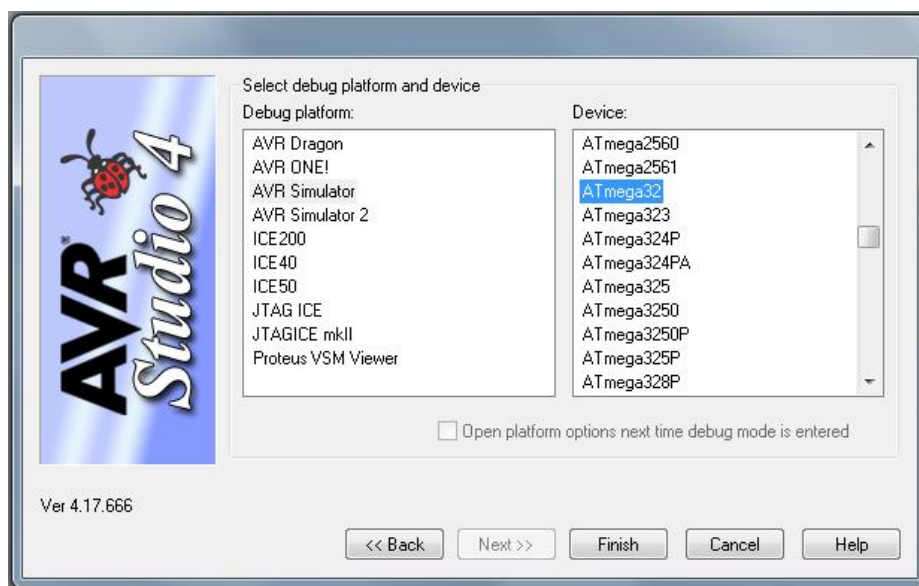
برنامه نویسی AVR به زبان C

زبان C میکروکنترلر AVR کاملاً شبیه به زبان C کامپیوتر است. البته هیچگاه فراموش نکنید که یک برنامه C در یک بستر کامپیوتری و یک برنامه C میکروکنترلر AVR در بستر یک میکروکنترلر اجرا می شود؛ استفاده از برخی تکنیکهای سطح بالا در برنامه نویسی C ممکن است منجر به ایجاد کدهای ماشین بزرگ با زمان اجرای طولانی شود. در این موارد بهتر است برنامه خود را هرچه ساده تر بنویسید تا کامپایلر بتواند کد ماشین کوتاهتری از آن بسازد.

برای ایجاد یک پروژه زبان C در AVRStudio، از منوی Project، گزینه New Project را انتخاب کنید. در کادری که ظاهر می شود، از قسمت نوع پروژه گزینه AVR GCC را انتخاب کنید و نام پروژه را در سمت راست وارد کنید:



با فشردن کلید Next، کادری ظاهر می شود که در آن نوع بستر اشکالزدایی و میکروکنترلر مورد استفاده را مشخص می کنیم:



با فشردن کلید Finish پروژه ایجاد شده و یک فایل خالی برای برنامه نویسی ایجاد می شود.

قالب برنامه های C

قالب کلی یک برنامه C به صورت زیر است :

```
#include <۱> نام فایل سرآیند
```

```
#include <۲> نام فایل سرآیند
```

```
...
```

اعلان متغیرهای عمومی

اعلان و تعریف توابع کاربر

```
int main(void){
```

اعلان متغیرهای محلی

دستورات برنامه (فراخوانی توابع کتابخانه ای و توابع کاربر)

```
return 0;
```

```
}
```

فایل‌های سرآیند^۱ که به همراه کامپایلر C ارایه می شوند (یا می توانند توسط کاربر نوشته شوند)، شامل تعریف توابعی هستند که برای ساده تر شدن برنامه نویسی در اختیار کاربر قرار داده شده است؛ مثلاً در فایل سرآیند io.h که در اکثر برنامه هایی که برای AVR نوشته می شود مورد استفاده قرار می گیرد، نام ثباتهای ورودی/خروجی، پورتهای، پرچمها و دیگر اجزای داخلی AVR و نیز توابعی برای کار کردن با آنها اعلان شده است؛ با شامل^۲ کردن این فایل در برنامه خود می توانید مستقیماً با این اجزا و توابع کار کنید.

انواع داده ها در C

اکثر انواع داده های زبان C در برنامه های C میکروکنترلر نیز قابل استفاده هستند؛ اما به دلیل اینکه AVR یک میکروکنترلر ۸ بیتی است و کامپایلر باید متغیرهای زبان C را به واحدهای ۸ بیتی

^۱ Header Files

^۲ include

تبدیل کند، برای تعریف متغیرهای برنامه خود سعی کنید از انواع داده زبان C به ترتیب زیر استفاده کنید تا حجم کد ماشین حاصل از برنامه های شما کوتاهتر شود:

انواع داده یک بایتی : unsigned char – (signed) char

انواع داده دوبایتی : unsigned int یا short int – (signed) int یا (signed) short

انواع داده ۴ بایتی : unsigned long – (signed) long – float

مثلاً برای تعریف شمارنده یک حلقه for که حداکثر آن ۲۰۰ است، به جای روش معمول در برنامه های C کامپیوتر که از داده های int استفاده می کند، از داده نوع unsigned char استفاده کنید.

مثالهایی از برنامه های C

اکنون تعدادی از مثالهای برنامه نویسی کاربردی به زبان C را بررسی می کنیم. توصیه می شود این مثالها که حاوی نکاتی مهم درباره برنامه نویسی به زبان C هستند را با دقت مطالعه کنید.

مثال برنامه ای بنویسید که دو عدد ۸ بیتی را بخواند، با هم جمع کند، حاصل را با عدد ۱۰ جمع کند و نتیجه را نمایش دهد.

در اینجا پورتهای PB و PC را ورودی و پورت PD را خروجی در نظر می گیریم.

```
#include <avr/io.h>
```

```
int main(void){
```

```
    DDRB = 0;           // Define PB as input.
    DDRC = 0;           // Define PC as input.
    DDRD = 0xFF;        // Define PD as output.
```

```
    while (1){
        PORTD = PINB + PINC + 10;
    }
    return 0;
```

```
}
```

در مورد این برنامه چند نکته قابل توجه است :

(۱) همانطور که گفته شد، فایل سرآیند io.h حاوی تعریف و نامگذاری ثباتهای داخلی AVR و توابع دستکاری آنهاست و شامل کردن آن در ابتدای برنامه ضروری است. در این فایل کلیه اجزای داخلی AVR با حروف **بزرگ** انگلیسی نامگذاری شده است. از آنجا که زبان C به بزرگی و کوچکی حروف حساس است، در نوشتن برنامه ها به این نکته دقت کنید.

(۲) در ابتدای برنامه پورتهای PB و PC به عنوان ورودی تعریف شده اند. در زبان C پیشوند **0x** قبل از یک عدد ثابت، نشانگر بیان آن عدد در دستگاه ۱۶ تایی است؛ اما اعداد دودویی مستقیماً قابل استفاده نیستند.

(۳) در برنامه های C، بعد از علامت // تا انتهای خط به عنوان توضیح در نظر گرفته می شود. از توضیحات برای واضح کردن قسمت های مختلف برنامه استفاده می کنیم.

(۴) حلقه (1) **while** به دلیل درست بودن دایمی شرط ادامه حلقه، بی نهایت مرتبه اجرا می شود. چنانچه بخواهیم این عملیات تنها یک مرتبه اجرا شود، برنامه باید به صورت زیر بازنویسی شود:

```
#include <avr/io.h>
```

```
int main(void){
    DDRB = 0;           // Define PB as input.
    DDRC = 0;           // Define PC as input.
    DDRD = 0xFF;        // Define PD as output.

    PORTD = PINB + PINC + 10;
    while (1);
    return 0;
}
```

پس از اجرای دستورات، برنامه در حلقه بینهایت (1) **while** به دام می افتد و باقی می ماند. این دستور مشابه خط فرمان **RJMP Here** است که برای قطع اجرای برنامه در انتهای برنامه های اسمبلی مورد استفاده قرار می گرفت.

برای ایجاد حلقه بینهایت، به جای دستور (1) **while**، می توان از دستور (2) **for** نیز استفاده کرد. توجه کنید که دستور متوقف کننده اجرای برنامه برای شبیه سازی برنامه در محیط هایی

که دارای برنامه ناظر (monitor) هستند (مانند AVRStudio) ضروری نیست. اما برای اجرای عملی برنامه روی میکروکنترلر باید حتماً آن را بنویسید.

نکته دیگر اینکه برنامه فوق ساده ترین نسخه ممکن برای عملیات مورد نظر است؛ اما چنانچه بخواهید آن را برای اشکالزدایی مناسب تر کنید، بهتر است به صورت زیر آن را بازنویسی کنید:

```
#include <avr/io.h>
```

```
int main(void){
```

```
    DDRB = 0;           // Define PB as input.
    DDRC = 0;           // Define PC as input.
    DDRD = 0xFF;        // Define PD as output.
```

```
    unsigned char a,b,c;
```

```
    while (1){
```

```
        a = PINB;
```

```
        b = PINC;
```

```
        c = a + b + 10;
```

```
        PORTD = c;
```

```
    }
```

```
    return 0;
```

```
}
```

مثال) برنامه ای بنویسید که مرتباً PB را بخواند و معکوس آن را در PC نمایش دهد.

```
#include <avr/io.h>
```

```
int main(void){
```

```
    DDRB = 0;
```

```
    DDRC = 255;
```

```
    while (1)
```

```
        PORTC = ~PINB;
```

```
    return 0;
```

```
}
```

علامت ~ باعث معکوس شدن بیت‌های P1 می شود. خوانا و ساده بودن این برنامه نسبت به برنامه اسمبلی با همین هدف در مثال بالا مشخص است.

مثال) برنامه ای بنویسید که دو عدد m و n را بخواند و حاصل 2m-n را در نمایش دهد.

عدد m را از پورت PD و عدد n را از پورت PB خوانده و حاصل مورد نظر را در پورت PC نمایش می دهیم.

```
#include <avr/io.h>
#define m PIND
#define n PINB
#define Out PORTC
```

```
int main(void){
    DDRD = DDRB = 0;
    DDRC = 255;
    while (1) {
        Out = 2*m - n;
    }
    return 0;
}
```

می بینید که با استفاده از رهنمود `#define` می توان پورتهای یا ثباتها را نامگذاری و به خواناتر شدن برنامه کمک نمود. به علاوه این کار باعث می شود بتوانید برنامه ای را که برای میکروکنترلر AVR نوشته شده است، به راحتی روی میکروکنترلرهای دیگر که از یک زبان شبیه به C حمایت می کنند، اجرا کنید؛ برای این کار تنها کافی است تعاریف `#define` را مطابق با سخت افزار آن میکروکنترلر تغییر دهید. مزیت دیگر استفاده از رهنمود فوق، این است که اگر تصمیم برنامه نویس راجع به پورت های ورودی و خروجی تغییر کرد، نیازی نیست کل برنامه را تغییر دهد و می تواند تنها با اصلاح این رهنمود تغییر مورد نظرش را اعمال کند.

مثال) برنامه ای بنویسید که بیت های پورت PC را یکی در میان مرتباً خاموش و روشن کند.

```
#include <avr/io.h>

void MyDelay(void){
    unsigned int i;
    for (i = 0; i < 50000; i++);
}

int main(void){
    DDRC = 255;
    PORTC = 0xAA;
    while (1) {
        MyDelay();
```

```

    PORTC = ~PORTC;
}
return 0;
}

```

در این برنامه از تابع MyDelay که شامل یک حلقه بدون دستور است، برای ایجاد تأخیر استفاده شده است (با اسیلوسکوپ یا به کمک نرم افزار شبیه ساز Proteus این تأخیر را اندازه بگیرید). البته در اکثر کامپایلرها برای اجرای صحیح این برنامه، باید سطح بهینه سازی (Optimization Level) کامپایلر را به صفر تغییر دهید؛ وگرنه کامپایلر حلقه for تأخیر را به جهت اینکه کاری انجام نمی دهد، به عنوان دستور زائد تشخیص داده و حذف می کند.

راه دیگر ایجاد تأخیر، استفاده از تایمرهاست که در ادامه خواهیم دید. به علاوه در فایل سرآیند delay.h توابع تأخیر _delay_ms و _delay_us برای ایجاد تأخیرهای برحسب میلی ثانیه و میکروثانیه تعریف شده اند.

مثال) برنامه ای بنویسید که مقادیر صفر تا ۱۰۰ را مرتباً با تأخیر زمانی ۵۰۰ میلی ثانیه به پورت PC ارسال کند.

```

#include <avr/io.h>
#include <util/delay.h>

int main(void){
    DDRC = 255;
    while (1) {
        unsigned char i;
        for (i = 0; i <= 100;i++){
            PORTC = i;
            _delay_ms(500);
        }
    }
    return 0;
}

```

مثال) برنامه ای بنویسید که مرتباً PB را بخواند و تا زمانی که صفر نشده، عدد ۱۰۰ را در PC بنویسد. با صفر شدن PB، عدد ۲۰۰ در PC نوشته شده و برنامه به پایان می رسد.

```

#include <avr/io.h>
int main(void){
    DDRB = 0;
    DDRC = 255;

```

```

while (PINB != 0) {
    PORTC = 100;
}
PORTC = 200;
while (1);
return 0;
}

```

مثال) برنامه ای بنویسید که مرتباً عددی بین صفر تا ۹ را از PB بخواند و آن را در نمایشگر 7-segment آند مشترک متصل به PC نمایش دهد. اگر عدد خوانده شده بزرگتر از ۹ باشد، نمایشگر باید خاموش شود.

```

#include <avr/io.h>
int main(void){
    unsigned char index,Seven_Seg_Codes[ ] =
    {0x03,0x9F,0x25,0x0D ,0x99, 0x49,0x41,0x1F,0x01,0x09};

    DDRB = 0;
    DDRC = 255;
    while (1) {
        if (PINB >= 10)
            PORTC = 0xFF; // Turn off the Common Anode 7-Seg
        else {
            index = PINB;
            PORTC = Seven_Seg_Codes[index];
        }
    }
    return 0;
}

```

فرض بر این است که پینهای a و b و ... و h نمایشگر هفت قسمتی، به ترتیب به پینهای PC7 و PC6 و ... و PC0 میکروکنترلر متصل شده‌اند.

کامپایلر برای ذخیره متغیرهای تعریف شده در برنامه مانند آرایه Seven_Seg_Codes از فضای RAM میکروکنترلر استفاده می‌کند. چون محتویات این آرایه در طول برنامه تغییر نمی‌کند، در بعضی کامپایلرها با تعریف آن به صورت const می‌توان آن را در حافظه ROM قرار داد.

پرسش) نشان دهید برنامه قبلی را به صورت ذیل هم می‌توان نوشت:

```
#include <avr/io.h>
int main(void){
    DDRB = 0;
    DDRC = 255;
    for (;;) {
        switch(PINB){
            case 0 : PORTC = 0x03;
                       break;
            case 1 : PORTC = 0x9F;
                       break;
            case 2 : PORTC = 0x25;
                       break;
            case 3 : PORTC = 0x0D;
                       break;
            case 4 : PORTC = 0x99;
                       break;
            case 5 : PORTC = 0x49;
                       break;
            case 6 : PORTC = 0x41;
                       break;
            case 7 : PORTC = 0x1F;
                       break;
            case 8 : PORTC = 0x01;
                       break;
            case 9 : PORTC = 0x09;
                       break;
            default : PORTC = 0xFF;
                       break;
        }
    }
    return 0;
}
```

مثال) برنامه ای بنویسید که زوج پورت PA-PB را با عدد ۱۶ بیتی 127A H جمع کند و حاصل را در زوج پورت PC-PD را نمایش دهد.

```
#include <avr/io.h>
```

```
int main(void){
```

```
    DDRA = DDRB = 0;    // Define as input
    DDRC = DDRD = 0xFF; // Define as output
```

```

for (;;) {
    unsigned char carry = 0;
    if (PINB + 0x7A > 255)
        carry = 1;
    PORTD = PINB + 0x7A;
    PORTC = PINA + 0x12 + carry;
}

return 0;
}

```

مثال) برنامه ای بنویسید که رشته *This is a message* را به PD ارسال کند.

```

#include <avr/io.h>
#include <util/delay.h>

```

```

int main(void) {
    DDRD = 255;
    const unsigned char Table[18] = "This is a message";
    unsigned char i;
    for (i = 0; i < 17; i++) {
        PORTD = Table[i];
        _delay_ms(500);
    }

    for (;;) {
        return 0;
    }
}

```

دقت کنید رشته ۱۷ کاراکتر دارد و با لحاظ کردن کاراکتر NULL انتهای رشته، طول آن ۱۸ کاراکتر خواهد بود. اما تنها ۱۷ کاراکتر اول آن باید به پورت D ارسال شود.

مثال) برنامه ای بنویسید که PB را بخواند و مجموع ارقام دهدهی آن را در PC بنویسد.

```

#include <avr/io.h>
int main(void) {
    DDRB = 0;
    DDRC = 255;
    while (1) {
        unsigned char n = PINB, sum = 0;
        sum = sum + n % 10; // or sum += n % 10;
        n = n / 10;
    }
}

```

```

sum = sum + n % 10;
sum = sum + n / 10;
PORTC = sum;
}
return 0;
}

```

این برنامه را از نظر خوانایی با نسخه اسمبلی آن مقایسه کنید.

برنامه نویسی بیتی

زبان C میکروکنترلر AVR دستورات بیتی ندارد و باید از دستورات دستکاری بیتی زبان C استاندارد و قواعد زیر برای برنامه نویسی بیتی استفاده کرد:

برای «یک» کردن بیت n ام یک عدد بدون تغییر بقیه بیت‌ها، آن عدد را با الگویی OR می‌کنیم (با استفاده از عملگر | در زبان C) که در آن الگو، بیت n ام «یک» و بقیه بیت‌ها «صفر» هستند. این الگو «ماسک یک» نامیده می‌شود.

برای «صفر» کردن بیت n ام یک عدد بدون تغییر بقیه بیت‌ها، آن عدد را با الگویی AND می‌کنیم (با استفاده از عملگر & در زبان C) که در آن الگو، بیت n ام «صفر» و بقیه بیت‌ها «یک» هستند. این الگو «ماسک صفر» نامیده می‌شود.

برای معکوس کردن بیت n ام یک عدد بدون تغییر بقیه بیت‌ها، آن عدد را با الگویی XOR می‌کنیم (با استفاده از عملگر ^ در زبان C) که در آن الگو، بیت n ام «یک» و بقیه بیت‌ها «صفر» هستند. این الگو «ماسک معکوس» نامیده می‌شود.

مثال برنامه‌ای بنویسید که LED متصل به PB4 را بدون تغییر بقیه بیت‌های این پورت خاموش و روشن کند.

```

#include <avr/io.h>

void MyDelay(void){
    unsigned int i;
    for (i = 0; i < 55000; i++);
}

int main(void){

```

```
DDRB = DDRB | 0x10;
```

```
while (1) {
    PORTB = PORTB | 0x10;
    MyDelay();
    PORTB = PORTB & 0xEF;
    MyDelay();
}
return 0;
}
```

برای «یک» کردن PB4، ثبات PORTB را با 00010000 یا 0x10، OR و برای «صفر» کردن این بیت، پورت مذکور را با 11101111 یا 0xEF، AND می کنیم.

پرسش نشان دهید حلقه اصلی برنامه را می توان با استفاده از ماسک تغییر به صورت زیر بازنویسی کرد:

```
while (1) {
    PORTB = PORTB ^ 0x10;
    MyDelay();
}
```

مثال برنامه ای بنویسید که یک رقص نور روی پورت PC ایجاد کند.

```
#include <avr/io.h>
#include <util/delay.h>
int main(void){
    DDRC = 0xFF;
    for (;;) {

        unsigned char i;
        unsigned char a = 1;
        for (i = 1; i <= 7; i++){
            PORTC = a;
            _delay_ms(500);
            a = a << 1;
        }
    }
    return 0;
}
```

عملگر << برای شیفت به چپ استفاده می شود. دستور $a = a << 1$ مقدار a را یک واحد به چپ شیفت داده و مقدار جدید را در متغیر a قرار می دهد.

پرسش) برنامه بالا را طوری اصلاح کنید که الف) جهت رقص نور را عوض کند ب) LED روشن دوتا دوتا جلو برود.

مثال) برنامه ای بنویسید که هرگاه کلید فشاری متصل به PB.0 فشرده («صفر») شود، یک واحد به محتویات PC اضافه کند.

```
#include <avr/io.h>
#include <util/delay.h>
```

```
void Debounce(void);
```

```
int main(void){
```

```
    DDRB = DDRB & 0xFE;
    DDRC = 0xFF;
```

```
    while (1){
        while (PINB & 0x01); // Wait until push button is pressed.
        Debounce();
        PORTC = PORTC + 1;
        while ((PINB & 0x01) == 0); // Wait until push button is released.
        Debounce();
    }
    return 0;
```

```
}
```

```
////////////////////////////////////
```

```
void Debounce(void){
    _delay_ms(100);
}
```

در ابتدا باید منتظر بمانیم تا کلید فشرده شود. سپس به محتویات PC یک واحد اضافه کرده و منتظر می مانیم تا کلید رها شود. بعد از تشخیص فشرده یا رها شدن کلید، به کمک تابع Debounce تأخیر کوتاهی ایجاد می کنیم تا لرزش های کلید از بین برود و برنامه را در تشخیص فشرده یا رها شدن کلید دچار اشتباه نکند.

پرسش) برنامه قبلی را طوری اصلاح کنید که هرگاه کلید فشاری متصل به PB.0 فشرده («صفر») شود، محتویات PC را در PD بنویسد.

مثال) برنامه‌ای بنویسید که حالت دو درب که به پین‌های صفر و یک پورت PB متصل هستند را کنترل کند؛ اگر یکی از آنها باز شود چراغ متصل به PC0 و اگر هر دو باز شود آژیر متصل به PC1 روشن شود. مدار تشخیص باز و بسته بودن درب در حالت باز بودن درب، سیگنال منطقی "یک" تولید می‌کند.

```
#include <avr/io.h>
```

```
int main(void){
```

```
    DDRB = DDRB & 0xFC;
```

```
    DDRC = DDRC | 0x03;
```

```
    while (1){
```

```
        if ((PINB & 0x03) == 0x03){
```

```
            PORTC = PORTC | 0x02;
```

```
            PORTC = PORTC & 0xFE;
```

```
        }
```

```
        else if (0 < (PINB & 0x03) && (PINB & 0x03) < 0x03){
```

```
            PORTC = PORTC | 0x01;
```

```
            PORTC = PORTC & 0xFD;
```

```
        }
```

```
        else if ((PINB & 0x03) == 0){
```

```
            PORTC = PORTC & 0xFC;
```

```
        }
```

```
    }
```

```
    return 0;
```

```
}
```

مثال) برنامه‌ای بنویسید که PC و PD را بخواند و اگر $PC > PD$ ، PB.6 را روشن و در غیر این صورت خاموش کند.

```
#include <avr/io.h>
```

```
int main(void){
```

```
    DDRC = DDRD = 0;
```

```
    DDRB = DDRB | 0x40;
```

```

    for (;;) {
        if (PINC > PIND)
            PORTB = PORTB | 0x40;
        else
            PORTB = PORTB & 0xBF;
    }
    return 0;
}

```

مثال) برنامه ای بنویسید که بیت‌های زوج و فرد PC را به ترتیب روشن و خاموش کند. تأخیر مابین روشن و خاموش شدن بیت‌ها با عددی که از PB خوانده می شود تنظیم می گردد. تاکنون فقط از توابع بدون پارامتر استفاده کرده ایم. این مثال از یک تابع پارامتردار استفاده می کند.

```

#include <avr/io.h>
#include <util/delay.h>

```

```

void MyDelay(unsigned char time); // 'time' is parameter of Delay
function.

```

```

int main(void){

```

```

    DDRC = 0xFF;
    DDRB = 0;

    while (1) {
        unsigned char t = PINB;
        PORTC = 0x55;
        MyDelay(t);
        PORTC = 0xAA;
        MyDelay(t);
    }
    return 0;
}

```

```

}
/////////////////////////////////////////////////////////////////
void MyDelay(unsigned char time){
    _delay_ms(50*time);
}

```

همانگونه که مشاهده می کنید، الگوی تابع MyDelay قبل و بدنه آن بعد از تابع main تعریف شده است تا برنامه ساخت یافته تر شود.

استفاده از دستورات اسمبلی در پروژه های C

گاهی برای کاهش طول کد ماشین یا زمان اجرای برنامه، در نقاط بحرانی یک نرم افزار سطح بالا از کدهای اسمبلی استفاده می کنیم. AVR GCC این قابلیت را دارد که در یک پروژه به صورت همزمان با فایل های C و اسمبلی کار کند. برای اطلاعات بیشتر به مراجع AVR GCC مراجعه کنید.

پرسشهای دوره ای

- (۱) برنامه ای بنویسید که مقادیر صفر تا ۲۵۵ را مرتباً به PB ارسال کند.
- (۲) برنامه ای بنویسید که یک رمز ۸ بیتی را بخواند و اگر برابر 10111001 بود، یک درب را باز کند و برنامه به پایان برسد. از یک کلید فشاری برای ورود رمز استفاده کنید. کاربر تا ۳ مرتبه فرصت دارد رمز ورودی را به درستی وارد کند، وگرنه یک آژیر به صدا در خواهد آمد.
- (۳) برنامه ای بنویسید که PC را بخواند و اگر بزرگتر از ۱۰۰ بود آن را به PA و در غیر این صورت به PB ارسال کند.
- (۴) الف) برنامه ای بنویسید که محتویات پورت PB را به صورت سریال با شروع از LSB روی PC5 ارسال کند.
- ب) برنامه ای بنویسید که بایتهای سریال ارسال شده با برنامه قسمت **الف** را روی پین PD.0 میکروکنترلر مقصد دریافت کند.
- (۵) برنامه ای بنویسید که توازن فرد پورت PD را روی بیت PA.7 نمایش دهد.
- (۶) برنامه ای بنویسید که محتویات پورت PB را به صورت دهدهی روی سه نمایشگر هفت قسمتی که به پورت PC متصل هستند، به صورت Refreshing نمایش دهد.
- (۷) برنامه ای بنویسید که حاصل PB / 8 را تا دو رقم اعشار روی سه نمایشگر هفت قسمتی که به پورت PC متصل هستند، به صورت Refreshing نمایش دهد.

