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.

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

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

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

۲- delay_us( ) : ایجاد تأخیر به اندازه آرگومان ورودی بر حسب us.

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

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

پورت های ورودی - خروجی (I/O):

- هر یک از پورت‌های 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 20mA (Max) جریان کشید.
- برای هر پورت در حافظه SRAM، 3 رجیستر وجود دارد:

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

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

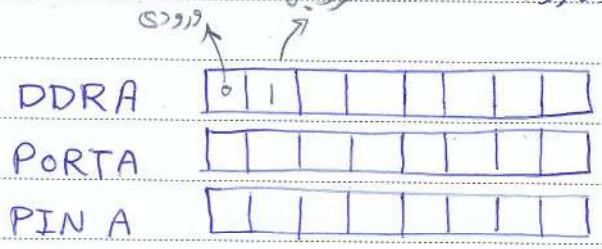
با مقدار دادن به DDRX، می‌توان ورودی یا خروجی بودن پورت را مشخص کرد.

۱ ← خروجی
۰ ← ورودی

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 PORTXn می توان مقاومت Pull up را فعال کرد.

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

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

Subject:
Year. Month. Date. ( )

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

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

PORTB ~= PORTB

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;   → ورودی A Pullup
DDRB = 0xFF;  → خروجی B
while (1){
    a = PINA;
   If (a==5){
      PORTB = PINA;
      delay_ms (1000)
     }
  }
}
```

اگر عدد زوج یا بگذارد:

① تقسیم بر 2

② آخرین بیت پورت چک شود. (LSB)

Subject:
Year. 1391 Month. 2 Date. 5 ( )

جلسه 7

مثال: تابعی بنویسید که بیت صفر پورت D را بخواند. در صورتیکه برابر یک بود، 6 بیت یک پورت D را 1 s 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;
    DDRD.1 = 1;
    While (1){
        Blink();
    }
}
```

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

Subject:
Year. Month. Date. ( )

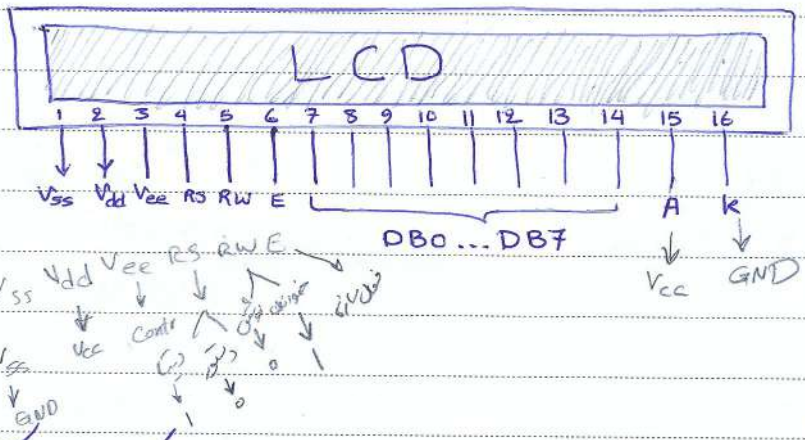
نمایشگر LCD:

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 MC
5- RW : مشخص می کند که اطلاعات بر روی LCD نوشته می شود یا خوانده می شود. → to MC
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.
DB - 7to14 : با صفر شدن E، داده و دستور در پایه DB0 - DB7 قرار می گیرد.
15- A : آند (نزدیک ترین پایه LCD)
16- k : کاتد (" " " LCD)

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]

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

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)

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

```
# asm
    equ_LCD_Port=0x15 ; Portc
# 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>

Subject:
Year. Month. Date. ( )

کاراکترها ← ROM
حافظه موقت ← DRAM
کاراکترهای ایجاد شده ← RAM
LCD Memory

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 کتابخانه <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 ستون های 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- سطر ← ستون (از صفر شروع می‌شود)
```
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

Subject:
Year. Month. Date. ( )

ASCII

LCD_Putchar( ) ← کد اسکی مورد نظر

مثال: LCD_Putchar("A") ← کد اسکی دهدهی

5- برای چاپ کردن کاراکتر بر روی LCD است.

itoa (A, B) معادل اسکی A را در B قرار می دهد. int → ascii

LCD_Puts( ) 6- برای چاپ رشته ذخیره شده در حافظه SRAM استفاده می شود.

LCD_Putsf( ) 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 می شود.

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

Itoa → تبدیل عدد Int به اسکی
Ltoa → " " Long " "
} → در کتابخانه استاندارد <stdlib.h> قرار دارند.

stdlib( )

مثال) Itoa(Value, Value1) مقدار Value چاپ می شود

LCD_Putsf(Value1) عبارت Value چاپ می شود

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
wizard  .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);
}
}
```

Subject:
Year. Month. Date. ( )

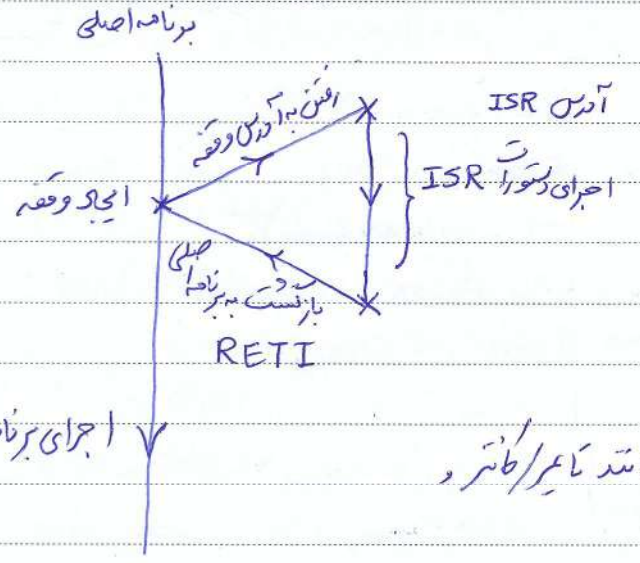
مثال) برنامه ای بنویسید که کلمه Micro را 5 بار به راست شیفت دهد.

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

وقفه (Interrupt):

برنامه میکرو خط به خط اجرا می‌شود. با ایجاد وقفه در برنامه ابتدا 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 ...

در کدوین برنامه برای تایمر یا کانتر فقط در حالت عادی است و بقیه قسمت ها یکی است.

Subject:
Year. 1391 Month. 8 Date. 20 ( )

- هریک 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 می 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 1) 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) $f_{Xtal} = 1\ MHz \longrightarrow T = 1\mu s$

کلاک خارجی ... Counter $t = 100\mu s \longrightarrow$ تعداد شمارش $= 100$

(2) افزایش تعداد شمارش

$f_{xtal} = 1\ MHz$, $t = 800\ \mu s$ → prescale = 8 → $f_{ClkTimer} = \frac{f_{IO}}{Ple.} = \dots$

clk IO → clkIO → Ple → Counter → Time

تعداد شمارش $= 100$

$8\mu s$

PaPCo

Subject :
Year . Month . Date . ( )

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

Prescale = 1, 8, 64, 256, 1024

- اگر هدف شمارش پالس باشد:

حالت‌های کاری تایمر/کانتر:

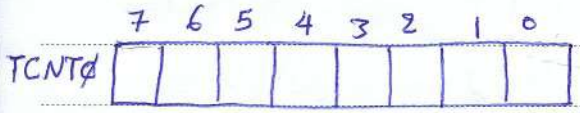
تایمر ۱:
1 - Normal Mode → تایمر صرف
2 - Fast PWM
3 - phase Correct PWM
4 - clear Time on Compare Match (CTC) ← CTC
یک مقدار خاص مقایسه می‌شود در صورت برابری مقدار تایمر/کانتر صفر می‌شود.

تایمر ۰ و ۲:
Normal Mode
CTC

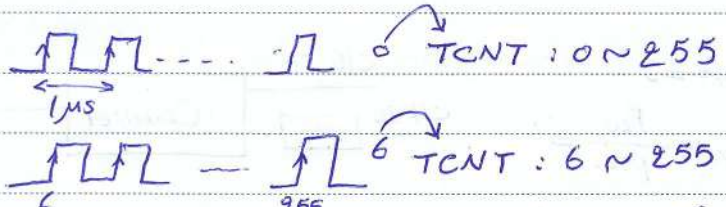
رجیسترهای تایمر/کانتر صفر:

۱- رجیستر TCNT0 (Timer Counter Register)

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



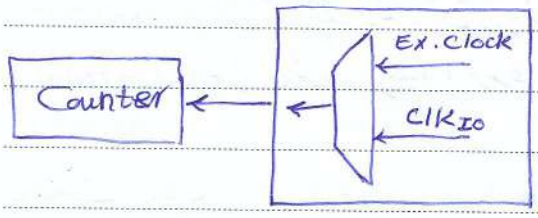
- مقدار اولیه رجیستر TCNT0 صفر می‌باشد که می‌توان به TCNT0 مقدار اولیه داد که تایمر از مقدار اولیه تا Max TCNT0 می‌شمارد.



- می‌توان در هر لحظه TCNT0 را مقداردهی کرد و یا خواند.

Subject:
Year. Month. Date. ( )

(Timer Counter Controll Register) TCCR0 _2



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

	7	6	5	4	3	2	1	0
TCCR0						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 هریک از بیت های CS00 → CS02 می توان تایمر را با کلاکی متفاوت راه اندازی کرد.
- در حالت آخر پالس خارجی متصل به Pb.0 با لبه بالا رونده یا پایین رونده شمارش می شود.

مدهای کاری تایمر: { مد نرمال / CompareMatch

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

CTC: Clear time on Compare Match

- حالت CTC: 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 فعال سازی 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	ICIE1	OCIE1B	OCIE1A	TOIE1	OCIE0	TOIE0	TIMSK

Timer OverFlow Interrupt Enable

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

تایمر صفر جا

- بیت صفر رجیستر TIMSK (TOIE0): با یک کردن بیت صفر وقفه مربوط به تایمر صفر فعال می‌شود یعنی در صورت سرریز شدن تایمر و برخاستن OVF، زیربرنامه وقفه مربوط به تایمر صفر اجرا می‌شود. (پرچم سرریز وقفه پس اجرای زیربرنامه به صورت خودکار صفر می‌شود.)

Compare OverFlow Interrupt Enable. (OCIE0): TIMSK رجیستر

- بیت یک رجیستر 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 مربوط 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 و دو است.

TCNT , TCCR , TIMSk , TIFR
(مقدار دهی) (رجیستر کنترلی) (فعال‌سازی وقفه) (Flag)

Subject:
Year. Month. Date. ( )

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

Capture

بیت‌های این رجیستر توسط خود میکرو مقداردهی می‌شود.

- بیت صفر TIFR : (TOVØ)

TOVØ : Timer Overflew Flag

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 وقفه‌های سراسری و وقفه‌های مربوط به تایمر صفر، زیربرنامه ISR اجرا می‌شود. و پس از اجرای کامل زیربرنامه مقدار TOVØ=0 شده و برنامه اصلی اجرا می‌شود.

- بیت یک TIFR : (OCFØ)

OverFlew Compare flag

در صورتی که مقدار تایمر با عدد Compare برابر شود و وقفه‌های مربوط به تایمر صفر، فعال شده باشند این Flag از صفر به یک تغییر وضعیت داده و زیربرنامه مربوط به Interrupt اجرا می‌شود. پس از اجرای زیربرنامه مربوط به وقفه، این پرچم به صورت خودکار صفر می‌شود.

5- رجیستر کنترلی خروجی : (OCR)

OCR : Output Control Register

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

نکته: علت اصلی استفاده از وقفه این است که اکثر از روش معمولی (بدون وقفه) یا همان روش سرکشی استفاده نکنیم بخواهیم یک دستور شرطی قرار دهیم تا مقدار تایمر چک شود. با توجه اینکه دستورات دیگری از میکرو می‌شود، 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 نمی‌شود. دقت روش سرکشی کم است.

Timer=250
TCNT=255
IF (TCNT=250)
شرط نادرست می‌شود

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

6 — 250ms
TCNT=0 — X — Max=255ms
مقدار اولیه TCNT
OVF (TOVØ=1)

Subject:
Year. Month. Date. ( )

تنظیمات تایمر در CodeWizard:

Timer → Timer ∅

1- 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.

$f_{ClkIo} = 1\ MHz$
Clk Value = 1024
$\Big\}\ f_{clk\ timer} = \frac{1\ MHz}{1024} \approx 0.997\ kHz \rightarrow T \simeq 1ms$

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 زیر برنامه ای به برنامه اصلی اضافه می‌شود. (زیر برنامه وقفه تایمر/کانتر).

7- CMP Int:

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

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

P4PCO

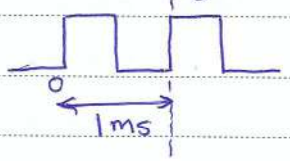
Subject:
Year. Month. Date. ( )

- برای فعال کردن وقفه سراسری با استفاده از دستور زیر به برنامه اضافه کرد:

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

$250\ ms = t_d$

:6



مثال: برنامه ای بنویسید که هر بار تایمر صفر Max شده، 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

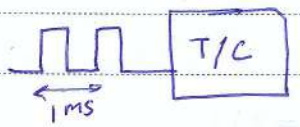
طلاب تایمر (Pre=1024)

فعال سازی وقفه تایمر 0

وجود حلقه while الزامی است

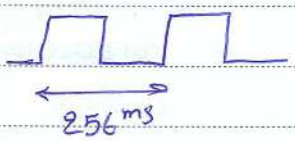
$f_{Clk\ Timer} = \frac{1\ MHz}{1024} \cong 1\ kHz$

$T = 1\ ms$



$t_d = 256\ ms$

Port0A.0 خروجی →



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;
}
```

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 = 0x01 ; PORTA = 0x00 ; // A.0 = out

TCCR0 = 0x05 → Prescale = 1024

TIMSk = 0x01 → Timer OVF Interrupt Enable

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

برای طراحی تایمر:

1- $f_{Timer} \leftarrow$ Pre., f_{xtal}

2- TCNT0

3- تعداد دفعات OVF

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

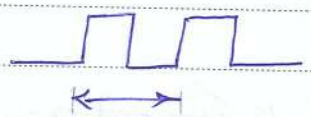
$f_{xtal} = 16$ MHz

Pre Scale = 1

255 − tCNT0

مثال: برنامه‌ای بنویسید که هر 1 ثانیه یک واحد به متغیر a اضافه کند.

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



0.0625 µs

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

$\frac{\text{زمان مطلوب}}{\text{زمان شمارش}} =$ تعداد OVF ← تعداد شمارش $t_{clk\,timer}$

TCNT0 = 56 مقدار اولیه

56 ——— 255

12.5 µs

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

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

Subject:
Year. Month. Date. ( )

```
#include <mega16.h>
Interrupt [tim0_ovf] void timer0_ovf_isr(void)
 {
  a++;
  if (a==80000){
     b++;
     a=0;
  }
  TCNT0 = 0x38; \\ TCNT0=56
 }
Void main(void){
 Long a,b;
 TCNT0 = 0x38;  tCNT0=56; // 56 Decimal = 38 Hex
 TCCR0 = 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 ظاهر کند.

CS02, CS01, ES00 = 101 OR 111

```
#include <mega16.h>
Void main(void){
PORTB=0xFF;
DDRB=0x00;   } B ورودی, Pullup
DDRD=0xFF;   } D خروجی
TCNT0 = 0x00;
TCCR0 = 0x07;  کانتر و لبه پایین رونده
while (1){
  PORTD=TCNT0;
 }}
```

PAPCO

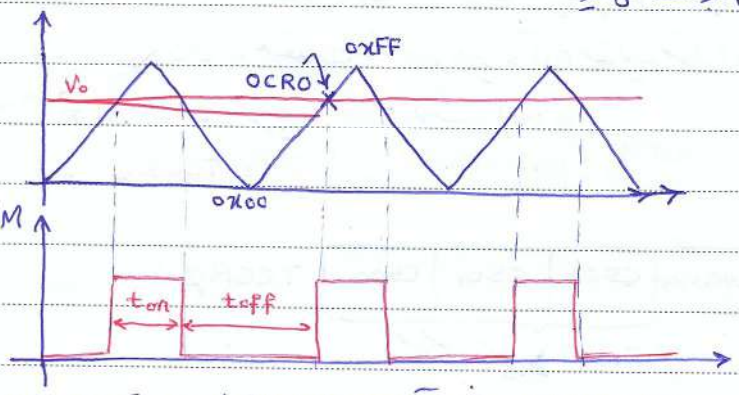
Subject:
Year. 1391 Month. 3 Date. 3 ( )

: PWM

PWM : Pulse Width Modulation

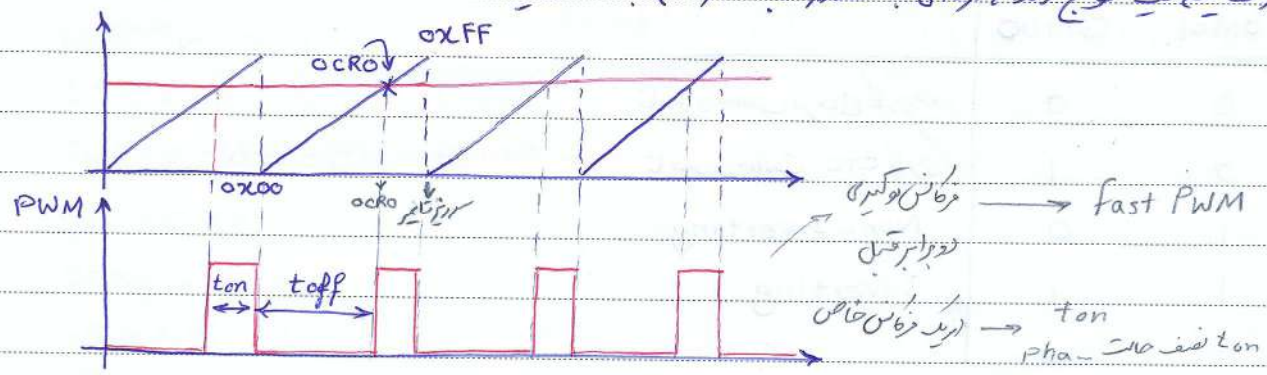
تولید موج PWM توسط تایمر صفر :

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



تولید PWM به روش تصحیح فاز Phase Correct PWM

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



Fast PWM تولید PWM

OCR0

- در حالت تصحیح فاز، تایمر به صورت صعودی و نزولی می‌شمارد، در این حالت هر لحظه مقدار تایمر (TCNT0) با مقدار OCR0 مقایسه می‌شود. هنگامی که مقدار تایمر برابر OCR0 (TCNT0 = OCR0) شد، پایه Pb3 (OC0) یک می‌شود، تایمر با مقدار Max می‌شمارد و سپس به صورت نزولی شروع به شمارش می‌کند، پایه Pb3 در مدت زمان مساوی شدن تایمر با OCR0 یک می‌ماند تا اینکه مقدار تایمر از OCR0 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 TCNT0 = OCR0 می‌شود، پایه Pb3 یک شده و تایمر تا Max می‌شمارد. به محض سرریز شدن تایمر Pb3 صفر می‌شود.

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

Subject:
Year. Month. Date. ( )

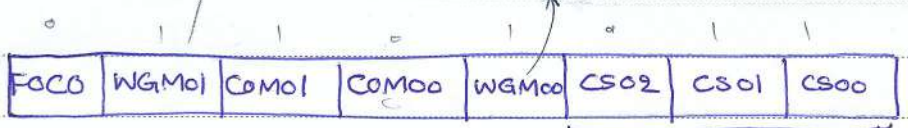
- اگر در حالتی که TCNTØ = OCRØ، پایه Pb3 یک شود، موج تولید شده Inverted نامیده می‌شود و اگر صفر شود Non-Inverted نامیده خواهد شد.

- 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 رابطه)

3- TCCRØ:



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

COM01	COMØØ	
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

- مشخص کننده نوع PWM تولیدی است: WGM01, WGM00

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