برنامه نویسی میکروکنترلر AVR به زبان C

جواد راستی

مقدمه

برای آشنایی با معماری و ساختمان داخلی یک میکروکنترلر، بهترین زبان برنامه‌نویسی 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 می‌توان 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 برنامه‌ها
- قابل فهم‌تر بودن برنامه های زبانهای سطح بالا نسبت به برنامه های زبان اسمبلی
- قابلیت استفاده از توابع از پیش آماده شده کامپایلر و کتابخانه‌های 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 زبان‌های 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 فایلهای زبان ماشین تولید شده توسط کامپایلرها نسبت به فایلهای زبان ماشین تولید شده توسط اسمبلرها به نحو قابل ملاحظه‌ای 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 می‌انجامد. در کل می‌توان 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 ساده‌تر، 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 می‌آید، آشنایی با ساختارهای زبان C ضروری است.

متداول‌ترین نرم‌افزار برای برنامه‌نویسی AVR به زبان C، نرم‌افزار CodeVision است که دارای کتابخانه‌های کاربردی برای راه‌اندازی دستگاههای جانبی مختلف می‌باشد و می‌تواند به کمک برنامه‌ساز CodeWizard، قسمت اعظم کدهای مورد نیاز یک پروژه را به صورت خودکار تولید کند. برای برنامه‌نویسی‌های حرفه‌ای از این نرم‌افزار استفاده می‌شود؛ اما فاقد شبیه‌ساز اجرا و اشکالزدا می‌باشد و از این نظر وابسته به نرم‌افزارهای

اشکالزدایی مانند AVRStudio است. محیط برنامه‌نویسی IAR نیز از نسخه‌های کارآمد، اما گران‌قیمت برنامه‌نویسی 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 نرم‌افزار فوق باید WinAVR را نیز نصب کنید. استفاده از کتابخانه‌های avr-libc می‌تواند کمک خوبی در نوشتن پروژه‌های حرفه‌ای زبان C برای شما باشد. برای اطلاعات بیشتر به آدرس ذیل مراجعه کنید:

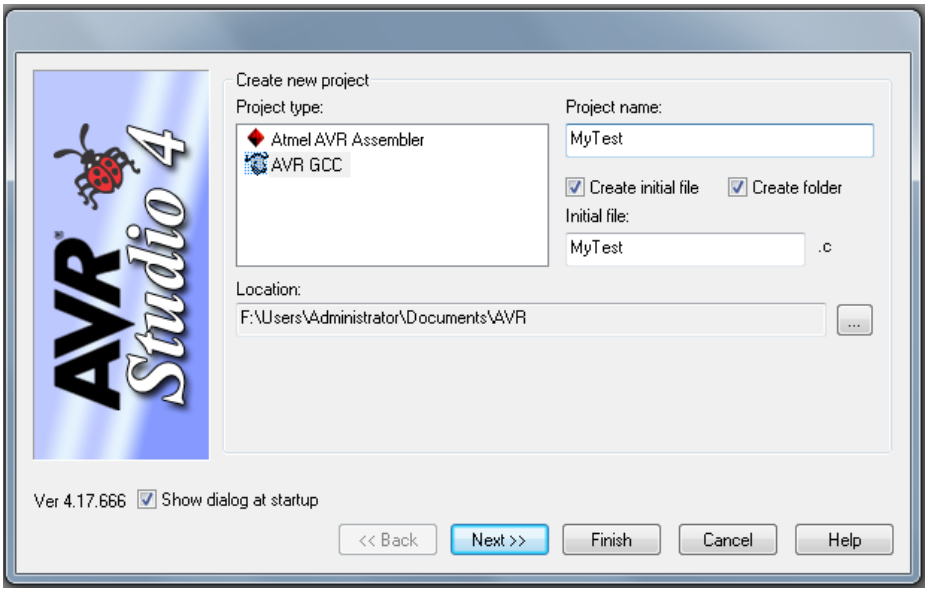
http://www.nongnu.org/avr-libc/

به علاوه در ادامه نحوه برنامه‌نویسی در Codevision و اتصال آن به AVRStudio برای اشکالزدایی را خواهیم آموخت.

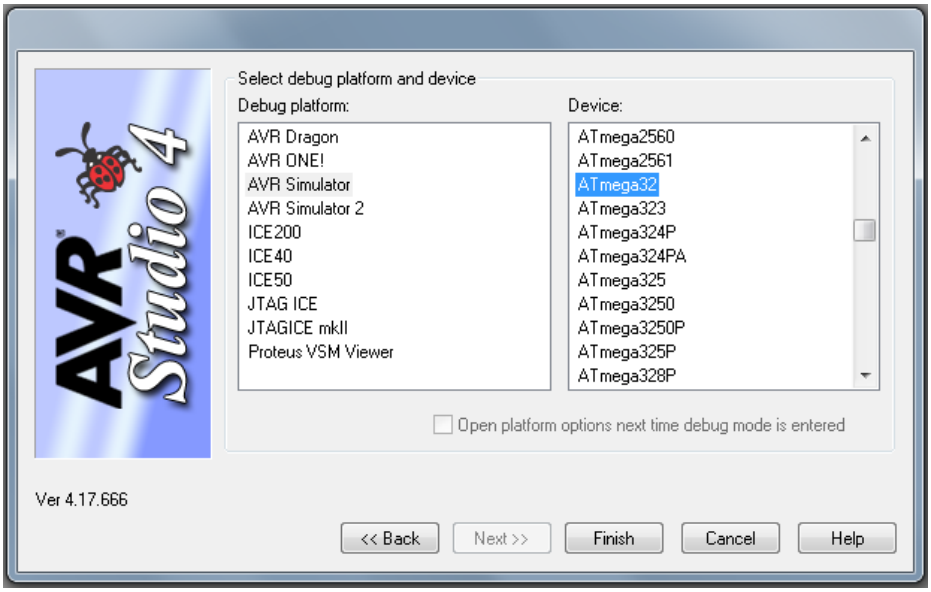
زبان C میکروکنترلر AVR کاملاً شبیه به زبان C کامپیوتر است. البته هیچگاه فراموش نکنید که یک برنامه C در یک بستر کامپیوتری و یک برنامه C میکروکنترلر AVR در بستر یک میکروکنترلر اجرا می‌شود؛ استفاده از برخی تکنیکهای سطح بالا در برنامه‌نویسی C ممکن است منجر به ایجاد کدهای ماشین بزرگ با زمان اجرای طولانی شود.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

برنامه‌نویسی و اشکالزدایی در محیط AVRStudio

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

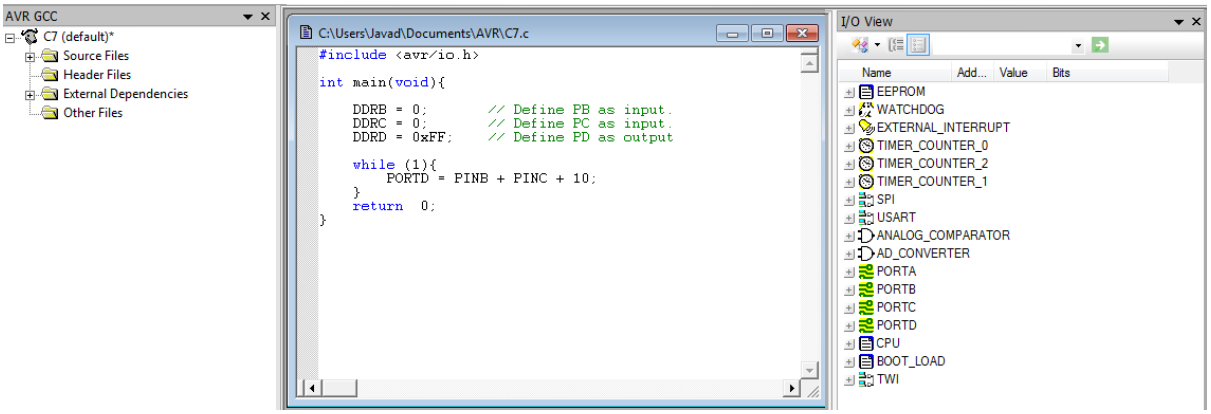


با فشردن کلید Next، کادری ظاهر می‌شود که در آن نوع بستر اشکالزدایی و میکروکنترلر مورد استفاده را مشخص می‌کنیم:

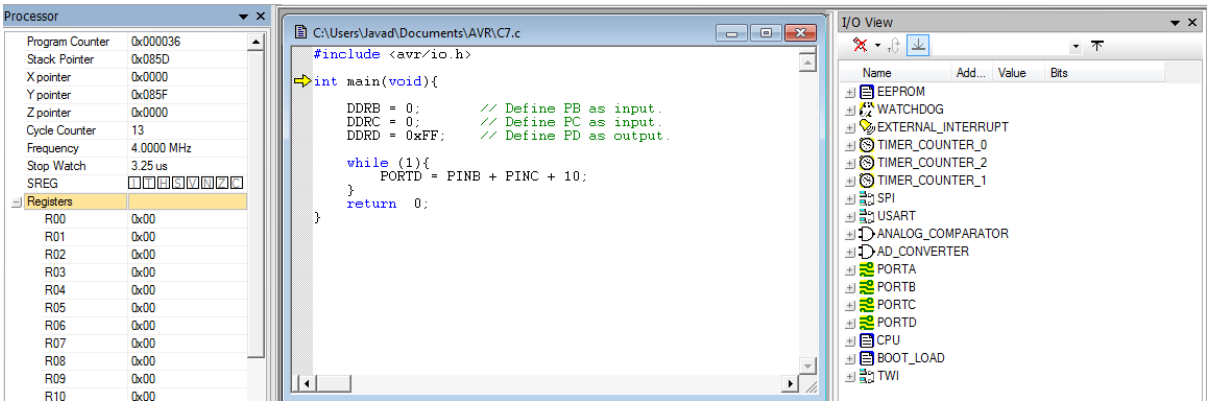


با فشردن کلید Finish پروژه ایجاد شده و یک فایل خالی برای برنامه‌نویسی ایجاد می‌شود.

اکنون برنامه زیر را در محیط ایجاد شده بنویسید.

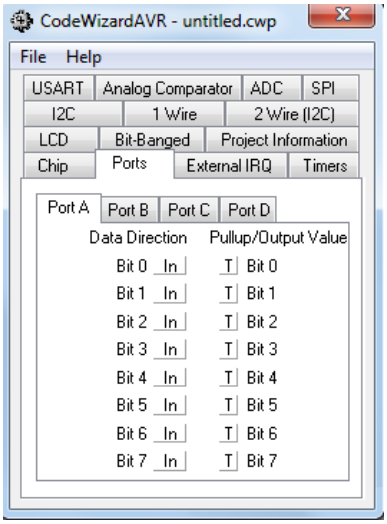


با انتخاب گزینه Build از منوی Build (یا فشردن کلید F7 یا انتخاب آیکون مربوطه از بالای صفحه)، برنامه را کامپایل کنید. در پنجره پایین صفحه، خطاهای و هشدارهای احتمالی برنامه شما دیده می‌شود. در صورتی که ترجمه با موفقیت انجام شده بود، با انتخاب گزینه Start Debugging از منوی Debug می‌توانید برنامه خود را خط‌به‌خط اجرا و نتیجه اجرا را به شکل زیر مشاهده کنید:



مقادیر ثباتها در کادر سمت چپ و مقادیر واحدهای ورودی/خروجی در کادر سمت راست دیده می‌شود. با انتخاب گزینه‌های دیگر منوی Debug می‌توانید به امکانات بیشتری برای اجرای مرحله‌به‌مرحله و اشکالزدایی برنامه خود دست یابید.

برنامه‌نویسی و اشکالزدایی در محیط CodeVision



برای ایجاد یک پروژه در محیط CodeVision، ابتدا از منوی فایل گزینه New و از کادری که ظاهر می‌شود گزینه Project را انتخاب کنید تا پروژه جدیدی گشوده شود. در کادر بعدی از شما پرسیده می‌شود آیا مایل به استفاده از برنامه‌ساز CodeWizard هستید یا خیر؟ استفاده از برنامه‌ساز تولید برنامه‌ها را بسیار ساده می‌کند. مثلاً با انتخاب مشخصات مورد نظرتان برای تنظیم بخش‌های مختلف میکروکنترلر، بسیاری از دستوراتی که باید توسط برنامه‌نویس نوشته شود، به صورت خودکار توسط برنامه‌ساز تولید می‌شود. عیب استفاده از CodeWizard تولید کدهای مفصل با بخش‌های بلااستفاده است که علاوه بر اشغال بی‌مورد فضای ROM میکروکنترلر، اشکالزدایی برنامه را نیز با مشکل مواجه می‌کند. اکنون که در ابتدای راه برنامه‌نویسی به زبان C برای میکروکنترلر هستیم، از استفاده از برنامه‌ساز CodeWizard صرفنظر می‌کنیم و برنامه‌ها را به صورت ساده می‌نویسیم. بنابراین از کادر اخیر گزینه No را انتخاب کنید. با تعیین نام پروژه در مرحله بعد، یک پروژه خالی ساخته شده و آماده است. در کادری که ظاهر می‌شود، زبانه C Compiler و از قسمت Chip گزینه ATmega32 را انتخاب کنید. در این کادر مشخصات دیگری از مدل برنامه‌نویسی میکروکنترلر نیز قابل تنظیم است.

اکنون مجدداً از منوی File گزینه New و این بار از کادری که ظاهر می‌شود گزینه Source را انتخاب کنید تا فایل اصلی برنامه شما ایجاد شود. کد زیر را در این فایل وارد و سپس آن را با یک نام دلخواه و پسوند .c ذخیره کنید:

```
#include <mega32.h>
void main(void){
	DDRD.0 = 1;
	while (1) {
		PORTD.0 = 0;
		PORTD.0 = 1;
	}
}
```

اکنون باید فایل برنامه ایجاد شده را وارد پروژه‌ای که در مرحله قبل ساختید بنمایید. برای این کار از منوی Project گزینه Configure را انتخاب کنید. در کادری که ظاهر می‌شود دکمه Add را بزنید و در لیستی که از فایل‌های برنامه نمایش داده می‌شود، نام فایل بالا را انتخاب کنید و سپس به تنظیمات پروژه پایان دهید (به اتفاقاتی که در کادر سمت چپ صفحه می‌افتد دقت کنید). اکنون از منوی Project گزینه Build All را انتخاب کنید. چنانچه در کادری که ظاهر می‌شود، خطا یا هشداری وجود نداشته باشد، یعنی پروژه و برنامه شما صحیح و قابل اجراست.

برای اشکالزدایی برنامه از منوی Setting گزینه Debugger را انتخاب کنید و در کادری که ظاهر می‌شود، آدرس نرم‌افزار AVRStudio را وارد کنید تا از این به بعد از این نرم‌افزار برای شبیه‌سازی و اشکالزدایی برنامه شما استفاده شود. با انتخاب گزینه Debugger از منوی Tools، نرم‌افزار AVRStudio اجرا می‌شود.

در نرم‌افزار AVRStudio از منوی File گزینه Open را انتخاب کنید. به مسیری که پروژه CodeVision شما در آن ذخیره شده بروید و فایلی که هم‌نام با پروژه شماست و پسوند cof. دارد را انتخاب کنید. اکنون AVRStudio ایجاد پروژه‌ای برای شبیه‌سازی برنامه شما را آغاز می‌کند که مراحل آن مانند آنچه در ابتدای این مطلب شرح داده شد، می‌باشد. در انتها برنامه شما قابل شبیه‌سازی و اشکالزدایی است.

در ادامه به مدل برنامه‌نویسی C در نرم‌افزارهای AVRStudio و CodeVision خواهیم پرداخت.

قالب برنامه های C

قالب کلی یک برنامه C به صورت زیر است :

```
#include <نام فایل سرآیند۱.h>
#include <نام فایل سرآیند۲.h>
…
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[1] که به همراه کامپایلر C ارایه می شوند (یا می‌توانند 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 AVRStudio نوشته می‌شود مورد استفاده قرار می‌گیرد، نام ثباتهای ورودی/خروجی، پورتها، پرچمها و دیگر اجزا داخلی AVR و نیز توابعی برای کار کردن با آنها اعلان شده است؛ با شامل[2] کردن این فایل در برنامه خود می‌توانید مستقیماً با این اجزا و توابع کار کنید.

انواع داده‌ها در C

اکثر انواع داده‌های زبان C در برنامه‌های 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 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 : (signed) char – unsigned char
انواع داده دوبایتی : unsigned int یا (signed) int – short int یا (signed) short
انواع داده ۴ بایتی : float – (signed) long – unsigned long

مثلاً برای تعریف شمارنده یک حلقه for که حداکثر آن ۲۰۰ است، به جای روش معمول در برنامه‌های C کامپیوتر که از داده های int استفاده می کند، از داده نوع unsigned char استفاده کنید.
نرم‌افزار CodeVision علاوه بر داده‌های استاندارد C، می‌تواند از انواع داده‌های دیگری نیز استفاده کند که در انتهای این مطلب به آنها خواهیم پرداخت.

مثالهایی از برنامه های C

اکنون تعدادی از مثالهای برنامه نویسی کاربردی به زبان C را بررسی می‌کنیم. برای هر مثال در ابتدا برنامه AVRStudio و سپس در مواقع لزوم برنامه CodeVision نوشته شده است تا با مشخصات برنامه‌نویسی در هر دو محیط آشنا شوید. توصیه می شود این مثالها که حاوی نکاتی مهم درباره برنامه نویسی به زبان C هستند را با دقت مطالعه کنید.

---
[1] ***Header Files***
[2] ***include***

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 ۱۶تایی 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 می‌شود. از توضیحات برای واضح کردن قسمتهای مختلف برنامه استفاده می کنیم.

۴) حلقه while (1) به دلیل درست بودن دایمی شرط ادامه حلقه، بی نهایت مرتبه اجرا می‌شود. 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 `while (1);` به دام می افتد و باقی می‌ماند. این دستور مشابه خط فرمان Here: 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 `while(1)`، می توان از دستور `for(;;)` نیز استفاده کرد. توجه کنید که دستور متوقف‌کننده اجرای برنامه برای شبیه‌سازی برنامه در محیط‌هایی 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 ساده‌ترین نسخه ممکن برای عملیات مورد نظر است؛ اما چنانچه بخواهید آن را برای اشکالزدایی مناسب‌تر 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;
    }
```

برنامه CodeVision این مثال به صورت زیر نوشته می‌شود:

```
#include <mega32.h>

void 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;
    }
}
```

تفاوتهای عمده یکی در نام فایلهای سرآیند و دیگری در نوع خروجی تابع main است که در محیط CodeVision باید از نوع void تعریف و دستور return 0; حذف شود.

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.

برنامه CodeVision با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

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 می‌دهیم.

```
#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 می‌توان پورتها یا ثباتها را نامگذاری و به خواناتر شدن برنامه کمک نمود. به‌علاوه این کار باعث می‌شود بتوانید برنامه‌ای را که برای میکروکنترلر AVR نوشته شده است، به راحتی روی میکروکنترلرهای دیگر که از یک زبان شبیه به C حمایت می‌کنند، اجرا کنید؛ برای این کار تنها کافی است تعاریف #define را مطابق با سخت‌افزار آن میکروکنترلر تغییر دهید. مزیت دیگر استفاده از رهنمود فوق، این است که اگر تصمیم برنامه‌نویس راجع به پورت‌های ورودی و خروجی تغییر کرد، نیازی نیست کل برنامه را تغییر دهد و می‌تواند تنها با اصلاح این رهنمود تغییر مورد نظرش را اعمال کند.

برنامه CodeVision با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

مثال) برنامه ای بنویسید که بیتهای پورت PC را یکی‌درمیان 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 نرم‌افزار شبیه‌ساز Proteus این تأخیر را اندازه بگیرید). البته در اکثر کامپایلرها برای اجرای صحیح این برنامه، باید سطح بهینه‌سازی (Optimization Level) کامپایلر را به صفر تغییر دهید؛ وگرنه کامپایلر حلقه for تأخیر را به جهت اینکه کاری انجام نمی‌دهد، به عنوان دستور زائد تشخیص داده و حذف می‌کند.

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 میلی‌ثانیه و میکروثانیه تعریف شده‌اند. در محیط CodeVision نیز همین فایل سرآیند و توابع وجود دارند؛ فقط کاراکتر _ درابتدای نام تابع حذف شده است.

برنامه CodeVision با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

مثال) برنامه‌ای بنویسید که مقادیر صفر تا ۱۰۰ را مرتباً با تأخیر زمانی ۵۰۰ میلی‌ثانیه 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;
}
```

برنامه محیط CodeVision به صورت زیر است:

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


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

مثال) برنامه‌ای 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 می‌رسد.

```
#include <avr/io.h>
int main(void){
	DDRB = 0;
	DDRC = 255;
	while (PINB != 0) {
		PORTC = 100;
	}
	PORTC = 200;
	while (1);
	return 0;
}
```

برنامه CodeVision با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

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 هفت‌قسمتی، 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 (یا flash در CodeVision) می‌توان آن را در حافظه ROM قرار داد.

برنامه CodeVision با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

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;
}
```

برنامه CodeVision با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

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.

برنامه CodeVision با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

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 CodeVision با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

برنامه‌نویسی بیتی

زبان C محیط AVRStudio دستورات بیتی ندارد و باید از دستورات دستکاری بیتی زبان C استاندارد و قواعد زیر برای برنامه‌نویسی بیتی استفاده کرد:

برای «یک» کردن بیت iام 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 iام «یک» و بقیه بیت‌ها «صفر» هستند. این الگو «ماسک یک» نامیده می‌شود.

برای «صفر» کردن بیت iام 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 iام «صفر» و بقیه بیت‌ها «یک» هستند. این الگو «ماسک صفر» نامیده می‌شود.

برای معکوس کردن بیت iام 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 iام «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 می‌کنیم.

محیط CodeVision دستورات بیتی دارد و به همین لحاظ نوشتن این گونه برنامه‌ها در این محیط بسیار ساده است. برنامه زیر را ببینید:

```
#include <mega32.h>

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

void main(void){

        DDRB.4  = 1;

        while (1) {
            PORTB.4 = 1;
               MyDelay();
               PORTB.4 = 0;
               MyDelay();
        }
}
```

توجه کنید دستورات بیتی CodeVision برای آدرس‌دهی بیتی ثباتهای I/O در بازه آدرسی صفر تا 0x1F معتبر هستند.

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

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

می‌توان از دستور PORTB.4 = ~PORTB.4; هم در محیط CodeVision استفاده کرد.

مثال) برنامه‌ای 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 می‌شود. دستور a = a << 1 مقدار a را یک واحد به چپ شیفت داده و مقدار جدید را در متغیر a قرار می‌دهد.

برنامه CodeVision با لحاظ تفاوتهای قبلی مانند برنامه بالاست و تفاوت دیگری ندارد.

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

مثال) برنامه‌ای 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 می‌مانیم تا کلید رها شود. بعد از تشخیص فشرده یا رها شدن کلید، به کمک تابع Debounce تأخیر کوتاهی ایجاد می‌کنیم تا لرزش‌های کلید از بین برود و برنامه را در تشخیص فشرده یا رها شدن کلید دچار اشتباه نکند.
برنامه CodeVision را خودتان بنویسید.

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 می کند.

```
#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;
}
```

برنامه CodeVision را خودتان بنویسید.

مثال) برنامه ای بنویسید که PC و PD را بخواند و اگر PD > PC، 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;
}
```

برنامه CodeVision را خودتان بنویسید.

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 ساخت‌یافته‌تر شود.

برنامه CodeVision را خودتان بنویسید.

استفاده از دستورات اسمبلی در پروژه‌های C

گاهی برای کاهش طول کد ماشین یا زمان اجرای برنامه، در نقاط بحرانی یک نرم افزار سطح بالا از کدهای اسمبلی استفاده می‌کنیم. AVR GCC این قابلیت را دارد که در یک پروژه به صورت همزمان با فایلهای c و اسمبلی کار کند. برای اطلاعات بیشتر به مراجع AVR GCC مراجعه کنید. در CodeVision می‌توانید دستورات اسمبلی را بین عبارات #asm و #endasm بنویسید.

تعیین محل ذخیره داده‌ها در CodeVision

در CodeVision می‌توان محل ذخیره انواع داده‌ها را تعیین کرد. با تعریف متغیرها به صورت عادی، فضای مورد نیاز آنها از حافظه‌ی SRAM تأمین می‌شود. مثال زیر یک متغیر عمومی نوع int را در خانه‌ی شماره 80h حافظه SRAM ذخیره می‌کند:

```
int x @0x80;
```

با تعریف داده به صورت register به کامپایلر فرمان داده می‌شود تا متغیر را در ثبات‌های میکروکنترلر ذخیره کند. البته حتی در صورت استفاده نکردن از این عبارت هم کامپایلر ممکن است این کار را انجام دهد. اگر از کلمه volatile قبل از تعریف داده استفاده کنید، آن متغیر در ثبات‌ها ذخیره نخواهد شد.

با تعریف متغیر به صورت const یا flash، داده مزبور به صورت ثابت در حافظه ROM و با تعریف متغیر به صورت eeprom، داده مزبور در حافظه EEPROM میکروکنترلر ذخیره خواهد شد.

اگر یک متغیر از نوع bit تعریف شود، در یکی از بیتهای ثباتهای R2 تا R14 ذخیره می‌شود و فقط می‌تواند مقادیر صفر یا یک را بپذیرد:

```
bit  x = 1;
```

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) برنامه‌ای 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) برنامه‌ای بنویسید که توازن فرد پورت PD را روی بیت PA.7 نمایش دهد.

۶) برنامه‌ای بنویسید که محتویات پورت PB را به صورت دهدهی روی سه نمایشگر هفت‌قسمتی که به پورت PC متصل هستند، به صورت Refreshing نمایش دهد.

۷) برنامه‌ای بنویسید که حاصل PB / 8 را تا دو رقم اعشار روی سه نمایشگر هفت‌قسمتی که به پورت PC متصل هستند، به صورت Refreshing نمایش دهد.