www.mohandesyar.com

عنـــــــوان

Subject:
Year. 90 Month. 12 Date. 10 ( )

اصول میکروکامپیوتر

سرفصل ها درس:

1- اصول کارکرد یک پردازنده

2- بررسی میکروکنترلرهای AVR (MEGA32)

3- برنامه نویسی به زبان C

4- بررسی و استفاده از امکانات AVR
- پورت های I/O
- اتصال LCD
- تایمر/کانتر
- ارتباط سریال
- ADC (A/D)

5- برنامه ریزی میکروکنترلرهای AVR به زبان Basic

مراجع:

1. The AVR MicroControllers And Embeded Systems Using Assembly And C
M. Reza Mazidi, Printcehall 2010

2. مرجع کامل میکروکنترلرهای AVR، پرتوی فر-مظاهریان-پیانلو، انتشارات آتش 1388

3. میکروکنترلرهای AVR به زبان C، حمید بادامی نژاد، انتشارات ادبستان

ارزیابی: پایانترم 60٪ میانترم 30٪ تمارین 10٪

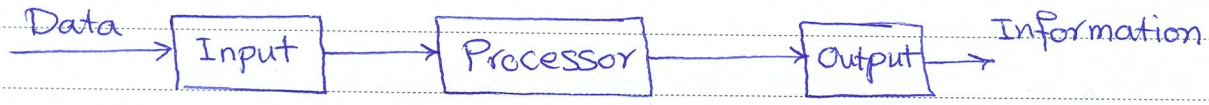
PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

فصل اول:

- سیستم کنترلی مبتنی بر پردازنده دارای سه قسمت اصلی زیر است:

1. ورودی: دریافت داده از محیط خارج
2. پردازنده: پردازش داده های ورودی
3. خروجی: تولید خروجی مطلوب



- مزیت پردازنده ها مجتمع بر مدارات منطقی دیجیتالی:

1. سادگی: حجم کم تر و سهولت در استفاده از آنها
2. قابلیت برنامه ریزی: تمام دستورات ورودی و خروجی و پردازشی در قالب دستورات نرم افزاری (برنامه) به پردازنده داده می شود.

- اجزای پردازنده:

1. واحد پردازش مرکزی CPU
2. حافظه
3. ورودی - خروجی

- برنامه ذخیره شده در حافظه توسط واحد CPU خط به خط اجرا می شود و خروجی مطلوب را ایجاد می کند.

- اتصال قسمت های مختلف یک پردازنده توسط Bus ها انجام می شود. (Data Bus - Address Bus)

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

- عملکرد پردازنده:

پردازش اطلاعات ذخیره شده در حافظه به صورت زیر انجام می شود:

1. خواندن دستورات از حافظه (واکشی - Fetch)
2. دکد کردن دستورات (Decode)
3. اجرا کردن دستورات (Execute)

- زبان پردازنده:

زبان قابل درک برای یک پردازنده، زبان صفر و یک (دودویی، زبان ماشین) است. بنابراین کوچک ترین واحد اطلاعات در این زبان یک بیت (صفر یا یک) است و اطلاعات به صورت رشته ای از بیت های صفر و یک است.

مثلاً برای خواندن دو عدد از ورودی و محاسبه مجموع (با زبان صفر و یک):

خواندن یک عدد : 00110101
خواندن یک عدد : 00110101
جمع کردن : 10010001
خروجی : 01000000

- به دلیل دشوار بودن این زبان، زبان اسمبلی معرفی شد.

- زبان اسمبلی: برای هر دستور معادل انگلیسی در نظر گرفته شده است:

- In ...
- In ...
- ADD...
- Out ...

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

- زبان اسمبلی، زبان وابسته به ماشین است. یعنی کدهای نوشته شده برای یک عملیات خاص در پردازنده های مختلف، متفاوت است. برای مثال دستور پاک کردن صفحه نمایش در زبان Basic، cls و یا در زبان C، دستور clrscr است اما در زبان اسمبلی به صورت زیر است:

```
MOV AH, 25
MOV AL, --
:
:
MOV DX, Ø
```

- برنامه ای که به زبان اسمبلی نوشته شده، توسط نرم افزاری که اسمبلر نامیده می شود به زبان صفر و یک تبدیل می شود. اگر برنامه به زبان C یا Basic باشد به این نرم افزار کامپایلر گویند.

- برنامه نویسی به زبانهای سطح بالا مثل C و Basic ساده تر است اما برنامه های نوشته شده به زبان اسمبلی حجم کم تری دارند و سرعت بالاتری را فراهم می کنند.

PAPCO
www.mohandesyar.com

Subject: جلسه دوم
Year. Month. Date. ( )

اجزاء پردازنده: (ALU , CU , Registers)

اجزاء پردازنده: Register (پرچم، انباره، شمارنده)، ALU، CU

۱. ثبات (Register): حافظه کوچکی هستند که در داخل پردازنده قرار دارد. تعداد رجیسترها و چندبیتی بودن آنها کارایی پردازنده را مشخص می کند.

الف) رجیستر انباره (Accumullator): رجیستر مهم منظوره است که برای ذخیره سازی عملیات ریاضی و منطقی استفاده می شود.

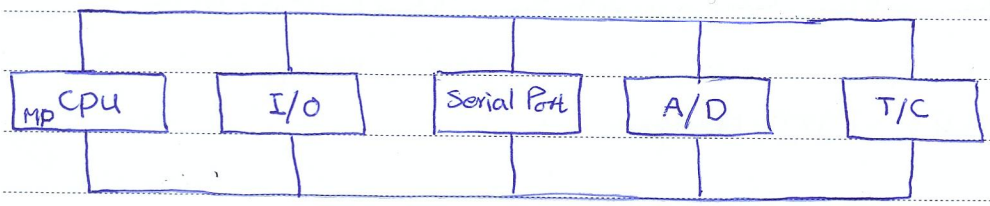
ب) رجیستر پرچم (Flag Reg.): بیت های رجیستر تغییر وضعیت در حین پردازش را نشان می دهد. مانند پرچم (ZF) اگر نتیجه محاسباتی صفر شود این پرچم تغییر وضعیت داده و مقدار آن یک می شود. (ZF=1)
مثال های دیگری که می توان برای بیت های پرچم استفاده کرد: ZF و OVF , Sign Flag.

ج) رجیستر شمارنده: Instruction Pointer Or Program Counter (IP) or (PC)
رجیستری است که برای شمارش خطوط برنامه استفاده می شود، با توجه به اینکه هر برنامه خط به خط اجرا می شود، مقدار رجیستر شمارنده با رسیدن به هر دستور یک واحد اضافه می شود.

۲. ALU: عملیات منطقی و ریاضی در این واحد انجام می شود.

3. CU: کنترل کننده سایر واحدها است و عملیات decode کردن دستورات در این واحد انجام می شود.

میکروپروسسور:



معایب استفاده از میکروپروسسور:
1. افزایش حجم
2. افزایش توان مصرفی
3. افزایش هزینه

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

میکروکنترلر:

تقطع یا چیپی است که امکانات جانبی مانند حافظه ها و تایمرکانترها و ... را به همراه پردازنده در خود جای داده است.

انواع میکروکنترلرهای AVR:

جزء اولین خانواده های AVR :
- AT Tiny
- AT 90S
- AT Mega

} 8 بیتی

- XMega } 16 بیتی

- به علت اینکه معماری استفاده شده در AVR از نوع معماری RISC می باشد، سرعت پردازش آن نسبت به سایر میکروکنترلرها بالاتر است. معیار اندازه گیری سرعت در این میکرو MIPS می باشد که میلیونها دستورالعمل را در یک ثانیه انجام می دهد. (هر دستور در یک پالس ساعت انجام می شود.)

RISC : Reduce Instruction Set Computer ~~Per Seconde~~

MIPS : Million " Per Second

- دستورات استفاده شده در مدل های پایین تر با کمی تغییر در مدل های بالاتر هم قابل اجرا باشد.
- میکروکنترلرهای AVR از لحاظ ولتاژ کاری در دو نوع L و معمولی موجودند که در مدل L ولتاژ کاری در محدوده $2.7^V - 5.5^V$ و در مدل معمولی محدوده ولتاژ $4.5^V - 5.5^V$ است.

2.7 - 5.5

PAPCO

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

میکروکنترلر ATMEGA32 :

امکانات میکروکنترلر ATMEGA32 :

1- 4 پورت ورودی و خروجی : پورت ها رابط دنیای بیرون و میکرو باشد که می تواند خروجی یا ورودی باشد.

2- تایمر و کانتر : T/C0 و T/C1 که 8 بیتی هستند و T/C2 ، 16 بیتی می باشد. → 8 bit → 0, 1 ; → 16 bit → 2

نکته: منابع تولید کلاک در میکرو ، اسیلاتور داخلی است که مقادیر 1, 2, 4, 8 MHz را دارا است و برای سرعت های بالاتر از کریستال خارجی 16 MHz در پایه 12 و 13 استفاده می شود. سرعت پردازش متناسب با کریستال (فرکانس کاری) است.

3- ارتباط سریال : (RXD , TXD)

4- منابع وقفه : در هنگام وقوع وقفه، میکرو عملیات را متوقف و زیر برنامه مربوط به وقفه را اجرا می کند.
هر یک امکانات میکرو وقفه مربوط به خود را دارد مانند وقفه T/C و A/D و ...

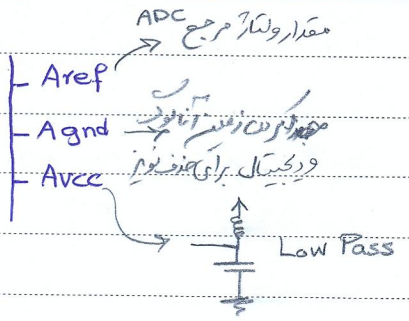
5- مولد موج PWM : (OCR1a , OCR1b)

Pulse Width Modulation

PAPCO

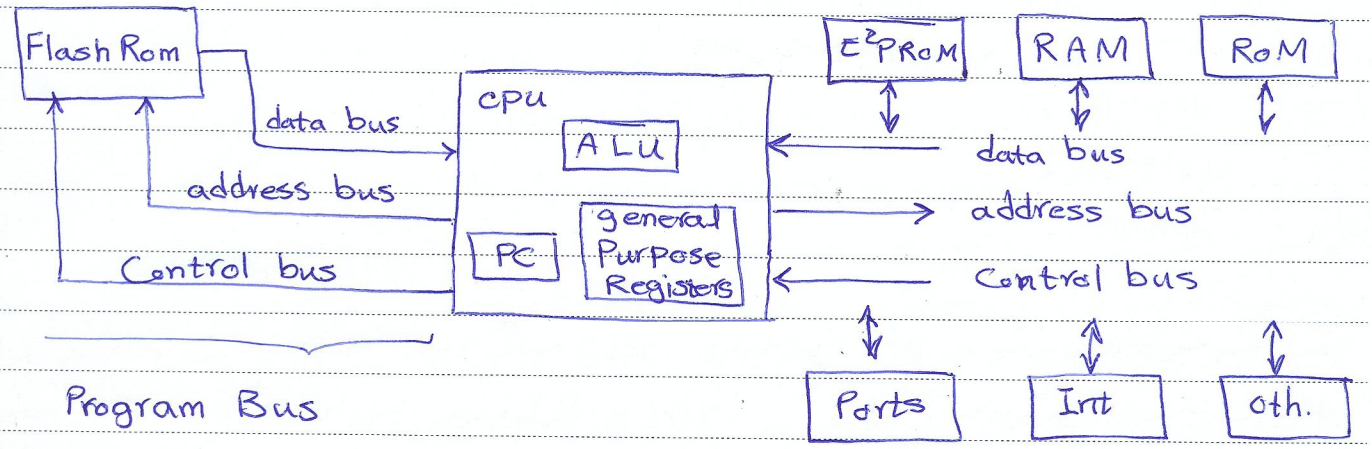
www.mohandesyar.com

Subject:
Year. 1391 Month. 1 Date. 15 ( )



6 ـ مبدل آنالوگ به دیجیتال: A/D 8 کانال

جلسه سوم: معماری میکروکنترلر:



انتقال و پردازش اطلاعات در هر پردازنده از طریق ارتباط گذرگاه‌ها با cpu انجام می‌شود، انواع گذرگاه‌هایی که در هر پردازنده وجود دارند، عبارتند از:

1ـ گذرگاه داده: تمامی اطلاعاتی که در سیستم پردازش می‌شود از باس داده عبور می‌کند. (برقرارکننده ارتباط میان پردازنده و حافظه)

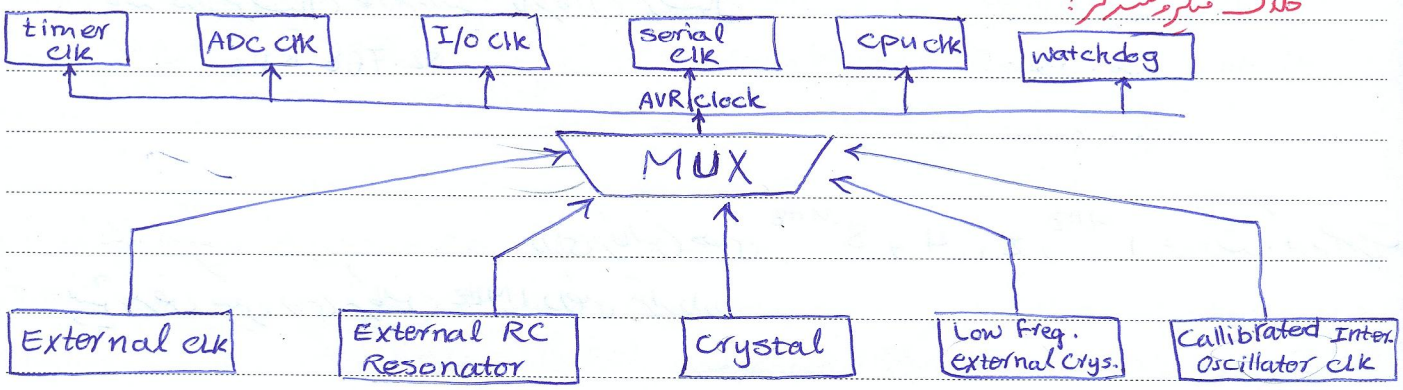
2ـ گذرگاه آدرس: مشخص‌کننده این است که اطلاعات در اختیار چه قسمتی قرار می‌گیرد.

3ـ گذرگاه کنترل: کنترل‌کننده ارتباط بین باس داده و آدرس است.

PAPCO
www.mohandesyar.com

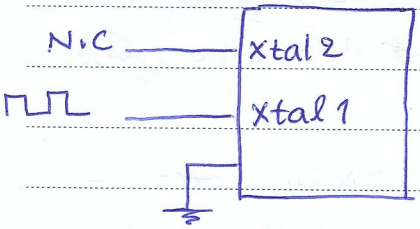
Subject:
Year. Month. Date. ( )

کلاک میکروکنترلر:

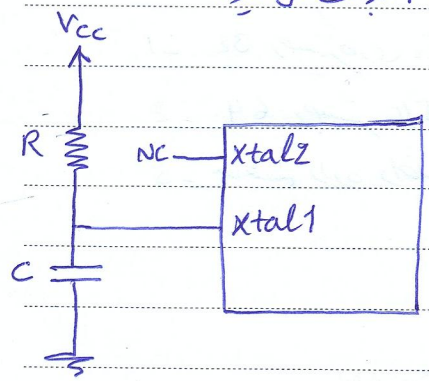


کلاک میکرو از منابع زیر تولید می‌شود که با انتخاب هر منبع و فیوز بیت مربوط به آن می‌توان میکرو را راه اندازی کرد:

۱- منبع کلاک خارجی: با دادن پالس به پایه XTAL1 و باز بودن XTAL2 می‌توان از کلاک خارجی به عنوان کلاک میکرو استفاده کنیم.

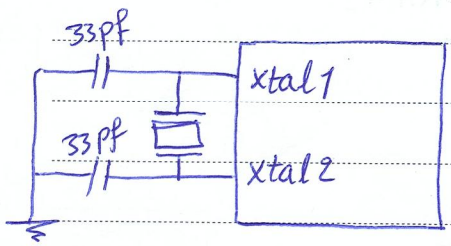


۲- اسیلاتور RC خارجی: با اتصال مدار زیر و رابطه $f \cong \frac{1}{3RC}$ بدست می‌آید.



نکته: $Min(C) \cong 22pf$

۳- کریستال خارجی: با استفاده از مدار روبرو و کریستال با فرکانس 16MHz بدست می‌آید.



PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

۴- کریستال خارجی فرکانس پایین:

مانند مدار کریستال خارجی است با فرکانس کریستال
32.768 KHz

۵- اسیلاتور کالیبره شده داخلی: دارای فرکانس های 8MHz و 4 و 2 و 1MHz می باشد، که به صورت پیش فرض میکرو روی فرکانس 1MHz داخلی قرار دارد.

حافظه میکروکنترلر:
- رجیستر کاری R_0 تا R31
- رجیستر I/O 64
- حافظه داخلی SRAM
- flash
- SRAM
- E^2PROM

1- Flash ROM : (Program Memory)

حافظه ای است که برنامه در آن ذخیره می شود. با پروگرام کردن میکرو، حافظه فلش پر می شود و خارج کردن میکرو از مدار پروگرامر، امکان تغییر حافظه وجود ندارد. این حافظه با قطع برق از بین نمی رود و حدود 10000 بار قابلیت نوشتن و پاک کردن دارد.

در میکرو ATMEGA16 حجم این حافظه 16KB می باشد.

2- حافظه SRAM: فضایی است که میکرو در هنگام اجرا کردن برنامه، متغیرها را در این فضا ذخیره می کند. با قطع برق اطلاعات این حافظه پاک می شود. اطلاعات توسط CPU در این حافظه نوشته می شود و یا از آن خوانده می شود. حافظه SRAM به سه بخش تقسیم می شود:

1- 32 رجیستر کاری R_0 تا R31
2- 64 رجیستر I/O
3- حافظه داده داخلی SRAM

Register Files (name) — Data Address Reg.

R_0	$0000
R_1	$0001
⋮	⋮
R29	
R30	
R31	$1F

I/O Register

$0001	$0020
$0002	$0021
⋮	⋮
$003F	$005F

Internal SRAM

$0060
$0061
⋮
$045F

PAPCO
www.mohandesyar.com

Subject:
Year. 91 Month. 1 Date. 22 ( )

Mega16 512 byte Mega32 1024 byte

3- حافظه E^2PROM:

برای ذخیره سازی داده هایی که می خواهیم با قطع برق از بین نرود ما می توان از این نوع حافظه استفاده کرد. با استفاده از دستورات لازم داده ها را در این حافظه می نویسیم. در Mega16 حجم این حافظه 512 byte می باشد. این حافظه 100 هزار بار قابلیت پاک شدن دارد.

منابع Reset:

میکرو برنامه در حال اجرا را متوقف کرده و از آدرس Reset شروع به کار می کند.

1- Reset شدن میکرو هنگام روشن کردن

2- ریست خارجی، وصل کردن پایه 9 به زمین.

3- Watchdog Reset: برای جلوگیری از هنگ کردن میکرو از کانتر Watch dog استفاده می شود. این کانتر با شمارش و هنگام رسیدن به مقدار Max، میکرو را ریست می کند.

4- Burn_out Reset: اگر ولتاژ میکرو از حد آستانه کمتر شود، میکرو Reset می شود.
Brownout

PAPCO

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

زبان C :

زبان C، زبانی با سطح میانی است که برای برنامه ریزی میکروکنترلرهای AVR قابل استفاده است. مزیت های این زبان بر زبان اسمبلی عبارتند از :

1- سادگی زبان C

2- انعطاف پذیری : یعنی با نوشتن برنامه به زبان C، نیازی به دانستن ساختار داخلی میکرو نیست و با تغییراتی جزیی می توان برنامه نوشته شده برای میکروکنترلر خاص را برای میکروهای دیگر نیز استفاده کرد.

3- داشتن دستورات کمتر

4- دسترسی به کتابخانه ها و توابع موجود

5- قابل فهم تر بودن برنامه زبان C

- مزیت استفاده از زبان اسمبلی بر C این است که در یک برنامه خاص، حجم کدهای تولید شده توسط اسمبلر، نسبت به کدهای تولید شده توسط کامپایلر حداقل 1/5 تا 2 برابر کمتر است. لذا اگر هدف ما سرعت و حجم کد پایین تر باشد بایستی از زبان اسمبلی استفاده کنیم.

نکات در مورد زبان C :

1- به جز دستوراتی که با علامت # شروع می شوند، تمامی دستورات ما به ; ختم می شوند.

```
a = a+b;
```

2- برای نوشتن عبارت توضیحی :

```
// توضیحات
/* توضیحات */
```

3- استفاده از آکولاد : هر آکولاد که باز می شود باید در جایی بسته شود.

```
if a==8 {
    ≡≡≡
}
```

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

توابع کتابخانه Header ← سرآیند

4- در یک خط می‌توان چندین دستور را نوشت:

a=a+b , a=a-b ;

5- تعریف متغیر قبل از استفاده متغیر است.

6- برای نامگذاری متغیرها می‌توان از حروف بزرگ و کوچک، اعداد و علامت Underline هم استفاده کرد. طول هر متغیر 32 کاراکتر است.

7- برای تعریف متغیرهای عددی از 3 شکل زیر استفاده می‌شود:

```
x = 255 ; // decimal
x = ØB11111111 ; // Binary
x = ØxFF ; // HexaDecimal
```

ساختار کلی برنامه:

- شروع هر برنامه با فایل‌های سرآیند (Header Files) می‌باشد، این فایل‌ها اطلاعاتی را به ابتدای برنامه الحاق می‌کنند. در واقع فایل‌های سرآیند توابعی از پیش تعریف شده می‌باشند.

```
# include <mega16.h>
```

- ساختار یک برنامه زبان C:

```
# include <1 فایل‌های سرآیند.h>
# include <2~ ~.h>
Global Var;
Defining Func;
int main Void {
Lacal Var;
Instructions;
}
```

- بعد از تعاریف کلی، main برنامه را داخل حلقه while می‌نویسیم. مقداردهی اولیه به متغیرها و رجیسترها قبل از حلقه انجام می‌گیرد.

```
while {
}
```


www.mohandesyar.com

Subject:
Year. Month. Date. ( )

انواع و محدوده داده ها:

- به دلیل اینکه عملیات میکرو به صورت 8 بیتی انجام می شود، بعد متغیرها به صورت 8 بیتی یا ضریبی از 8 تعریف می شود.

نوع متغیر	تعداد بیت	محدوده
bit	1	0, 1
(signed) char	8	-128 to 127
unsigned char	8	0 to 255
signed int	16	-32768 to 32767
unsigned int	16	0 to 65535
(signed) Long	32	-2147483648 to 2147483647
unsigned Long	32	0 to 4294967295
float	32	

انواع متغیرها:

۱- متغیرهای سراسری: قابل دسترس در تمام توابع برنامه

۲- " محلی: فقط در توابعی که تعریف می شوند، قابل استفاده است.

کلاس:

۱- متغیرهای Static: متغیرهایی هستند که در فراخوان های مختلف توابع، مقدار ثابت دارد.

```
static int n=1;
```

۲- متغیرهای Extern: با تعریف این متغیر، حافظه ای تشکیل می شود که قابل دسترسی است.

```
extern int a;
```

SRAM

3- متغیر eeprom: اگر هدف تعریف متغیرهایی باشد که با قطع برق از بین نرود، می توان متغیر را از نوع eeprom تعریف کنیم.

```
eeprom نوع متغیر نام متغیر ;
```

```
eeprom int a;
```


www.mohandesyar.com

Subject:
Year. 91 Month. 1 Date. 23 ( )

4- متغیرهای فلش (Flash): متغیرهایی هستند که فقط قابل خواندن هستند.

```
Flash int a;
Const
```

جلسه سوم:

تعیین آدرس ذخیره داده در حافظه SRAM:

```
int y @0x80;
```

ذخیره متغیر y در آدرس 80H

برای ارتباط سریال استفاده از آرایه ها سرعت بالاتری دارد.

آرایه ها: به یک سری از متغیرهای هم نوع آرایه می گویند. آرایه را می توان در فضای RAM یا E^2PROM ذخیره کرد.

- فرمت معرفی آرایه یک بعدی:

نوع آرایه — اسم آرایه — [طول آرایه]

int s[3] $\xrightarrow{OR}$ int s[ ]=[1,4,7,3]

- برای معرفی آرایه دو بعدی:

نوع آرایه — اسم آرایه — [بعد اول (سطر)][بعد دوم (ستون ها)]

int s[4][3]

- نحوه ذخیره آرایه دو بعدی به صورت سطری است.

عملگرها:

1- عملگرهای محاسباتی:

3 { +	a+b		
3 { −	a−b		
2 { ×	a*b	int x,y;	int x,y;
2 { /	a/b	x=10;	x=10;
2 { %	a%b	y=++x;	y=x++;
تقدم 1 { ++	a++	y=11, x=11	y=10, x=11
تقدم 1 { −−	a−−		

PAPCO

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

2- عملگرهای رابطه ای: مرتبط کننده دو عملوند.

>	x > y
>=	x >= y
==	x == y
!=	x != y

3- عملگرهای منطقی:

!	not	!x
&&	and	x && y
||	OR	x || y

تقدم:
1. !
2. >=
3. == , !=
4. &&
5. ||

4- عملگرهای بیتی:

and	&
OR	|
not	~
XOR	^
Right shift	>>
Left "	<<

تعداد شیفت >> اسم متغیر

a = 0b 0000 0001

y = a>>1 y = 0b 1000 0000

دستورات پیش پردازش:

1- دستور #inclue: برای ضمیمه کردن فایل های سرآیند از دستور (include) استفاده می شود که این فایل ها به دو صورت وجود دارد:

الف) فایل هایی که در کامپایلر موجود است: #include<mega8.h>

ب) فایل هایی که توسط برنامه نویس اضافه می شود: #include "Num.h"

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

2- define : این دستور به صورت سمبلیک عبارتی را با عبارت دیگر جایگزین می کند.

```
# define LED Portd.0
```

define LED Portd.0

کلمه LED را با Portd.0 جایگزین می کند.

```
# undef LED
```

```
# define LED Portd.0
# Undef LED
```

- حلقه های تکرار:

۱- حلقه تکرار for :

```
For (گام شمارش ; شرط حلقه ; مقدار اولیه شمارنده) {
دستورات
}
```

حلقه بی نهایت:

```
for( ; ; ){
}
```

2- حلقه while :

```
while (شرط) {
}
```

ابتدا شرط حلقه چک می شود، در صورتی که درستی شرط، دستورات اجرا می شود.

حلقه بی نهایت:

```
while (1){
}
```

3- حلقه Do - while :

```
Do {
} while (شرط)
```

ابتدا دستورات اجرا می شود بعد شرط حلقه چک می شود.

دستورات شرطی:

۱- If :

```
If (شرط) {
}
```

2- If-else

```
If (شرط) {
دستورات 1 ; }
else if (شرط) {
دستورات 2 ; }
else {
دستورات 3 ;
}
```

PAPCO

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

Switch - Case - 3:

```
switch ( عبارت ) {
    Case مقدار1 :
    <دستورات 1>
      break;

    Case مقدار2 :
    <دستورات 2>
      break;
      :
    default {
    <دستورات N>
    }
```

۱- با دستور break می‌توان از حلقه خارج شد.
۲- نوشتن default اختیاری است.
3- مقادیر Case ها باید متفاوت باشد.
4- از دستور switch می‌توان به صورت تودرتو استفاده کرد.

نوشتن برچسب:

```
{ Lable_Name ;
  ═══
  ═══
  goto xy;
       LableName
```

با استفاده از دستورات goto می‌توان به برچسب پرش کرد.

goto اسم برچسب

مثال:

```
while(1){
    if a==10 goto xy;
    a++;
  }
xy:
    Portd.0=1
```

PAPCO

www.mohandesyar.com

نحوه نوشتن توابع:

فرمت نوشتن یک تابع به صورت زیر است:

```
نوع تابع   نام تابع   (پارامترها) {

    دستورات ;
}
```

- هر تابع براساس هدفی که نوشته می‌شود دارای ورودی و خروجی است. به ورودی های تابع آرگومان تابع می‌گویند و به مقداری که تابع برمی‌گرداند، مقدار برگشتی آن تابع گویند.

1- هر تابع قبل از main نوشته می‌شود.
2- معرفی ورودی های تابع در تابع صورت می‌گیرد.
3- برای فراخوانی تابع، از نام تابع استفاده می‌شود.
4- در صورتی که تابع ورودی نداشت از Void استفاده می‌کنیم.

- توابع فراخوانی شده در برنامه براساس مقدار بازگشتی به دو دسته تقسیم می‌شوند:

الف) توابعی که یک مقدار بازگشتی دارند. برای بدست آوردن مقدار بازگشتی از دستور Return استفاده می‌شود.

عبارت: Return

ب) توابعی که مقدار بازگشتی ندارند، از نوع Void تعریف می‌شوند.

مثال: تابعی بنویسید که $w = \frac{x*y}{f}$ را به صورت ... 1، 12، 5 محاسبه کند.

```
#include <Mega32.h>
float g, w;
float name (int x, int y, int f){
g = x*y;
g = g/f;
return (g);
}
```

```
void main (void){
while (1){
w = name (1, 12, 5);
}
}
```


www.mohandesyar.com

Subject:
Year. Month. Date. ( )

توابع کتابخانه‌ای:

توابع کتابخانه‌ای موجود در کامپایلر، با دستور include فراخوانی و به برنامه الحاق می‌شود.

```
# include <math.h>
```

توابع تأخیر:

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

توابع تأخیر را به صورت فوق به برنامه الحاق می‌کنیم.

1- delay_ms() : این تابع اندازه آرگومانی که دریافت می‌کند، در اجرای برنامه تأخیر ایجاد می‌کند.

```
Void delay_ms(unsigned int)
```

مثال → delay_ms(1000) // $t_d = 1000ms$

2- delay_us() : ایجاد تأخیر به اندازه آرگومان ورودی برحسب μs.

مثال → delay_us(1000) // $t_d = 1000\mu s$

```
Void delay_us(unsigned int)
```

پورت های ورودی - خروجی (I/O):

- هر یک از پورت های میکرو دارای قابلیت خواندن، نوشتن و نسبت دادن متغیر است (Read - write - Modify)
- از هر پورت میکرو به صورت مستقل می‌توان به عنوان ورودی یا خروجی استفاده کرد.
- از هر پین می‌توان به اندازه 20^{mA} (Max) جریان کشید.
- برای هر پورت در حافظه SRAM، 3 رجیستر وجود دارد:

1- نوشتن داده در پورت: (رجیستر PORTX) (Port Data Register)

2- رجیستر جهت داده: (رجیستر DDRX) (Data Direction Register)

با مقدار دادن به DDRX، می‌توان ورودی یا خروجی بودن پورت را مشخص کرد.

1 ← خروجی
0 ← ورودی

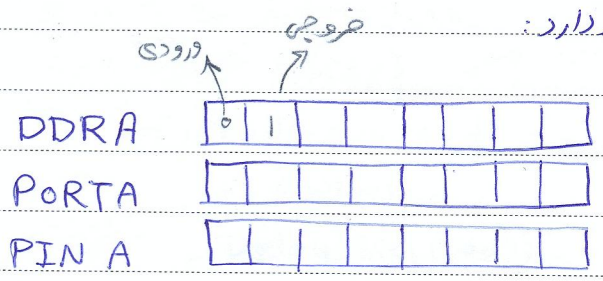
PAPCO

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

3- رجیستر خواندن از پورت : (PINX) (Port Input Pins Register)

- مثلاً در پورت A ، 3 رجیستر به صورت زیر وجود دارد:



- به طور مستقل می توان به هر پین (بیت) پورت دسترسی داشت:

PORTXn , PINXN , DDRXN

نام پورت
شماره پین

نکته مهم: در حالت ورودی می توان هر پورت را با مقاومت Pull up در نظر گرفت که با قرار دادن 1 ، مقاومت Pull up در نظر گرفته می شود و در صورت قرار دادن صفر، پورت در حالت امپدانس بالا (High Amp.) قرار می گیرد. مثلاً با نوشتن عدد 1 در PORT Xn می توان مقاومت Pull up را لحاظ کرد.

نکته 2: در حالتی که پورت در حالت خروجی است با نوشتن صفر یا یک در PORT XA می توان مقدار پورت را صفر یا یک نماییم.

مثال 1: برنامه ای بنویسید که اطلاعات از پورت D بخواند و در پورت B بنویسد.

```
#include <mega16.h>
Void main (void){
DDRD=0x00;
PORTD=0xFF;
DDRB=0xFF;
while (1){
PORTB=PIND;
}
}
```

ورودی Portd

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

2- برنامه‌ای بنویسید که هر 0.5s محتویات پورت B را مکمل کند.

```
#include <mega16.h>
#include <delay.h>
Void main (Void){
  DDRB = 0xFF;
  PORTB = 0x00;
while (1){
    PORTB ~= PORTB;
    Delay_ms (500)
  }
}
```

→ خروجی B

PORTB ~= PORTB

255

تمرین: برنامه‌ای بنویسید که عددی را پورت A بخواند در صورتی که عدد برابر 5 بود، آنرا در پورت B ذخیره کند.

```
#include <Mega16.h>
#include <delay.h>
Void main(Void){
  unsigned char a;
DDRA = 0x00;
PORTA = 0xFF;
DDRB = 0xFF;
while (1){
    a = PINA;
    If (a==5){
      PORTB = PINA;
      delay_ms (1000)
    }
  }
}
```

→ ورودی A
pullup

→ خروجی B

اگر عدد زوج باشد برود:

① تقسیم بر 2

② آخرین بیت پورت چک شود. (LSB)

PAPCO

www.mohandesyar.com

مثال: تابعی بنویسید که بیت صفر پورت D را بخواند. درصورتیکه برابر یک بود، بیت یک پورت D را 1 روشن کند.

```
# include <mega16.h>
# include <delay.h>
Void Blink (Void){
    if (PIND.0 ==1){
        PIND.1=1;
        Delay_ms(1000);
        PIND.1=0;
    }
}
Void main (Void){
    DDRD.0=0;   ← ورودی
    DDRD.1=1;   ← خروجی
    while (1){
        Blink();
    }
}
```

در دستور If می‌توان یک پین را چک کرد اما کل یک پورت را هم می‌توان خواند.

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

نمایشگر LCD :

برای نمایش خروجی برنامه ها و اشکال زدایی برنامه استفاده می شود. هر LCD از طریق ترانزیستور واسط، داده و دستور را از میکرو دریافت می کند و به حروف و یا کاراکترهایی که در حافظه دارد، تبدیل می کند.

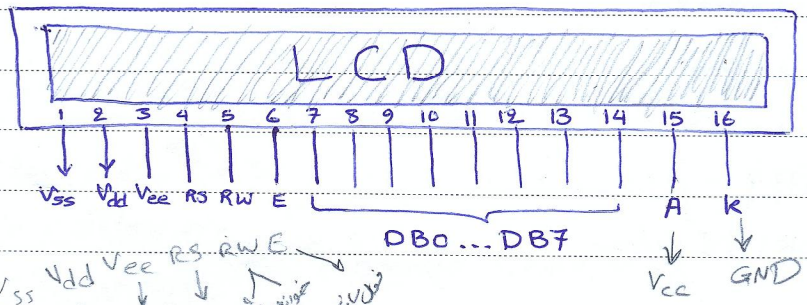
- انواع LCD :

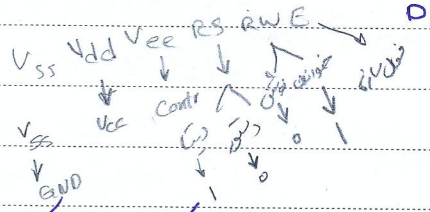
1- LCD متنی: در حافظه حروف A-Z و a-z، اعداد و تعدادی کاراکتر را ذخیره دارد.

2- LCD گرافیکی: اشکال مختلف را هم نشان می دهد.

- LCD های متنی بر اساس تعداد سطر و ستون تقسیم می شوند مثلاً 16x2، 16x1، 40x4 و ....

برای مثال LCD با اندازه 16x2 دارای 16 ستون و 2 سطر می باشد.





1- V_{ss} : زمین

2- V_{dd} : تغذیه

3- V_{ee} : تنظیم کنتراست. با قرار دادن سر متغیر پتانسیومتر به پایه 3 و وصل کردن دو سر دیگر به زمین و تغذیه و تغییر پتانسیومتر می توان کنتراست داده های نوشته بر روی LCD را تعیین کرد.

4- RS : مشخص می کند که ورودی LCD، داده یا دستور است، در حالتیکه ورودی، داده باشد، RS=1 و در حالتی که ورودی، دستور باشد، RS=0 می باشد. → to μC

5- RW : مشخص می کند که اطلاعات بر روی LCD نوشته می شود یا خوانده می شود. → to μC

RW=0 ← نوشتن اطلاعات

RW=1 ← خواندن داده

6- E : فعال ساز LCD است. با تغییر سطح پالس از 1 به صفر، داده و دستورات وارد شده به DB0 تا DB7 به LCD وارد می شود.

7 to 14 - DB : با صفر شدن E، داده و دستور در پایه DB0 - DB7 قرار می کند.

15- A : آند (نزدیک زمینه LCD)

16- K : کاتد (" " " LCD)

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

نحوه اتصال پایه های LCD به میکروکنترلر:

اگر در پروژه ای هدف اتصالات کمتر باشد، پایه های DB0 - DB3 را به میکرو وصل نمی کنیم و فقط DB4 - DB7 را به میکرو وصل می کنیم ولی اگر هدف سرعت تبدیل بالا (دیتا به کاراکتر) باشد بایستی از هر 8 پایه DB0 - DB7 استفاده کرد.

در نرم افزار CodeVision، امکان تنظیم برخی از پارامترها در قسمت CodeWizard فراهم شده است. مثلاً برای اتصال پایه های LCD به میکرو می توان از تنظیمات CodeWizard استفاده کرد.

با انتخاب پورت مورد نظر و تعداد ستون های LCD، نحوه اتصال پایه های LCD به میکرو به کاربر داده می شود.

LCD Port [C ▼]

تعداد پایه های LCD — Lines/chars [16 ▼]

Port Bit 0	→	E (1)	(LCD Pin 6)
" " 1	→	RS (2)	(" " 4)
" " 2	→	RW (3)	(" " 5)
" " 3	→	N.C	
" " 4	→	DB4	(" " 11)
" " 5	→	DB5	(" " 12)
" " 6	→	DB6	(" " 13)
" " 7	→	DB7	(" " 14)

بایستی پایه های LCD را مطابق جدول روبرو به میکرو وصل نمود.

با انجام تنظیمات، دستور زیر به ابتدای برنامه الحاق می شود:

```
# asm
    equ_LCD_Port=0x15 ; Port C
# end asm
```

آدرس پورت

equ: شبه دستوری که عبارت LCD_Port را برابر عدد 0x15 قرار می دهد.
این دستورات مشخص کننده این است که پایه های LCD به پورت C متصل شده است و علاوه بر این دستورات کتابخانه LCD را به ابتدای برنامه الحاق می کند. <LCD.h>

www.mohandesyar.com

ROM ← کاراکترها
DRAM ← حافظه موقت
RAM ← کاراکترهای جدید
LCD Memory

Subject:
Year. Month. Date. ( ) 1391 / 2 / 12

- هر LCD دارای سه نوع حافظه است:

1- ROM: که شامل کد اسکی کاراکترها، اعداد و حروف انگلیسی است که با دادن هر کد به LCD، عبارت مورد نظر چاپ می‌شود.

2- DRAM: حافظه موقت برای چاپ داده‌ها.

3- RAM: حافظه 64 بایتی که قادر به ذخیره کاراکترهای جدید است.

توابع LCD:

توابع LCD در یک پیکانه <LCD.h> قرار دارند که با ایجاد تنظیمات wizard به ابتدای برنامه الحاق می‌شوند.
این توابع عبارتند از:

```
# include <LCD.h>
```

1- LCD منتظر دریافت داده و دستور از میکرو می‌ماند.

```
LCD_Ready ()
Void LCD_Ready (Void)
```

2-

```
LCD_init ()
Void LCD_init (unsigned int)
```

Ex: LCD_init (16)

این تابع تعداد ستون‌های LCD را مشخص می‌کند. یعنی آرگومان تابع تعداد ستون‌ها است و با تنظیم wizard این تابع به برنامه اضافه می‌شود.

3-

```
LCD_clear ()
Void LCD_clear (Void)
```

این تابع صفحه نمایش را پاک می‌کند و مکان نما را به سطر اول و ستون اول منتقل می‌کند.

4-

ستون (از صفر شروع می‌شود) ← سطر

```
LCD_gotoxy (x , y)
Void LCD_gotoxy (unsigned int char x, unsigned int y)
```

موقعیت نوشتن بر روی LCD را مشخص می‌کند که x شماره ستون (0-15) و y شماره سطر (0 و 1) است.

```
LCD_gotoxy (5,0)
```

سطر اول، ستون ششم

PAPCO

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

ASCII

کد اسکی قرار می‌گیرد

LCD_Putchar( ) — کد اسکی می‌دهد

مثال: LCD_Putchar ("A")

5- برای چاپ کردن کاراکتر بر روی LCD است.

itoa (A, B) معادل اسکی A را در B قرار می‌دهد.

int → ascii

6- برای چاپ رشته ذخیره شده در حافظه SRAM استفاده می‌شود.

LCD_Puts( )

Putsf

LCD_Putsf( )

7- برای چاپ رشته ای که در حافظه Flash ذخیره شده است.

مثال: LCD_Putsf ("Micro")

با استفاده از علامت " " ، معادل کد اسکی به LCD فرستاده می‌شود.

- در متغیرهای عددی، باید مقدار متغیر را به کد اسکی آن تبدیل کرد. مثلاً متغیر Value که دمای محیط در آن ذخیره می‌شود، برای چاپ کردن آن از دستورات تبدیل عدد به کد اسکی استفاده می‌کنیم:

Itoa → تبدیل عدد Int به اسکی
Ltoa → " " Long " "

} → <stdlib.h> در کتابخانه استاندارد قرار دارند.

stdlib( )

مثال) Itoa(Value, Value1) — مقدار Value چاپ می‌شود

LCD_Putsf(Value1) — عبارت Value چاپ می‌شود

LCD_Putsf ("Value1")

مثال: برنامه ای بنویسید که هر 1s عبارت Micro را بر روی LCD چاپ کند.

```
#include <Mega16.h>
# include <delay.h>
# asm
 .equ_LCD_Port=0x15 ; PortC
# endasm
#include <LCD.h>
Void main (Void) {
wizard { LCD_init (16);
 while (1){
 LCD_Ready ( );
 LCD_Clear( );
 delay_ms (50);
 LCD_gotoxy (0,0);
 LCD_Putsf(" Micro");
 delay_ms (1000);
 }
 }
```

wizard

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

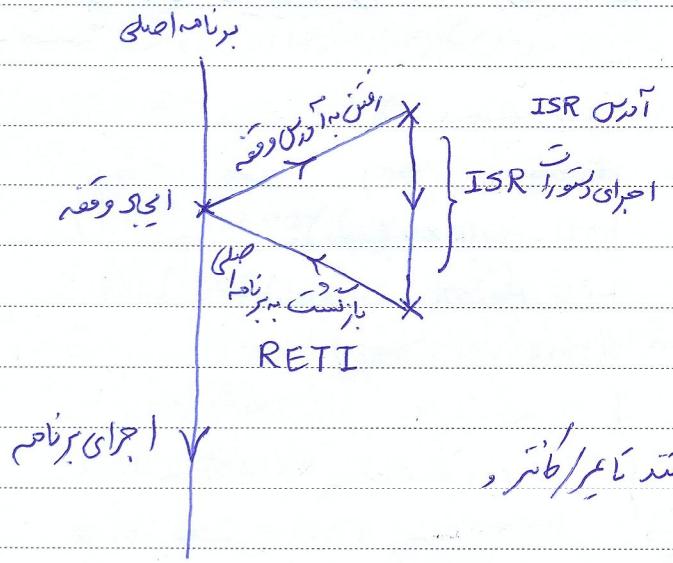
مثال) برنامه‌ای بنویسید که کلمه Micro را 5 بار به راست شیفت دهد.

```
int i;
void main (Void){
 LCD_init(16);
 for(i=0 ; i<5 ; i++) {
    LCD_clear();
    LCD_gotoxy(i,0);
    LCD_Putsf ("Micro");
    delay_ms(1000);
  }
 }
 while(){
 }
```

وقفه (Interrupt):

برنامه میکرو خط به خط اجرا می‌شود و با ایجاد وقفه در برنامه، اجرای برنامه متوقف شده به ISR یا روتین سرویس وقفه پاسخ داده می‌شود و پس از اجرای دستورات ISR به برنامه اصلی برگشت داده می‌شود.

ISR: Interrupt Service Routine



- دستگاههای مختلف می‌توانند از میکرو تقاضای وقفه کنند مانند تایمر/کانتر و A/D و ارتباط سریال و ...

PAPCO
www.mohandesyar.com

در کدویژن برنامه ریزی تایمر یا کانتر فقط در کلاک خارجی تایمر است و بقیه قسمت ها یکی است.

Subject:
Year. 1391 Month. 2 Date. 20 ( )

- هر یک از اینترایپت های مربوط به امکانات میکرو دارای آدرسی در حافظه Flash است که توسط شرکت سازنده مشخص شده است.
- هر کدام از امکانات در صورتی تقاضای وقفه می دهند که وقفه مربوط به آن امکان، فعال شده باشد.
- مثلاً در میکرو ATMEGA16، آدرس مربوط به سرریز تایمر صفر 0X12 است.
- اگر چند منبع وقفه میکرو هم زمان تقاضای وقفه کنند، پاسخ وقفه، براساس اولویت وقفه می باشد یعنی امکاناتی که در قسمت بالاتری از جدول قرار دارند، اولویت بالاتری دارند. (امکاناتی که آدرس وقفه کوچکتری دارند، اولویت بالاتری دارند.)

تایمر/کانتر:

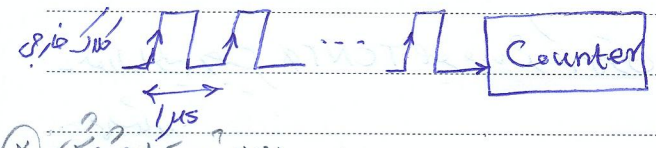
انواع T/C که در میکرو هستند عبارتند از:

1. تایمر/کانتر صفر ← 8 بیتی (0-255)
2. تایمر/کانتر یک ← 16 بیتی (0-65535)
3. تایمر/کانتر دو ← 8 بیتی (به عنوان کانتر استفاده نمی شود)

تایمر: شمارنده ای است که کلاک داخلی میکرو را می شمارد، بر خلاف کانتر که پالس خارجی را می شمارد.

- اگر هدف شمارش پالس خارجی باشد، سیگنال را باید به صورت خارجی به پایه های Pb.0 (تایمر کانتر صفر) و Pb.1 (تایمر ۱) متصل کرد.
- تایمر 2 به عنوان کانتر استفاده نمی شود.
- در کانتر، فقط لبه های بالارونده / پایین رونده مهم است نه دوره زمانی (فرکانس).
- در تایمر، کلاک میکرو (یا تقسیمی از کلاک میکرو) که دارای دوره زمانی ثابت می باشد، شمارش می شود، به علت ثابت بودن کلاک، تعداد شمارش در تایمر مفهوم زمانی دارد. (متناسب با زمان است).

(1) $f_{Xtal} = 1\ MHz \longrightarrow T = 1\mu s$

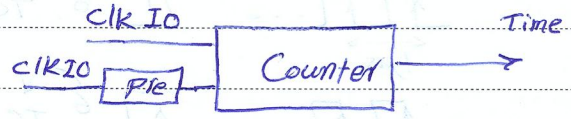


$t = 100\mu s \longrightarrow$ تعداد شمارش $= 100$

(2) افزایش تعداد شمارش

$\begin{cases} f_{xtal} = 1\ MHz \\ t = 800\ \mu s \end{cases} \longrightarrow$ prescale = 8

$f_{clkTimer} = \frac{f_{IO}}{pre.} = \dots$



تعداد شمارش $= 100$

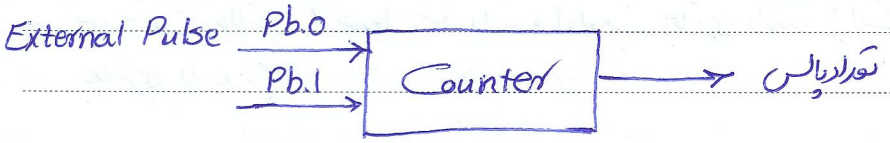
PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

- برای اندازه گیری زمان، کلاک کانتر یا از کلاک داخلی میگیرد یا تقسیمی از کلاک گرفته می‌شود.

Prescale = 1, 8, 64, 256, 1024

- اگر هدف شمارش پالس باشد:



مدهای کاری تایمر/کانتر:

CTC

تایمر 1:
1 - Normal Mode → تایمر صفر
2 - Fast PWM
3 - phase Correct PWM
4 - clear Time on Compare Match (CTC) با یک مقدار خاص مقایسه می‌شود در صورت برابری مقدار تایمر/کانتر صفر می‌شود.

تایمر 2 و 0:
Normal Mode
CTC

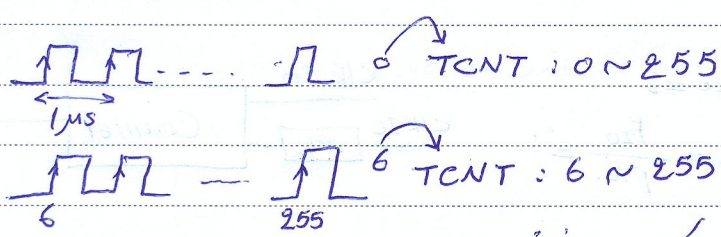
رجیستر های تایمر/کانتر صفر:

۱- رجیستر TCNT0 (Timer Counter Register)

- رجیستری است که تعداد شمارش را ذخیره می‌کند. به ازای هر لبه بالارونده/پایین رونده از کلاک، یک واحد به TCNT0 اضافه می‌شود و وقتی که به مقدار 0xFF (Max یا Top) رسید مقدار TCNT0 به مقدار Min (0x00) تبدیل می‌شود که در این حالت Flag مربوط به سرریز تایمر صفر تغییر وضعیت می‌دهد.

	7	6	5	4	3	2	1	0
TCNT0								

- مقدار اولیه رجیستر TCNT0 صفر می‌باشد که می‌توان به TCNT0 مقدار اولیه داد که تایمر از مقدار اولیه تا Max TCNT0 می‌شمارد.



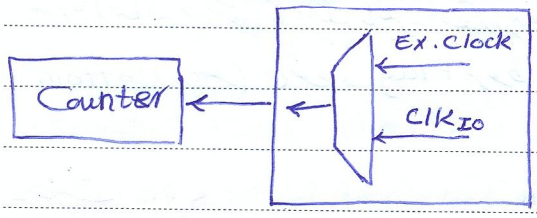
تعداد شمارش = 250

- می‌توان در حالی TCNT0 را مقدار دهی کرد و یا خواند.

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

(Timer Counter Controll Register) TCCR∅ -2



مشخص کننده این است که از تایمر به عنوان تایمر استفاده می شود یا کانتر
همچنین فرکانس کلاک تایمر نیز با این رجیستر مشخص می شود.

	7	6	5	4	3	2	1	0
TCCR∅						CS02	CS01	CS00

CS02	CS01	CS00	
0	0	0	تایمر متوقف شده است.
0	0	1	$f_{clkIO} = f_{timer}$ (Pre. = 1)
0	1	0	$f_{Timer} = \frac{f_{clkIO}}{8}$ (Pre. = 8)
0	1	1	(Pre = 64)
1	0	0	(Pre = 256)
1	0	1	(Pre = 1024)
1	1	0	کلاک کانتر خارجی و شمارش با لبه بالا رونده
1	1	1	" " " " " " پایین رونده

- با مقداردهی CS00 → CS02 می توان تایمر را از حالت خاموش بودن خارج کرد.
- با تغییر هر یک از بیت های CS00 → CS02 می توان تایمر را با کلاک های متفاوت راه اندازی کرد.
- در 2 حالت آخر پالس خارجی متصل به Pb.0 با لبه بالا رونده یا پایین رونده شمارش می شود.

PAPCO
www.mohandesyar.com

مدهای کاری تایمر: { مد نرمال / مد Compare Match

- حالت نرمال: تایمر/کانتر از مقدار اولیه شروع به شمارش کرده، به ازای هر لبه بالارونده (پایین‌رونده) یک واحد به تعداد شمارش شده اضافه می‌شود تا اینکه تایمر به مقدار Max (TOP) برسد. با آمدن لبه بالارونده بعدی مقدار تایمر برابر Min یا Bottom (صفر) می‌شود و Flag مربوط به سرریز فعال می‌شود.

- حالت CTC: CTC: Clear time on Compare Match

اگر تعداد پالس‌های شمارش شده (عدد تایمر) با عدد Compare برابر شود، تایمر صفر می‌شود و پرچم سرریز یک می‌شود.

3- رجیستر فعال کردن وقفه مربوط به تایمر صفر (TIMSK):

TIMSK: Timer/Counter Interrupt Mask

این رجیستر جهت فعال‌سازی وقفه مربوط به تایمر به کار می‌رود که بیت صفر و یک آن مربوط به تایمر صفر است.

(اندازه‌گیری فرکانس) Capture تایمر 1

7	6	5	4	3	2	1	0	
OCIE2	TOIE2	TICIE1	OCIE1B	OCIE1A	TOIE1	OCIE0	TOIE0	TIMSK

Timer OverFlow Interrupt Enable

تایمر 1 روی 2 پایه می‌تواند موج PWM بدهد (OCR1A و OCR1B)

تایمر صفر

- بیت صفر رجیستر TIMSK: (TOIE0)

T/C over Flow Interrupt Enable

با یک کردن بیت صفر وقفه مربوط به تایمر صفر فعال می‌شود یعنی در صورت سرریز شدن تایمر و خطای OVF، زیربرنامه وقفه مربوط به تایمر صفر اجرا می‌شود. (پرچم سرریز وقفه پس اجرای زیربرنامه به صورت خودکار صفر می‌شود.)

- بیت یک رجیستر TIMSK: (OCIE0)

Compare OverFlow Interrupt Enable.

با یک کردن این بیت، وقفه مربوط به تایمر صفر در حالت CTC فعال می‌شود. یعنی هرگاه مقدار تایمر با مقدار Compare برابر شد، پرچم مربوط به حالت CTC تغییر وضعیت داده و زیربرنامه مربوط به آن اجرا می‌شود.

OCR1A OCR1B → Pd.4 , Pd.5 روی پایه‌های مربوطه اجرا می‌شود

مقدار اولیه CTC قرار می‌گیرد

4- رجیستر TIFR:

Timer/Counter Interrup Flag Registers.

این رجیستر در بردارنده بیت‌های پرچم در تایمر/کانتر صفر، یک یا دو است.

PAPCO
www.mohandesyar.com

TCNT , TCCR , TIMSk , TIFR

مقدار دهی — رجیستر کنترلی — فعال سازی وقفه — Flag

Subject:
Year. Month. Date. ( )

7	6	5	4	3	2	1	0	
OCF2	TOV2	ICF1	OCF1B	OCF1A	TOV1	OCFØ	TOVØ	TIFR

Capture مد

تمامی بیت های این رجیستر توسط خود میکرو مقدار می گیرد.

TOVØ : Timer overflew Flag

- بیت صفر TIFR : (TOVØ)

با شمارش تایمر از صفر تا مقدار Max ، به محض ماکزیمم شدن تایمر، مقدار بیت صفر این رجیستر (TOVØ) برابر یک شده و در صورت فعال کردن وقفه های سراسری و وقفه های مربوط به تایمر صفر، زیر برنامه ISR اجرا می شود. و پس از اجرای کامل زیر برنامه مقدار TOVØ=0 شده و برنامه اصلی اجرا می شود.

Overflew Compare flag

- بیت یک TIFR : (OCFØ)

در صورتی که مقدار تایمر با عدد Compare برابر شود و وقفه های مربوط به تایمر صفر، فعال شده باشد این Flag از صفر به یک تغییر وضعیت داده و زیر برنامه مربوط به Interrupt اجرا می شود. پس از اجرای زیر برنامه مربوط به وقفه، این پرچم به صورت خودکار صفر می شود.

OCR : output Control Register

5- رجیستر کنترلی خروجی : (OCR)

عرض پالس موج PWM ، فرکانس موج به وجود آمده در پایه OCR1A و OCR1B را مشخص می کند.

نکته: علت اصلی استفاده از وقفه این است که اگر از روش معمولی (بدون وقفه) یا همان روش سرکشی استفاده کنیم باید یک دستور شرطی قرار دهیم تا مقدار تایمر چک شود. با توجه اینکه دستورات فرمانی از میکرو می گیرد، ممکن است قبل از رسیدن به دستور IF مقدار تایمر به مقدار دلخواه برسد و هنگام رسیدن به دستور IF مقدار تایمر از مقدار دلخواه گذشته باشد پس دستور IF اجرا نمی شود. دقت روش سرکشی کم است.

Timer = 250

TCNT = 255

IF (TCNT = 250)

شرط نادرست می شود

نکته ۲: برای تولید زمان دلخواه به رجیستر TCNTØ مقدار اولیه می دهیم تا وقتی که به Max برسد OVF یک شود. روش دیگر این است که از مد CTC استفاده کنیم.

6 — 250ms

TCNT = 0 ——— X ———→ Max = 255ms

TCNT مقدار اولیه

OVF (TOVØ = 1)

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

تنظیمات تایمر در Code Wizard :

Timer → Timer ∅

۱- مشخص کردن منبع کلاک :

Clock Source : { System Clock → Timer / External Clock → Counter

با انتخاب S.C از تایمر صفر با کلاک داخلی استفاده می‌شود و با انتخاب E.C، پالس متصل به Pb.0 شمرده می‌شود.

2- مقدار Clock : Pre Scale

Clock Value :

در صورت انتخاب S.C در حالت قبل، این گزینه فعال شده و مقادیر مختلف prescale را می‌توان انتخاب کرد.

$f_{clk\,Io} = 1\ MHz$

$Clk\ Value = 1024$

$f_{clk\ timer} = \frac{1\ MHz}{1024} = 0.997\ kHz \rightarrow T \simeq 1ms$

3- مشخص کردن حالت کاری تایمر :

Mode : { Normal / Compare

1391/3/2

4- Timer Value : مقدار اولیه تایمر (TCNT∅) در حالت نرمال

5- Compare Value : مقدار اولیه تایمر در حالت CTC

6- OVF Int : با انتخاب این گزینه زیربرنامه ای به برنامه اصلی اضافه می‌شود. (زیربرنامه وقفه تایمر/کانتر).

7- CMP Int :

با فعال کردن OVF Int زیربرنامه اینترایپت به صورت زیر به ابتدای برنامه اضافه می‌شود:

```
Interrupt [timer_OVF] Void timer∅_OVF_isr (Void)
{
    // Enter Your Code Here.
}
```

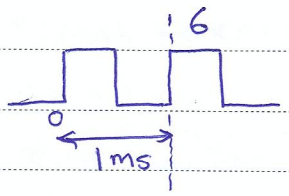
PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

- برای فعال کردن وقفه سراسری بایستی دستور زیر را به برنامه اضافه کرد:

```
# asm ("sei")
```

250 ms = t_d



مثال: برنامه ای بنویسید که هر بار تایمر صفر Max شد، تقاضای وقفه کند و بیت صفر پورت A را NOT کند.

$f_{xtal} = 1^{MHz}$ PreScale = 1024

مراحل برنامه نویسی تایمر:

1- فراخوانی تابع ها

2- زیر برنامه وقفه

3- main (مقداردهی رجیسترهای تایمر)

```
# include <Mega16.h>
Interrupt [tim0_OVF] Voide Timer0_OVF isr(Void)
{
    PORTA.0 = !PORTA.0;
}
Void main (Void){
    DDRA = 0x01;
    PORTA = 0x00;
    TCNT0 = 0x00;
    TCCR0 = 0x05;
    TIMSK = 0x01;
    # asm ("sei")
    while (1) {
    }
}
```

زیر برنامه وقفه

پورت A خروجی

مقدار اولیه تایمر

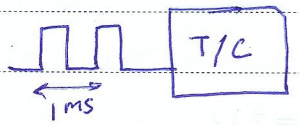
کلاک تایمر (Pre=1024)

فعال سازی وقفه تایمر

وجود حلقه while الزامی است

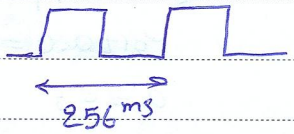
$f_{clk\ Timer} = \frac{1^{MHz}}{1024} \cong 1kHz$

T = 1ms



$t_d = 256^{ms}$

خروجی Port0A.0 →



PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

مثال: برنامه‌ای بنویسید که هر 250^{ms} ، پورت A را NOT کند.

```
#include <mega16.h>
Interrupt [timo_OVF] Void timer0_OVF_isR (Void)
{
  Porta.0 != Porta.0 ;
  TCNT0 = 6 ;
}
Void main (Void) {
  ~~int a~~
  DDRA = 0x01 ;
  PORTA = 0x00 ;
  TCNT0 = 0x06 ;
  TCCR0 = 0x05 ;
  TIMSK = 0x01 ;
  # asm ("sei")
  while (1) {
    }
  }
```

مثال: برنامه‌ای بنویسید که هر 500^{ms} پورت A بیت صفر را NOT کند. ← $(500^{ms} = 2 \times 250^{ms})$

```
# include <mega16.h>
Interrupt [tim_OVF] Void Timer0_OVF_isr (Void)
{
    a++;
    If (a==2){
        Porta.0 != Porta.0 ;
        a=0;
      }
    TCNT0 = 0x06;
}
```

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

```
Void main (void) {
int a;
DDRA = 0x01;
PORTA = 0x00;
TCNT0 = 0x06;
TCCR0 = 0x05;
TIMSK = 0x01;
a = 0;
#asm ("sei")
while(1) {
}
}
```

A.0 } DDRA, PORTA } A.0 = out

TCCR0 → Prescale = 1024

TIMSK → Timer OVF Interrupt Enable

#asm("sei") → Enable All Interrupt Source

برای طراحی تایمر:
1- $f_{Timer} \leftarrow$ Pre. و f_{xtal}
2- TCNT0
3- تعداد رخداد OVF

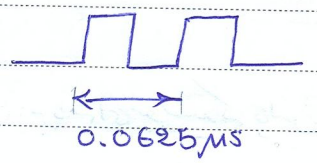
$f_{clk_T} = \frac{16^{MHz}}{1}$ → $T_{clk\,timer} = 0.0625\,\mu s$

$255 - TCNT0$

$f_{xtal} = 16\,MHz$
Pre Scale = 1

مثال: برنامه ای بنویسید که هر 1s یک واحد به متغیر b اضافه کند.

$f_{clk\,Timer} = 16^{MHz} \rightarrow T = 0.0625\,\mu s$ $\qquad 1^s = 4 \times 250^{ms}$

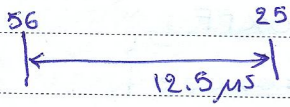


$\frac{0xFF - TCNT0}{t_{clk\,Timer}} =$ تعداد شمارش

$\frac{\text{زمان مطلوب}}{\text{زمان شمارش}} =$ تعداد OVF

زمان شمارش ← تعداد شمارش $t_{clk\,timer}$

$TCNT0 = 56$ مقدار اولیه



→ تعداد شمارش = 200 زمان در یک بار سرریز شدن: $200 \times T_{clk\,Timer} = 12.5\,\mu s$

تعداد رخداد OVF $= \frac{1^s}{12.5\,\mu s} = 80000$

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

```
# include <mega16.h>
Interrupt [timØ_OVF] Void timerØ_OVF_isr(Void)
  {
    a++;
    if (a==80000){
        b++;
        a=0;
    }
    TCNTØ = 0x38; \\ TCNTØ = 56
  }
Void main(Void){
  Long a,b;
  TCNTØ = 0x38;   tCNTØ = 56; // 56 Decimal = 38 Hex
  TCCRØ = 0x01;
  TIMSk = 0x01;
  a=0;
  # asm ("sei")
  while(1){
    }
  }
```

مثال: برنامه‌ای بنویسید که پالس متصل به Pb.0 را شمرده و تعداد شمارش را در پورت D ذخیره کند.

CS02, CS01, CS00 = 101 OR 111

```
# include <mega16.h>
Void main(Void){
PORTB = 0xFF;   } B ورودی , Pullup
DDRB = 0x00;
DDRD = 0xFF;    } D خروجی
TCNTØ = 0x00;
TCCRØ = 0x07;   کانتر ولیه پایین رونده
while (1){
  PORTD = TCNTØ;
}}
```

PAPCO

www.mohandesyar.com

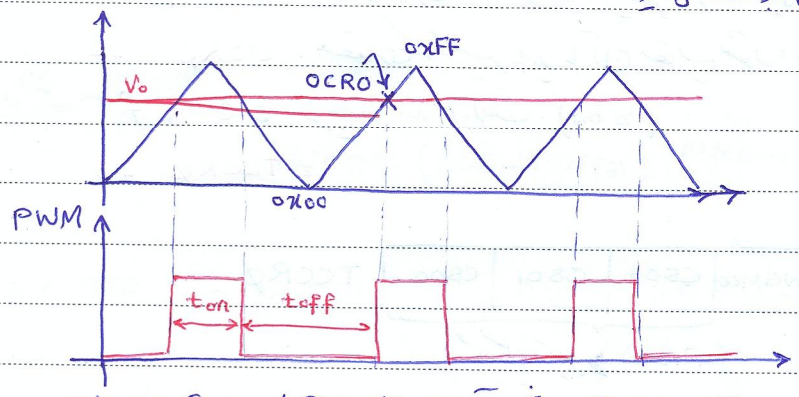
Subject:
Year. 1391 Month. 3 Date. 3 ( )

: PWM

PWM : Pulse Width Modulation

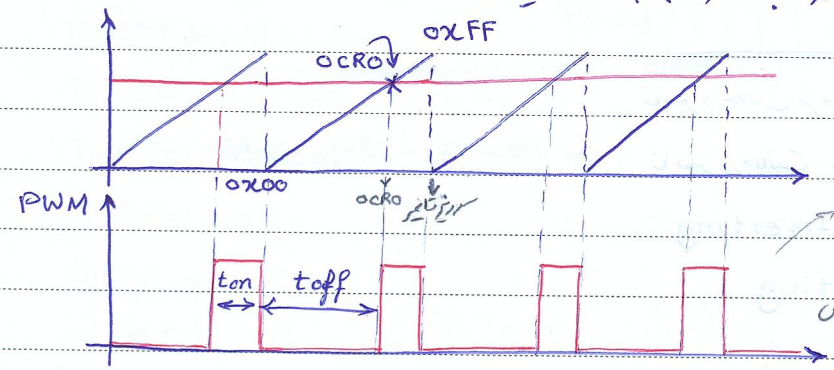
تولید موج PWM توسط تایمر صفر:

1- از مقایسهٔ یک موج مثلثی با مقداری ثابت (dc) بدست می‌آید.



Phase Correct PWM تولید PWM به روش تصحیح فاز

2- از مقایسه یک موج دندانه اره‌ای با مقدار ثابت (dc) بدست می‌آید.



فرکانس دو برابر قبل → Fast PWM

از یک فرکانس خاص → ton
pha- نصف حالت ton

Fast PWM تولید PWM

OCRØ

- در حالت تصحیح فاز، تایمر به صورت صعودی و نزولی می‌شمارد، در این حالت هر لحظه مقدار تایمر (TCNTØ) با مقدار OCRØ مقایسه می‌شود. هنگامی که مقدار تایمر برابر OCRØ (TCNTØ = OCRØ) باشد پایه Pb3 (OCØ) یک می‌شود، تایمر تا مقدار Max می‌شمارد و سپس به صورت نزولی شروع به شمارش می‌کند، پایه Pb3 در مدت زمان مساوی شدن تایمر با OCRØ یک می‌ماند تا اینکه مقدار تایمر از OCRØ کوچکتر شود، سپس پایه Pb3 صفر می‌شود.

- در حالت Fast PWM، شمارش تایمر صعودی است. تایمر از 0x00 شروع به شمارش کرده، در لحظه‌ای که TCNTØ = OCRØ می‌شود، پایه Pb3 یک شده و تایمر تا Max می‌شمارد. به محض سرریز شدن تایمر Pb3 صفر می‌شود.

$$f_{phase\ Cor.} = \frac{f_{fast\ PWM}}{2}$$

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

- اگر در حالتی که TCNTØ = OCRØ، پایه Pb3 یک شود، موج تولید شده Inverted نامیده می‌شود و اگر صفر شود Non-Inverted نامیده خواهد شد.

- رجیسترها:

1- TCNTØ: تعداد شمارش شده توسط تایمر (8 بیتی)

2- OCRØ: مقداری که تایمر باید با آن مقایسه شود، در این رجیستر ذخیره می‌شود که مشخص کننده f و D.C موج PWM است. (8 bit) ← با توجه به روابط

3- TCCRØ:

WGM00 WGM01

FOC0	WGM01	COM01	COM00	WGM00	CS02	CS01	CS00	TCCRØ
0	1	1	0	1	0	1	1	

(Pre.) کلاک تایمر

- COM00, COM01: مشخص کننده Inverted یا Non-Inv. بودن PWM است.

COM01	COM00	
0	0	تایمر در حالت نرمال کار می‌کند.
0	1	تایمر در حالت CTC کار می‌کند. → Reserved for CTC
1	0	Non-Inverting
1	1	Inverting

- WGM01, WGM00: مشخص کننده نوع PWM تولیدی است.

WGM01	WGM00	حالت تایمر	Max
0	0	نرمال	0xFF
0	1	P.C. PWM	0xFF
1	0	CTC	OCRØ
1	1	Fast PWM	0xFF

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

- برای تولید موج PWM باید پایه Pb3 به عنوان خروجی تعریف شود.
- برای محاسبه فرکانس موج در حالت P.C. PWM از رابطه زیر استفاده می‌کنیم.

$$f_{PWM} = \frac{f_{clk\,I/O}}{Prescale \times 510} \longleftrightarrow \text{Phase Correct PWM}$$

($f_{clk\,I/O} \to f_{xtal}$, $510 \to 2 \times 255$)

$$f_{PWM} = \frac{f_{clk\,I/O}}{Prescale\,(256 - TCNT0)} \longleftrightarrow \text{Fast PWM}$$

مثال: برنامه‌ای بنویسید که موج PWM را در حالت های P.C و Fast با استفاده از مقدار OCR0 = 0x80 ، $f_{xtal} = 1\,MHz$ و Pre. = 1 و Non_Inverting تولید کند.

Phase Correct

```
# include <mega16.h>
Void main (Void){
PORTB = 0x00;
DDRB = 0x08;
TCNT0 = 0x00;
TCCR0 = 0x29;
OCR0 = 0x80;
while (1){
    }
}
```

→ Pb.3 = out

→ Fast → TCCR0 = 0x49

	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	
TCCR0	0	0	1	0	1	0	0	1	Ph. C.
TCCR0	0	1	0	0	1	0	0	1	Fast

$$f_{PWM} = \frac{1\,MHz}{1 \times 510} \cong 2\,kHz$$

$$f_{PWM} = \frac{1\,MHz}{1 \times (256 - 0)} \simeq 4\,kHz$$

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

f_{xtal} $f_{xtal1} = \cdots$

مثال: برنامه‌ای بنویسید که با فرکانس 1KHz و D.C = 20% با روش Fast تولید کند.

$$\begin{cases} f_{xtal} = 16\ MHz \\ pre = 64 \end{cases}$$

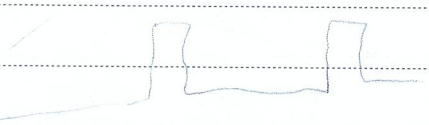
$$f_{pwm} = 1KHz = \frac{16^{MHz}}{64(256-\underbrace{TCNT0}_{6})} \longrightarrow TCNT0 = 6$$

$$D.C = \frac{\overset{t_{on}}{OCR0}}{\underbrace{256 - TCNT0}_{T_0}} \times 100\%$$

$$20\% = \frac{OCR0}{256} \times 100\% \longrightarrow OCR0 = 50 \longrightarrow \text{Non-Inverting}$$

```
# include <mega16.h>
Interrupt [tim0_ovf] void timer0_ovf_isr(void){
    TCNT0 = 0x06;
}
void main(void){
PORTB = 0x00;
DDRB = 0x08;  Pb.3
TCNT0 = 0x06;
TCCR0 = 0x6B;
OCR0 = 0x32;   50 Decimal
TIMSK = 0x01;
# asm ("sei")
while (1){
    }
}
```

WGM00	Com01	Com00	WGM01					
0	1	1	0	1	0	1	1	TCCR0
6				B				



PAPCO

www.mohandesyar.com

وقفه در حالت Compare :

اگر تایمر از مقدار اولیه شروع به شمارش کند، با رسیدن به مقدار مشخص OCRØ، مقدار تایمر سرریز شده و دوباره از مقدار min (صفر) شروع به شمارش می کند.

با فعال کردن وقفه مربوط به تایمر (در حالت CTC) می توانیم با هر بار سرریز شدن زیر برنامه وقفه اجرا می شود.

مثال: برنامه ای بنویسید که با 200 بار شمارش کلاک تایمر ۰ تایمر صفر را سرریز کند. (با وقفه)

```
#include <Mega16.h>
Interrupt [timØ_Comp] Void timerØ_isr (Void)
    {

    // Place Your Code Here.

    }
Void main (Void) {
    TCCRØ = 0x01;
    TCNTØ = 0x00;
    OCRØ = 0xC8;
    TIMSK = 0x02;
    # asm ("sei")
    while (1) {
        }
    }
```

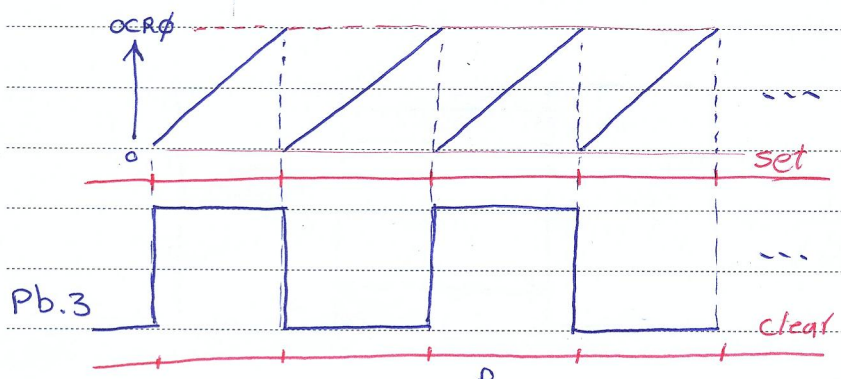
اشکال دارد

PreScale = 1

$f_{xtal} = 1\ MHz$

هر بار 200 µs

- استفاده از حالت CTC برای تولید موج مربعی:



تایمر صفر از مقدار اولیه شروع به شمارش کرده و با رسیدن به مقدار OCRØ دوباره صفر می شود. و در این مدت زمان پایه PB.3 را Not می کند.

فرکانس موج ساخته شده در پایه PB.3

$$f_{OCR} = \frac{f_{clk\,IO}}{2 \times PreScale(1 + OCRØ)}$$

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

- با مشخص کردن بیت های COMØØ و COMØ1 می توان مشخص کرد که در فواصل شمارش ما پایه Pb.3 ، Not Set یا Clear شود.

COM01	COM00	
0	0	مد نرمال
0	1	NOT و CTC
1	0	CTC و صفر Clear
1	1	CTC و یک Set

مثال: برنامه ای بنویسید که با استفاده از حالت CTC با فرکانس 1MHz بر روی پایه Pb.3 تولید کند.

$$\begin{cases} f_{xtal} = 16\ MHz \\ PreScale = 1 \end{cases}$$

$$1\ MHz = \frac{16\ MHz}{2 \times 1 \times (1 + OCRØ)} \rightarrow OCRØ = 7$$

	0	0	1	1	0	0	1	TCCRØ
	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	

```
Void main(Void){
    DDRB = 0x08;
    TCNTØ = 0x001
    TCCRØ = 0x19;
    OCRØ = 0x07;
    while(1){
        }
    }
```

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

مثال: با فشردن کلید متصل به Pd.0 فرکانس افزایش
" " " Pd.1 " " " کاهش

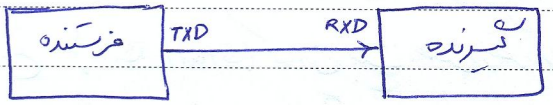
```
void main (void) {
    DDRB = 0x08;
    DDRD = 0x00;
    TCNT0 = 0x00;
    TCR0 = 0x19;
    OCR0 = 0x07;
    while (1) {
        if (Pind.0 == 0)   OCR0 = OCR0 - 100
        if (Pind.1 == 0)   OCR0 = OCR0 + 100
        delay_ms (500)
    }
}
```


ارتباط سریال:

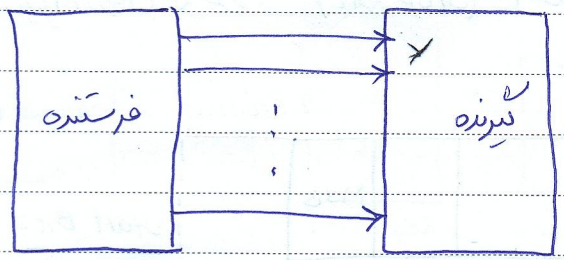
- در ارتباط سریال، اطلاعات به صورت بیت به بیت ارسال می‌شود یعنی در ارسال 8 بیت اطلاعات، هر بیت با یک کلاک پالس ارسال می‌شود. امّا در ارتباط موازی، اطلاعات در یک کلاک پالس ارسال می‌شود پس سرعت ارتباط موازی از ارتباط سریال بیشتر است.

- در ارتباط سریال، برای ارسال به دو سیم احتیاج داریم:
  1- دریافت Reeieve (RXD)
  2- ارسال Transmint (TXD)

- در ارتباط موازی به ازای هر بیت، از یک سیم استفاده می‌شود، لذا در ارتباط موازی حجم و هزینه افزایش پیدا می‌کند.



ارتباط سریال
(Serial)



ارتباط موازی
(Parallel)

www.mohandesyar.com

Subject:
Year. Month. Date. ( )

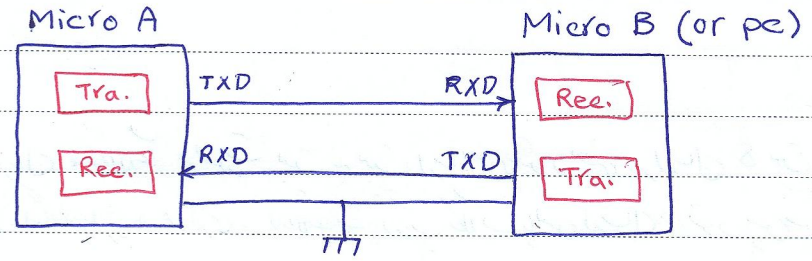
انواع ارتباط سریال:

1- ارتباط Simplex (یک طرفه): فرستنده فقط ارسال و گیرنده دریافت می کند.

2- ارتباط Duplex (دو طرفه): هر دستگاه هم می تواند فرستنده باشد و هم گیرنده.

Duplex:
- Half-Duplex: ارسال و دریافت همزمان صورت نمی گیرد.
- Full-Duplex: ارسال و دریافت به صورت همزمان می تواند صورت بگیرد.

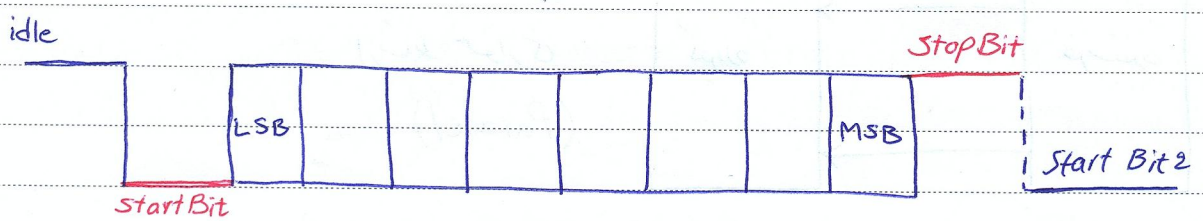
- میکروکنترلرهای AVR دارای ارتباط سریال از نوع Full-Duplex می باشد.



1391.3.16

- ارتباط سریال به دو صورت سنکرون و آسنکرون انجام می شود.
- در روش سنکرون در هر بلوک می توان تعدادی داده و یا کاراکتر ارسال کرد. امّا در روش آسنکرون در هر ارسال یک داده یا کاراکتر ارسال می شود.
- برای ارسال هر داده یا کاراکتر، یک بلاک Block (یا Frame) داده ارسال می شود که به صورت زیر است:

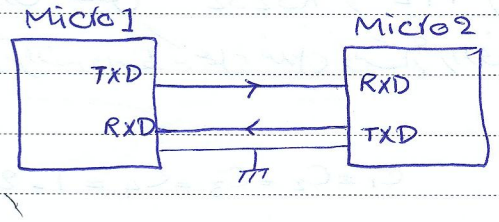
1- بیت شروع (Start Bit) که این بیت صفر در نظر گرفته می شود.
2- بعد از بیت شروع داده ارسال می شود. از LSB تا MSB.
3- بیت پایان ارسال می شود. (Stop Bit) که تعداد بیت های پایان 1 یا 2 می باشد.



PAPCO
www.mohandesyar.com

اتصال دو میکرو:

- برای ارسال اطلاعات، از معیار Baud Rate یا نرخ ارسال جهت تعیین سرعت استفاده می‌شود که برحسب bps بیان می‌شود که بیانگر تعداد بیت‌های ارسالی در هر ثانیه است.
- بایستی نرخ ارسال در فرستنده و گیرنده به صورت مساوی تنظیم شود، در غیر این صورت اطلاعات به درستی منتقل نمی‌شود.



8051 → UART

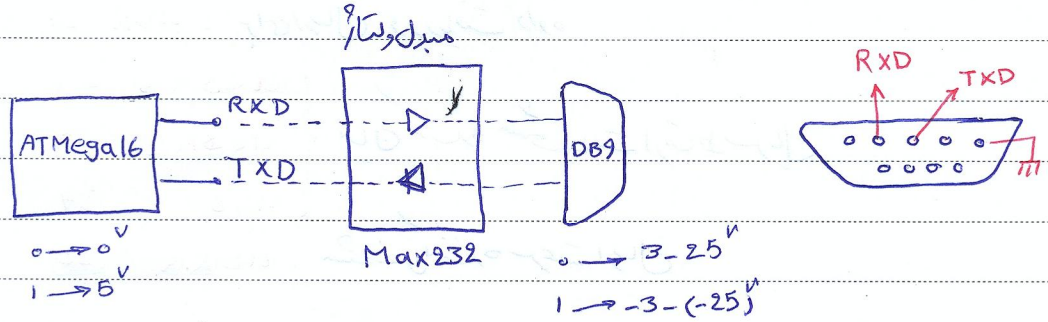
- ارتباط سریال در میکروکنترلرهای AVR از نوع USART می‌باشد:

(Universal Synchronouse Asyn. Recieve Transmit)

- این پروتکل برای ارسال از میکرو به کامپیوتر و بالعکس تعریف شده است و دارای خواص زیر می‌باشد:

1- ارتباط دوطرفه Full Duplex
2- ارتباط سنکرون و آسنکرون
3- ارسال داده به صورت 5، 6، 7، 8 و یا 9 بیتی.
4- تولید و ارسال بیت توازن (Parity) به صورت زوج یا فرد.
5- ارسال بیت پایان (یک یا دو بیت)
6- ارسال وقفه به میکرو در هنگام ارسال یا دریافت

- برای برقراری ارتباط سریال میان میکرو و کامپیوتر از استاندارد RS232 استفاده می‌شود.
- در کامپیوتر، کانکتوری ۹ پین (DB9) وجود دارد که با منطق RS232 کار می‌کند.
- در استاندارد RS232 ، صفر منطقی معادل $3 \to 25^V$ و یک منطقی معادل $-3 \to -25^V$ است.



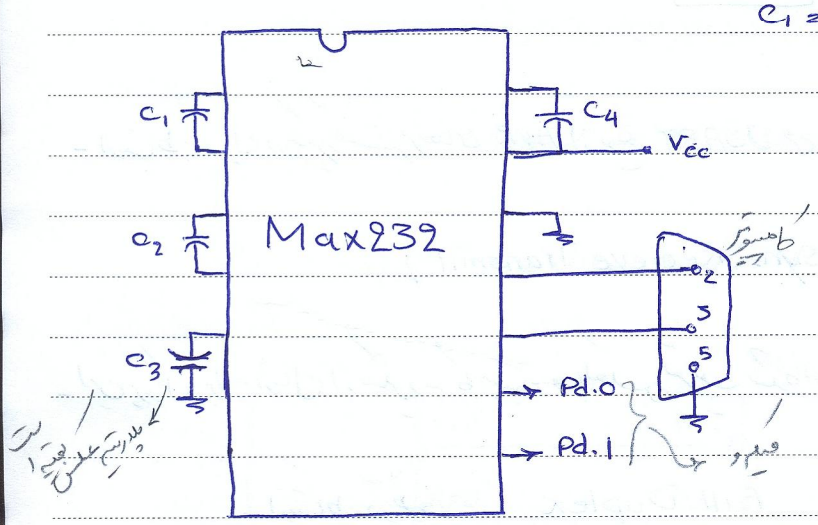
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

- در کانکتور DB9 پایه 2 پایه RXD و پایه 3، TXD می باشد.
- در میکرو پایه Pd.0 مربوط به دریافت (RXD) و پایه Pd.1 مربوط به ارسال (TXD) است.

- به علت یکسان نبودن منطق RS232 و TTL (متفاوت بودن سطح ولتاژ)، برای اتصال میکرو به کامپیوتر بایستی از ترانسه واسط که عمل تبدیل ولتاژ را انجام می دهد، استفاده کرد.

$C_1 = C_2 = C_3 = C_4 = 1 - 22\mu F$



- بلاک دیاگرام ارتباط سریال دارای 3 بخش اصلی می باشد:

1- پالس ساعت: مشخص کننده سرعت ارسال و دریافت است.
2- قسمت گیرنده (RXD)
3- قسمت فرستنده (TXD)

- رجیسترهای پورت سریال: پورت سریال دارای رجیسترهای زیر می باشد:

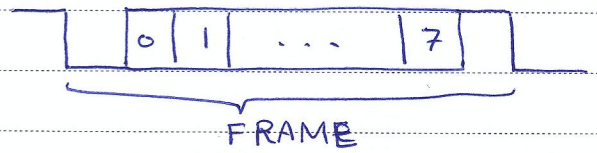
1- UDR: برای ارسال و دریافت داده
2- UCSRA
3- UCSRB
4- UCSRC
} بیان کننده مشخصات ارتباط سریال
5- UBRR: مشخص کننده سرعت ارسال

PAPCO
www.mohandesyar.com

Subject:
Year. Month. Date. ( )

- چگونگی ارسال در ارتباط سریال:

- برای ارسال داده 8 بیتی، ابتدا داده در رجیستر UDR نوشته می‌شود، سپس به رجیستر شیفت دهنده انتقال پیدا می‌کند و داده به صورت بیت به بیت تبدیل می‌شود و از طریق پایه TXD ارسال می‌شود.



- نحوه دریافت در ارتباط سریال:

- در گیرنده، اطلاعات به صورت بیت به بیت در رجیستر شیفت دهنده دریافت شده و پس از دریافت کامل فریم داده، اطلاعات در رجیستر UDR ذخیره می‌شود.

۱- رجیستر ورودی - خروجی داده: (UDR) Universal I/O Data Register

این رجیستر برای ارسال و دریافت داده استفاده می‌شود. این رجیستر دارای دو بافر RXB و TXB است.

در RXB اطلاعات دریافتی ذخیره می‌شود.
در TXB اطلاعات ارسالی نوشته می‌شود.

PAPCO
www.mohandesyar.com

ثبات های پورت سری USART در میکرو کنترلرهای AVR :

۱- ثبات داده ورودی خروجی UDR (USART I/O Data Register)

- این ثبات، برای دریافت و ارسال اطلاعات است:

- دارای یک بافر ارسال TxB است که اطلاعات ارسالی در آن نوشته می‌شود.
- " " " دریافت RxB " " دریافتی " ذخیره می‌گردد.

7	6	5	. . .	2	1	0	
RxB [7:0]							UDR
TxB [7:0]							

* به محض اینکه اطلاعات در ثبات UDR نوشته شد، در صورتیکه قسمت فرستنده USART فعال باشد، اطلاعات مذکور به ثبات شیفت دهنده ارسال، منتقل می‌شود و از طریق پایه TXD به صورت سری، (یک بیت یک بیت) ارسال می‌گردد.

۲- ثبات های کنترل و وضعیت UCSRA، UCSRB و UCSRC :

این ثبات ها برای کنترل و مشخص نمودن طرز کار پورت سری به کار برده می‌شود. مثلاً با این ثبات ها می‌توان USART را در حالت سنکرون، آسنکرون قرار داد، تعداد بیت های تبادل اطلاعات را مشخص کرد، وقفه را فعال یا غیرفعال کرد و ...

۲-۱- ثبات کنترل وضعیت UCSRA (USART Control and Status Register A)

این ثبات شامل بیت های پرچم دریافت و ارسال و بیت های پرچم وقوع خطا است.

7	6	5	4	3	2	1	0	
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA

* بیت ۷ - بیت پرچم دریافت کامل کاراکتر (RXC) (USART Reciever Complet)

زمانی که اطلاعات سری به بافر دریافت ثبات UDR رسید، و در آن قرار گرفت، آن وقت بیت پرچم دریافت اطلاعات کامل یک کاراکتر برابر ۱ می‌شود. یک شدن RXC به معنای حاضر بودن اطلاعات دریافتی در UDR است.

* اگر اطلاعاتی در ثبات UDR نباشد، بیت پرچم RXC برابر صفر است.

www.mohandesyar.com

* بیت (۶) : بیت پرچم پایان ارسال کاراکتر TXC (USART Transmit Complet)

هنگامی که یک کاراکتر در ثبات بافر UDR ، از طریق پایه TXD ارسال شده، آنگاه بیت پرچم ارسال TXC = ۱ می شود که نمایانگر پایان ارسال کاراکتر است.

* بیت (۵) : بیت پرچم خالی بودن ثبات داده : UDRE
USART Data Register Empty

زمانی که بیت UDRE برابر یک شود، نشانه این است که ثبات بافر ارسال UDR خالی است و آماده دریافت اطلاعات جدید است.

* بیت (۴) : بیت پرچم خطای FE : (Frame Error)

موقعی که بیت پایان در انتهای کاراکتر برابر ۰ باشد، یعنی اطلاعات یک بایت تکمیل نشده

لذا بیت خطای FE = ۱ می شود.

** زمانی که اطلاعاتی در ثبات UCSRA نوشته می شود، این بیت صفر نوشته می شود.

* بیت (۳) : بیت پرچم خطای DOR : (Data over Run)

زمانی که بافر دریافت پر باشد و کاراکتر دیگر بخواهد ارسال شود، یعنی بیت شروع صفر شود،

در این صورت DOR = ۱ می شود.

** موقعی که اطلاعاتی در ثبات UCSRA نوشته می شود، این بیت صفر نوشته می شود.

* بیت (۲) بیت پرچم توازن PE (Parity Error)

اگر بیت توازن دریافتی اشتباه باشد، PE برابر یک می شود

در موقع نوشتن در UCSRA، در این بیت بایستی صفر نوشت.

* بیت (۱) : بیت دو برابر کردن سرعت ارسال : Double the USART transmission Speed

با یک کردن این بیت سرعت ارسال، ۲ برابر می گردد.

* در حالت سنکرون این بیت صفر است.

* بیت (۰) : حالت مولتی پروسسور : Multi - Processor Communication Mode

وقتی این بیت یک شود، یعنی حالت مولتی پروسسور استفاده می شود.

www.mohandesyar.com

۲-۲) ثبات کنترل وضعیت UCSRB :

این ثبات برای فعال کردن وقفه و فعال کردن فرستنده و گیرنده است :

7	6	5	4	3	2	1	0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

* بیت (۷): بیت فعال کردن وقفه دریافت RXCIE

Rx Complete Interrupt Enable.

با یک کردن این بیت، در صورتیکه بیت پرچم دریافت RXC در ثبات UCSRA برابر یک باشد، وقفه کلی فعال شده باشد، آن وقت وقفه دریافت اطلاعات فعال می‌شود.

* بیت (۶) : بیت فعال کردن وقفه ارسال TXCIE

با یک کردن این بیت، در صورتیکه بیت پرچم TXC در ثبات UCSRA برابر یک باشد، و وقفه کلی فعال شده باشد، آن وقت وقفه ارسال اطلاعات فعال می‌شود.

* بیت (۵) : بیت فعال کردن وقفه در حالت خالی بودن ثبات داده UDRIE :

با یک کردن این بیت، اگر بیت پرچم ~~UDRE~~ خالی بودن ثبات داده (UDRE) در ثبات UCSRA برابر یک باشد و وقفه کلی فعال شده باشد آنگاه وقفه خالی بودن ثبات داده فعال می‌باشد.

* بیت (۴) : بیت فعال کردن گیرنده RXEN (Rx Enable)

برای فعال کردن قسمت گیرنده پورت سری USART بایستی RXEN را برابر یک کرد.

در این صورت پایه RXD، آماده دریافت اطلاعات می‌شود.

* بیت (۳) : بیت فعال کردن فرستنده TXEN

بیت فعال کردن قسمت فرستنده بایستی TXEN را برابر یک کرد.

در این حالت پایه TXD، آماده ارسال می‌شود.

* بیت (۲) : تعداد بیت‌های یک کاراکتر UCSZ2 (Character Size)

بیت UCSZ2 و UCSZ1 و UCSZ0 در ثبات UCSRC تعداد بیت‌های کاراکتر را مشخص می‌کند.

* بیت (۱) : نهمین بیت دریافت RXB8

هنگامی که تبادل اطلاعات ۹ بیتی انتخاب شود، در این صورت بیت RXB8 نهمین بیت کاراکتر دریافتی خواهد بود.

* بیت (۰) : نهمین بیت ارسال TXB8

اگر تبادل اطلاعات ۹ بیتی باشد، TXB8، نهمین بیت کاراکتر ارسالی خواهد بود.

www.mohandesyar.com

۲-۳) ثبات کنترل و وضعیت UCSRC :

با این ثبات، سنکرون یا آسنکرون بودن، تعداد بیت ارسالی، فعال کردن بیت توازن، را مشخص می‌کنیم :

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

* بیت (۷) : انتخاب ثبات : (USART Mode Select)

زمانی که با ثبات UCSRC کار می‌کنیم، این بیت را برابر یک قرار می‌دهیم.

* بیت (۶) : انتخاب حالت سنکرون یا آسنکرون :

UMSEL ← { ۰ ← آسنکرون
۱ ← سنکرون .

* بیت (۴ و ۵) : انتخاب حالت بیت توازن :

UPM1	UPM0	حالت بیت توازن
۰	۰	غیر فعال
۰	۱	زوج شده
۱	۰	بیت توازن زوج فعال شود
۱	۱	" " فرد .

اگر این بیت فعال شود، فرستنده، صورت اتوماتیک، این بیت را همراه با کاراکترهای ارسال می‌کند، گیرنده بررسی کرده در صورت ارسال اشتباه، پرچم PE در رجیستر UCSRA را یک می‌کند.

* بیت (۳) : انتخاب تعداد بیت های پایان . (USART Stop bits Select)

USBS ← { ۱ ← دو بیت پایان
۰ ← یک " "

* بیت (۱ و ۲) . تعیین کننده تعداد بیت های کاراکتر :

UCSZ2	UCSZ1	UCSZ0	تعداد بیت S
۰	۰	۰	5
۰	۰	۱	6
۰	۱	۰	7
۰	۱	۱	8
۱	۱	۱	9

www.mohandesyar.com

* بیت (۵) : انتخاب پلاریته پالس در حالت سنکرون Clock polarity

این بیت در حالت سنکرون استفاده می‌شود . در حالت آسنکرون باید صفر باشد .

UCPOL	لحظه تغییر اطلاعات ارسالی روی پین فرستنده TXD	لحظه نمونه برداری اطلاعات ورودی گیرنده
۰	در لبه بالا رونده پالس ساعت XCK	در لبه پایین رونده پالس
۱	" " پایین رونده " "	" بالا " "

www.mohandesyar.com

- فرستنده پورت سری USART :

* فرستنده USART با فعال کردن بیت TXEN در ثبات UCSRB فعال می‌شود، و در این صورت پایه TXD میکروکنترلر برای پورت سری خروجی می‌گردد.

* قبل از ارسال اطلاعات باید به ثبات های پورت سری مقدار اولیه داده شود.

- دریافت اطلاعات توسط گیرنده :

* با یک کردن بیت فعال ساز RXEN در ثبات کنترل UCSRB، گیرنده پورت سری USART فعال می‌شود، در این صورت پایه به صورت اتوماتیک پایه RXD میکروکنترلر، برای ورودی اطلاعات سری USART تعریف می‌شود.

* قبل از دریافت یا ارسال اطلاعات، باید با مقدار اولیه دادن به ثبات ها، سرعت ارسال اطلاعات (B.R)، تعداد بیت ها و ... را تعیین کرد.

* تولیدکننده پالس داخلی : Internal clock Generation : The Band Rate Generator

پالس ساعت میکروکنترلر OSC، به قسمت تولیدکننده پالس تبادل اطلاعات یا Band Rate داده می‌شود و متناسب با مقداری که در ثبات کنترل UBRR قرار می‌دهیم، پالس ساعت تبادل اطلاعات مطابق فرمول زیر تولید می‌شود.

$$\text{Band Rate} = \frac{f_{osc}}{16(UBRR+1)}$$

f_{osc} : فرکانس اسیلاتور

UBRR : مقداری که در ثبات UBRR قرار می‌دهیم.

مثلاً اگر $f_{osc}=1^{MHz}$ باشد و UBRR = 25 باشد خواهیم داشت

$$\text{Band Rate} = \frac{1^{M}}{16(25+1)} \simeq 2400\ Hz$$

یعنی سرعت تبادل اطلاعات، $2400^{bit/s}$ است.

* با یک کردن بیت U2X در ثبات کنترل UCSRA، سرعت ارسال و دریافت ۲ برابر می‌شود یعنی:

$$\text{Band Rate} = \frac{f_{osc}}{8(UBRR+1)}$$

* با استفاده از جدول مربوط، با داشتن سرعت های مختلف پورت سری (Band Rate های متفاوت مانند 2400، 4800، 9600 و ...) و با داشتن فرکانس پالس ساعت ۱ و 2، 4، 8 ^{MHz} مقدار ثبات UBRR تعیین می‌شود و در آن قرار داده می‌شود.

www.mohandesyar.com

۳- ثبات تعیین سرعت پورت سری UBRR :

ثبات ۱۶ بیتی UBRR ، دارای یک بایت کم ارزش UBRRL و یک بایت پرارزش UBRRH است :

15	14	13	12	11	10	9	8	
URSEL	-	-	-	UBRR[11:8]				UBRRH
UBRR [7:0]								UBRRL
7	6	5	4	3	2	1	0	

مطابق با جدول مربوطه با مقدار دهی ثبات UBRR ، B.R. ارسال مشخص می‌شود .

* بیت (۱۵): بیت انتخاب ثبات (Register Select)

با این بیت می‌توان ثبات UBRRH یا UCSRC را انتخاب نمود :

وقتی که اطلاعات از ثبات UBRRH خوانده می‌شود ، این بیت برابر صفر است .

در ثبات " نوشتن " " " ۱ "

* بیت (۱۱) تا (۰) : ثبات سرعت پورت سری :

بیت های ۰ - ۱۱ یک ثبات ۱۲ بیتی است که مشخص کننده Baud Rate است .

ثبات UBRRH شامل ۴ بیت پرارزش و ثبات UBRRL شامل ۸ بیت کم ارزش است .

مثلاً اگر $f_{OCS}=8MHz$ و Baud Rate=9600 باشد ، مطابق جدول بایستی UBRR=51 یا 0x33

انتخاب شود که در این حالت UBRRH=0x00 و UBRRL=0x33 است .

www.mohandesyar.com

* برنامه‌ای بنویسید که در میکروکنترلر AVR ،

• پورت سری USART اعداد 0x77 و 0x88 را در هر 500ms از پایه TXD میکرو ارسال کند.

• اگر عدد 0x55 در پایه RxD میکرو رسید، بیت 6 پورت D را یک کند و اگر عدد 0x66 رسید بیت 6 پورت D را صفر کند. (۸ بیتی آسنکرون - توازن نداریم و یک S.B داریم)

$f_{xtal} = 8M$

$B.R = 56000$

مقداردهی رجیستر ها

1) UCSRA = 0x00

2)

7			4				
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	~~TXB8~~
1	0	0	1	1	0	0	0

(7) RXCIE = 1 ← اگر بیت پرچم RXC در ثبات UCSRA برابر یک شود وقتی سرویس وقفه دریافت و زیر برنامه وقفه اجرا می‌شود (اگر 0x55 دریافت شد، پورت D را یک کرده و اگر 0x66 رسید، صفر می‌کند)

4 و 3 ← فعال کردن فرستنده و گیرنده.

UCSRB = 0x98

3) UCSRB = 0x86

URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
1	0	0	0	0	1	1	0

URSEL ← 1 ← کار با ثبات UCSRC

UMSEL ← 0 ← آسنکرون

UPM1 و UPM0 ← 0 و 0 ← بیت توازن نداریم

UCSZ1 و UCSZ0 = 1 و UCSZ2 = 0 ← ۸ بیتی

USBS = 0 ← یک S.B داریم.

4) $56000 = \frac{8 \times 10^6}{16\,(UBRR+1)} \longrightarrow UBRR \simeq 8$

$\Rightarrow \begin{cases} UBRRH = 0x00 \\ UBRRL = 0x08 \end{cases}$

www.mohandesyar.com

```
# include <mega8.h>
# include <delay.h>
# include <stdio.h>
interrupt [USART_RXC] void USART_rx_isr(void)
{
  char data;
  data = UDR;
  if (data =0x55)
        portd.6 =1;
  if (data = 0x66)
        port d.6 =0;
}

void main, void {

DDRD = 0x40 ;
UCSRA = 0x00;
UCSRB = 0x98;
UCSRC = 0x86;
UBRRH = 0x00;
UBRRL = 0x08;

# asm ("sei")
  while (1) {
    UDR = 0x77;
    delay_ms (500);
    UDR = 0x88
  } delay_ms (500)

}
```

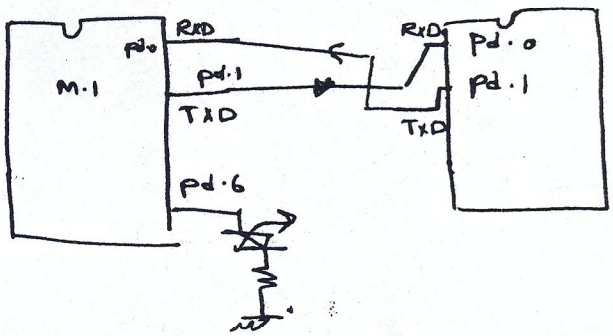
A

B

C

* حلقه while مرتباً تکرار می‌شود، تا وقفه رخ دهد.
در قسمت C، مرتباً 0x77 و 0x88 ارسال می‌گردد.

* اگر در این مدت اطلاعاتی از میکرو دیگر به پایه ورودی دریافت RXD برسد، روتین سرویس وقفه یعنی بخش A اجرا می‌شود.



www.mohandesyar.com

ثبات داده ورودی - خروجی RXB, TXB } ← UDR

در سنکرون صفر است.

دو برابر کردن سرعت

Recieve Complet — Data Register Empty — Data Over Run

RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

UCSRA

Transmit Complet — Frame Error — Parity Error — مولتی پروسسور

پرچم ارسال و دریافت و خطا

وقفه دریافت — وقفه ارسال — فعال کردن گیرنده — فعال کردن فرستنده — بیت ۹ دریافت — بیت ۹ ارسال

RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

UCSRB

UDR وقفه خالی بودن

کاراکتر سایز

فعال کردن وقفه و فعال کردن ارسال و دریافت

UCSZ2 → UCSZ0

انتخاب UCSRC

0 → آسنکرون
1 → سنکرون

1 → دو بیت پایان
0 → یک بیت پایان

سنکرون

انتخاب پلاریتی پالس در حالت

بیت توازن

URSEL	UMSEL	UPM1	UPMØ	USBS	UCSZ1	UCSZØ	UCPOL

UCSRC

UPM1	UPMØ	توازن
0	0	غیرفعال
0	1	زوج
1	0	توازن زوج
0	1	توازن فرد

UCSZ2	UCSZ1	UCSZØ	تعداد بیت
0	0	0	5 بیتی
0	0	1	6 بیتی
0	1	0	7 بیتی
0	1	1	8 بیتی
			—
			—
			—
1	1	1	9 بیتی

1 → 0
2 → 1
3 → 2
15 → 7

0 → بالارونده
1 → پایین رونده

15 — 11 — 10 — 9 — 8

URSEL	—	—	—		UBRR[11:8]		UBRRH
UBRR[7:0]							UBRRL

U2X → 0 → $B.R = \frac{f_{osc}}{16(UBRR+1)}$

U2X → 1 → $B.R = \frac{f_{osc}}{8(UBRR+1)}$

www.mohandesyar.com

$$f_{CTC} = \frac{f_{xtal}}{2 \times Pre\,(1 + OCR\emptyset)}$$

CTC فرکانس

asm ("sei")

FOC0	WGM01	COM01	COM00	WGM01	CS02	CS01	CS00

TCCR$\emptyset$

CS02	CS01	CS00	
0	0	0	0 → timer Stopped
0	0	1	1 → Pre = 1
0	1	0	2 → Pre = 8
0	1	1	3 → Pre = 64
1	0	0	4 → Pre = 256
1	0	1	5 → Pre = 1024
1	1	0	6 → کانتر ↑
1	1	1	7 → کانتر ↓

PWM

COM01	COM00	
0	0	تایمر در حالت Normal
0	1	CTC (toggle)
1	0	Non-Inverting
1	1	Inverting

WGM01	WGM00	حالت تایمر	Max
0	0	Normal	0xFF
0	1	ph.Cor. PWM	0xFF
0	0	CTC	OCR$\emptyset$
1	1	Fast PWM	0xFF

$= \frac{f_{xtal}}{Pre \times 510}$

$= \frac{f_{xtal}}{Pre(256 - TCNT\emptyset)}$

Pb.3 = out

COM01	COM00	
0	0	Normal
0	1	CTC (toggle) not
1	0	CTC (Clear)
1	1	CTC (Set)

```
Interrupt [tim0_Comp] Void timer0_Comp_isr(Void)
{
//Codes
}
```

تایمر 2 — تایمر صفر

COIE2	TOIE2	IOC1	COIE1B	COIE1A	TOIE1	COIE$\emptyset$	TOIE$\emptyset$

TIMSK

وقفه

سرریز CTC — سرریز تایمر صفر (Normal)

```
Interrupt [tim0_ovf] Void timer0_ovf_isr(Void)
{
// Codes
}
```

تایمر یک

پرچم سرریز CTC — پرچم سرریز Normal

OCF2	TOV2	TCF1	OCF1B	OCF1A	TOV1	OCF$\emptyset$	TOV$\emptyset$

TIFR

پرچم

www.mohandesyar.com

$$f_{clk\ timer} = \frac{f_{xtal}}{Pre\ Scale}$$

تایمر صفر $\longrightarrow$ $\frac{0xFF - TCNT0}{t_{clk\ Timer}} =$ تعداد شمارش

زمان یک بار سرریز شدن $\longrightarrow$ تعداد شمارش $\times T_{clk\ timer}$

$$\text{تعداد دفعات OVF} = \frac{\text{زمان مطلوب}}{\text{زمان شمارش}}$$

مثال: ساخت یک ثانیه با تایمر صفر

$$f_{clk\ timer} = \frac{16\ MHz}{1} = 16\ MHz$$

$$T = 0.0625\ \mu s$$

$$1^s = 4 \times 250^{ms} \quad TCNT0 = 56$$

$$\text{تعداد شمارش} = FF - 56 = 200$$

$$\text{زمان یک بار سرریز} = 200 \times 0.0625 = 12.5\ \mu s$$

$$\text{تعداد دفعات OVF} = \frac{1^s}{12.5\ \mu s} = 80,000$$

www.mohandesyar.com