الباب السادس CHAPTER (6)

المؤقتات Timers

يحتوى الـ PIC18F452 على أربعة مؤقتات Timers والتى يمكن ان تستخدم لأغراض مختلفة مثل توليد نبضة كل فترة زمنية Generating timing signals والتى يمكن تستخدم مع الـ Interrupt لعمل أشياء مفيدةً مثل تنفيذ برنامج ما كل فترة زمنية. كما يمكن أن تستخدم أيضاً لعد النبضات القادمة من مصدر خارجى Counting external pulses وخاصة فى تطبيقات التحكم فى المواتير الكهربائية أو خطوط الإنتاج أو أنظمة الأتصالات الرقمية. ويجب أن تعرف فى المقام الأول أن الـ Timers التى بداخل الـ Microcontrollers ما هى إلا عبارة عن عدادت Counters ويمكن إستخدام هذه الـ Counters إما كـ Timers للأغراض المختلفة أو كـ Counters لعد النبضات الخارجية External pulses. ويحتوى الـ PIC18F452 على أربعة تايمرات وهم Timer0 و Timer1 و Timer2 و Timer3. ونظراً لأن تركيب الـ Timer3 مشابه جداً لتركيب الـ Timer1، فإننا فى هذا الباب سوف نتناول تركيب الـ Timer0 و Timer1 و Timer2 فقط. كما سوف نتعلم كيفية برمجة وإستخدام هذه الـ Timers. وكيفية إستخدام الـ Interrupt معهم لتنفيذ برنامج ما كل فترة زمنية.

6.1-Timer0

يمكن أن يعمل الـ Timer0 بنظامين إما بنظام 8-bits mode أو 16-bits mode. حيث أن الـ Counter الخاص به يمكن أن يعد من 0 إلى 255 إذا كان يعمل فى الـ 8-bit mode أو يعد من 0 إلى 65535 وذلك إذا كان يعمل فى الـ 16-bit mode. ويتم ضبط خصائص هذا التايمر عن طريق الـ T0CON register.

شكل (6-1) يوضح تركيب الـ Timer0. فى هذا الشكل يُلاحظ أنه يوجد Register يسمى الـ TMR0 وهو عبارة عن الـ Counter الخاص بالـ Timer0. كما يوجد قطعة تسمى الـ Prescaler والتى تستخدم فى المقام الأول للتحكم بسرعة عد الـ TMR0 counter. فعندما يتم ضبط الـ Prescaler على 1:2 فهذا يعنى أنه سوف يتم قسمة عدد النبضات القادمة للـ TMR0 على 2 . فعلى

سبيل المثال إذا تم ضبط الـ Prescaler على 1:2 وكان عدد النبضات الكلية الداخلة للـ Timer0 تساوى 100 pulse فإن TMR0 سوف يعد 50 عدة فقط. أما إذا تم ضبط الـ Prescaler على 1:4 فإن الـ TMR0 سوف يعد 25 عدة. كما يلاحظ فى نفس الشكل أنه يوجد Two Multiplexers. حيث أن الـ Mux الأول يستخدم للإختيار بين مصدر الـ Pulses (Internal أو External) التى سوف تعمل على زيادة قيمة الـ TMR0. أما الـ Mux الأخر فيستخدم لتحديد هل يريد المستخدم توظيف الـ Prescaler من عدمة، وذلك بسبب أن أقل قيمة للـ Prescaler هى 1:2 ولا يوجد به 1:1.

شرح الـ T0CON Register

TMR0ON bit:

يستخدم هذا الـ Bit لتشغيل وإطفاء الـ Timer0 فعندما تكون قيمة هذا الـ Bit تساوى 0 فإن الـ Timer0 سوف يكون OFF وعندما يكون 1 فإن الـ Timer0سوف يكون ON.

T08BIT bit:

يستخدم هذا الـ Bit لتحديد الـ Mode الذى سوف يعمل به عداد التايمر من حيث كونه 8-bits أو 16-bits. ويجب أن تعرف أنه فى الـ 8-bit mode فإن العداد TMR0 سوف يتكون من TMR0L فقط كما هو موضح فى شكل (2-4). وعندما يكون فى الـ 16-bits mode فإن العداد TMR0 سوف يتكون من TMR0L و TMR0H كما هو موضح فى شكل (3-6).

T0CS bit:

يستخدم هذا الـ Bit لتحديد مصدر نبضات الـ Timer 0 بحيث يكون من الخارج External عن طريق الطرف RA4/T0CKI أو من الداخل Internal (من الـ Crystal oscillator). إذا تم جعل T0CS=1 فإن الـ Timer 0 سوف يعمل كـ Counter يعمل على عد النبضات القادمة من الخارج External pulses. وإذا كانت قيمة T0CS=0 فإن الـ Timer0 سوف يعمل كـ Timer يصدر نبضة على طرف الـ Overflow كل فترة زمنية يتم تحديدها طبقاً لمتطلبات المستخدم.

T0SE bit:

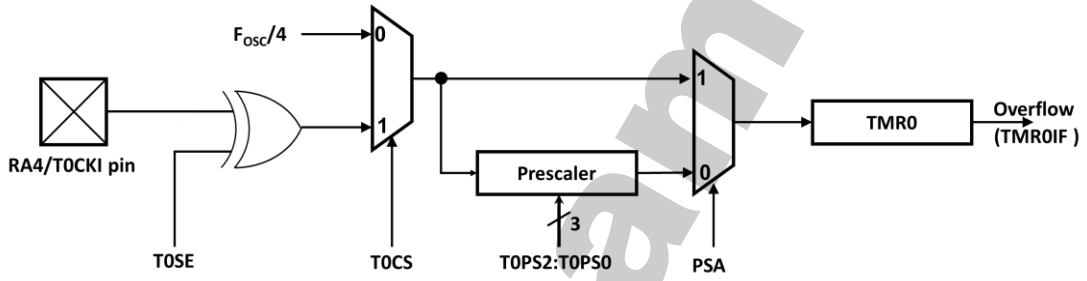
يستخدم هذا الـ Bit لتحديد نوع الـ Pulse التى سوف تعمل على زيادة الـ TMR0 من حيث كونها Rising edge أو Falling edge.

PSA bit:

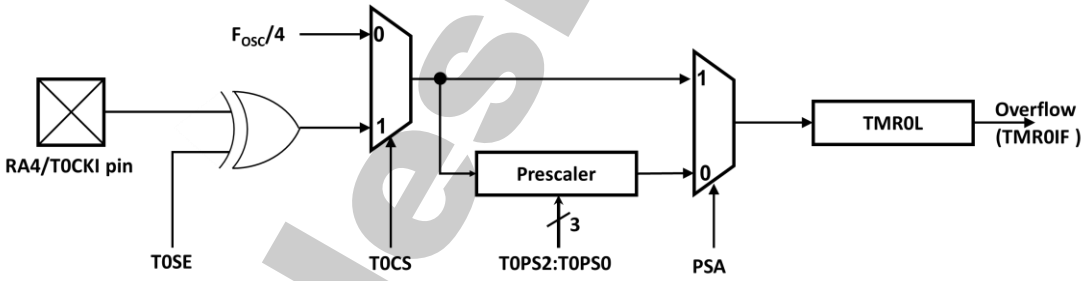
يستخدم هذا الـ Bit لتحديد هل إذا كان يريد المستخدم توظيف الـ Prescaler أو لا يريد إستخدامة.

T0PS2:T0PS0 bits:

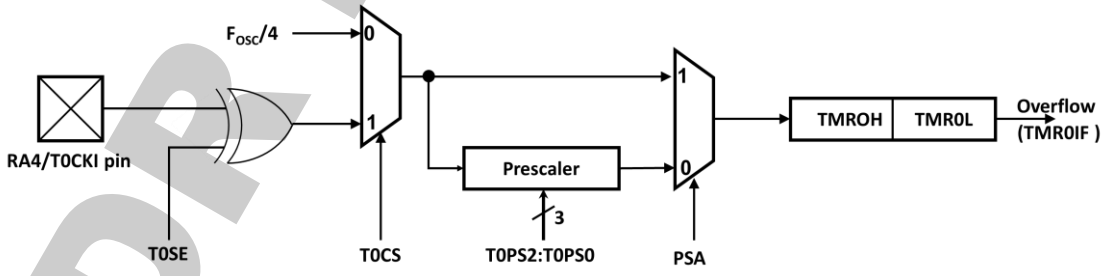
تستخدم هذه الـ Bits لتحديد قيمة الـ Prescaler.



شكل (6-1): التركيب العام الـ Timer0



شكل (6-2): تركيب الـ Timer0 فى 8-bits mode.



شكل (6-3): تركيب الـ Timer0 فى 16-bits mode.

T0CON Register

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT:** Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS:** Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE:** Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA:** Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0:** Timer0 Prescaler Select bits
111 = 1:256 prescale value
110 = 1:128 prescale value
101 = 1:64 prescale value
100 = 1:32 prescale value
011 = 1:16 prescale value
010 = 1:8 prescale value
001 = 1:4 prescale value
000 = 1:2 prescale value

6.1.2 Operating timer 0 as a counter

Example (6.1)

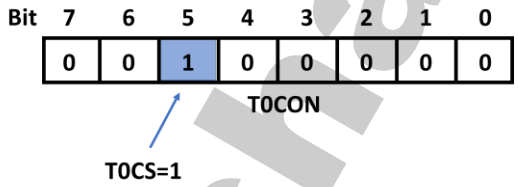
By using Timer0, design a microcontroller circuit to count the number of bottles in a production line. If the number of bottles equals 12, then turn ON a wrapping mechanism for 30 seconds by using relay connected to RD0 pin and then repeat the procedure.

Solution:

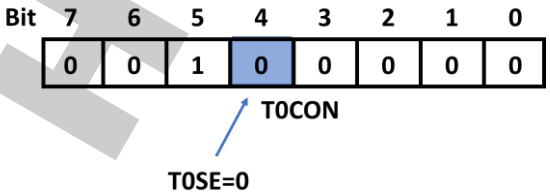
فى هذا المثال يريد أن يستخدم الـ Timer0 لكى يعمل كـ Counter. حيث أنه يريد عد الزجاجات الموجودة على خط إنتاج، فإذا كان عدد الزجاجات يساوى 12 زجاجة، فإنه سوف يتم تشغيل ماكينة التغليف لمدة 30 ثانية وذلك عن طريق ريلاى موصل بالطرف RD0.

يجب أولاً أن تتذكر أن الطرف RA4/T0CKI هو الخاص بالـ External clock pulses الخاص بالـ Timer0. ولكى يعمل الـ Timer 0 كـ Counter لعد النبضات الخارجية External pulses يجب أن يتم عمل الخطوات الآتية:

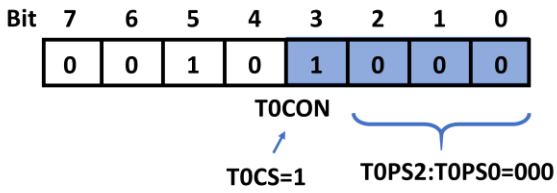
1. يتم ضبط الطرف RA4/T0CKI لكى يعمل كدخل Input عن طريق الـ TRISA register.
2. يتم ضبط مصدر النبضات Clock source لكى يكون من الخارج External عن طريق جعل T0CS=1 الموجود بداخل الـ T0CON register.



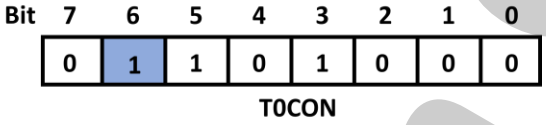
3. إذا لم يحدد نوع الـ Pulse التى سوف تعمل على زيادة عداد الـ Timer0، فإنه يتم جعل نوع النبضة Rising edge (low-to-high) عن طريق جعل T0SE=0 الموجود بداخل الـ T0CON register.



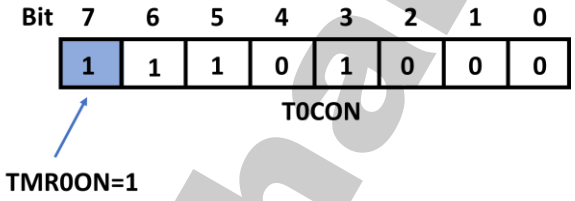
4. حيث أنه فى هذا المثال لم يحدد قيمة الـ Prescaler كما أن الشئ الذى سوف يتم عده ليس عالى التردد، فإنه سوف يتم الإستغناء عن الـ Prescaler عن طريق جعل الـ PSA=1 مع جعل قيمة الـ T0PS2:T0PS0=000.



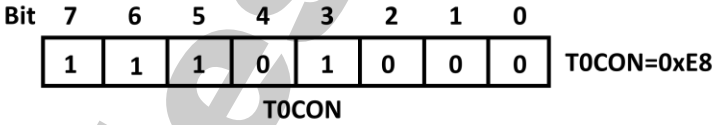
5. وحيث أن أقصى قيمة سوف نصل لها لا تتعدى الـ 255 فإننا سوف نجعل الـ Timer0 يعمل فى الـ 8-bit mode عن طريق جعل الـ T08BIT=1.



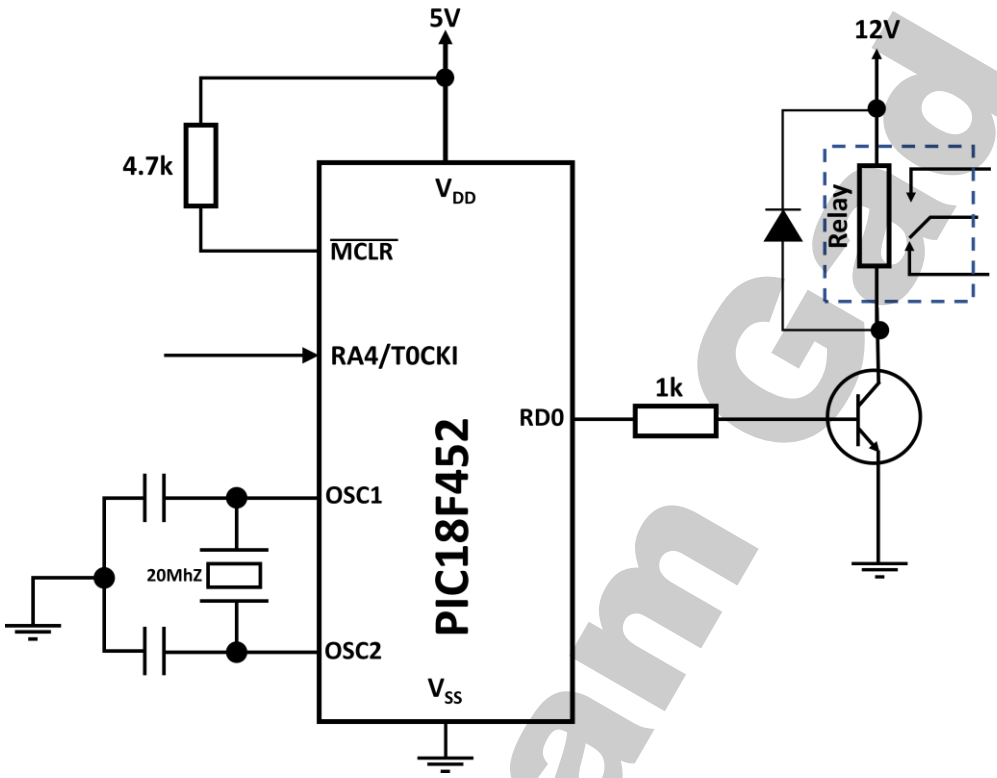
6. وأخيراً يتم تشغيل الـ Timer 0 عن طريق جعل الـ TMR0ON=1.



7. وبذلك تكون المحصلة النهائية لقيمة الـ T0CON register كالآتى:



8. يتم تصفير قيمة الـ TMR0L لبدء العد.
9. يتم قراءة قيمة الـ TMR0L بشكل مستمر إلى أن يصل قيمته إلى 12 وذلك لتشغيل ماكينة التغليف.



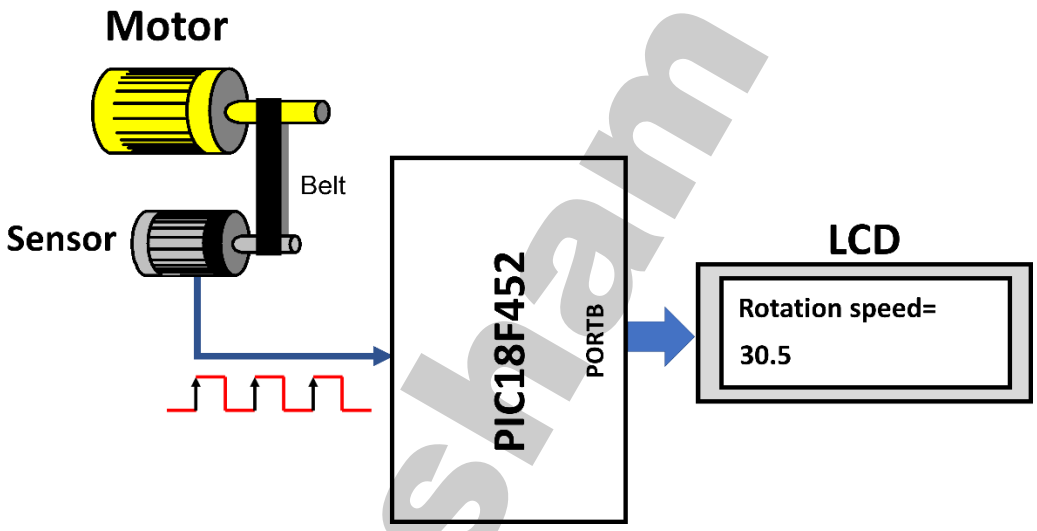
Program:

```
void main()
{
    TRISD.b0=0;
     PORTD.b0=0;
     TRISA.b4=1;
     T0CON=0xe8;

     while(1)
     {
          TMR0L=0;
          while (TMR0L<12);
          PORTD.b0=1;
          delay_ms(30000);
          PORTD.b0=0;
     }
}
```

1. تصفير العداد
2. الإنتظار حتى يصل العد إلى 12
3. إذا وصل قيمة العداد إلى 12 ، يتم تشغيل الـ Relay الموصل بالطرف RD0.
4. أنتظر لمدة 30 ثانية
5. إطفاء الـ Relay
6. تكرار كل ماسبق

Example (6.2)

By using Timer0, design a microcontroller circuit to measure the rotation speed of a motor shaft in (revolutions/sec). The motor shaft position is measured using optical encoder sensor. The resolution of the sensor is (1000 pulse/rev). The rotation speed should be displayed on the LCD screen, which is connected to PORTB. (The maximum number of pulses/second is less than 60000 pulses).

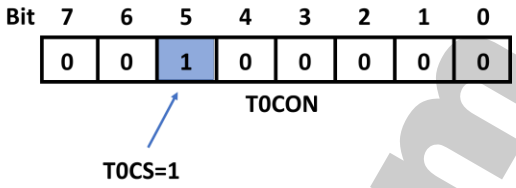


Solution:

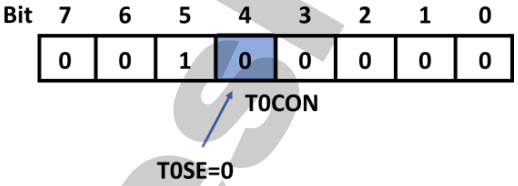
فى هذا المثال يريد أن يستخدم الـ Timer0 لكى يعمل كـ Counter، ولكن هذه المرة يريد إستخدامه لقياس سرعة دوران ماتور (عدد اللفات/الثانية) وعرض هذه القيمة على شاشة LCD. فى هذا المثال تم إستخدام Sensor يسمى الـ Optical encoder، حيث أن هذا الـ Sensor يعمل على إعطاء مجموعة من الـ Pulses أو النبضات تتناسب مع زاوية دوران الـ Shaft الخاص بالماتور وبالتحديد 1000 نبضة لكل لفة. فعلى سبيل المثال إذا كانت عدد النبضات القادمة من الحساس 2000 نبضة فى فهذا يعنى أن الـ Shaft دار دورتين. وإذا أعطى 1500 نبضة فهذا يعنى أن الـ Shaft دار دورة ونصف. ويمكن قياس سرعة الدوران عن طريق عدد النبضات القادمة من الحساس فى الثانية الواحدة ثم قسمتها على 1000 للحصول على سرعة دوران الـ Shaft.

فكرة هذا البرنامج هى نفس فكرة البرنامج السابق ولكن هذه المرة سوف نستخدم التايمر فى الـ 16- bits mode وذلك بسبب أن عدد الـ Pulses القادمة من الـ Optical encoder فى الثانية الواحدة غالباً سوف تكون أكبر من 255 pulse فى الثانية الواحدة. لذلك يتم عمل الخطوات الآتية:

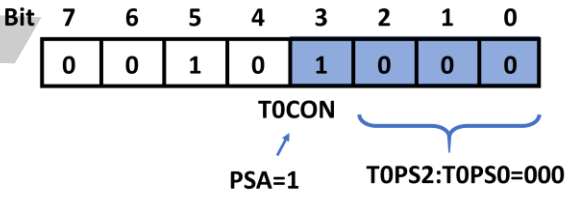
1. يتم ضبط الطرف RA4/T0CKI لكى يعمل كدخل Input عن طريق الـ TRISA register.
2. يتم ضبط مصدر النبضات Clock source لكى يكون من الخارج External عن طريق جعل T0CS=1 الموجود بداخل الـ T0CON register.



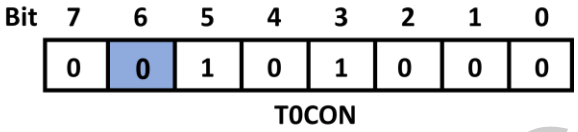
3. إذا لم يحدد نوع الـ Pulse التى سوف تعمل على زيادة الـ Timer0 فإنه يتم جعل نوع النبضة Rising edge (low-to-high) عن طريق جعل T0SE=0 الموجود بداخل الـ T0CON register.



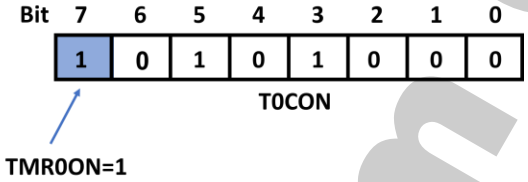
4. حيث أنه فى هذا المثال قال أن عدد النبضات لن تزيد عن 60000 فى الثانية الواحدة، فإنه سوف يتم الإستغناء عن الـ Prescaler، وذلك بسبب أن الـ Counter الخاص بالتايمر يمكن أن يصل إلى 65535. لذلك سوف نجعل الـ PSA=1 مع جعل قيمة الـ T0PS2:T0PS0=000



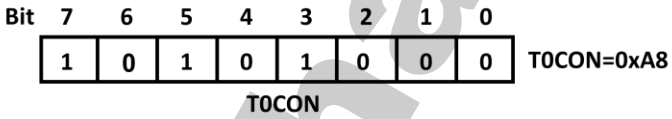
5. وحيث أن أقصى قيمة سوف نصل لها سوف تتعدى الـ 255 فإننا سوف نجعل الـ Timer0 يعمل فى الـ 16-bit mode عن طريق جعل الـ T08BIT=0.



6. وأخيراً يتم تشغيل الـ Timer0 عن طريق جعل الـ TMR0ON=1.



7. وبذلك تكون المحصلة النهائية لقيمة الـ T0CON register كالآتى:



8. يتم قراءة قيمة الـ Counter الخاص الـ Timer0 عن طريق الـ TMR0L و الـ TMR0H.

ملحوظة مهمة:

نظراً لأن الـ CPU الذى بداخل الـ PIC18F452 نوعة 8-bit، فإنه لا يستطيع التعامل مع الـ Timer0 فى 16-bits mode بشكل مباشر. أى أنه عند **تسجيل** قيمة على الـ TMR0L والـ TMR0H فإن الـقيمة التى سوف تسجل على الـ TMR0H يجب أن **تسبق** القيمة التى سوف تسجل على الـ TMR0L والعكس صحيح عن القراءة منهما.

فعلى سبيل المثال:

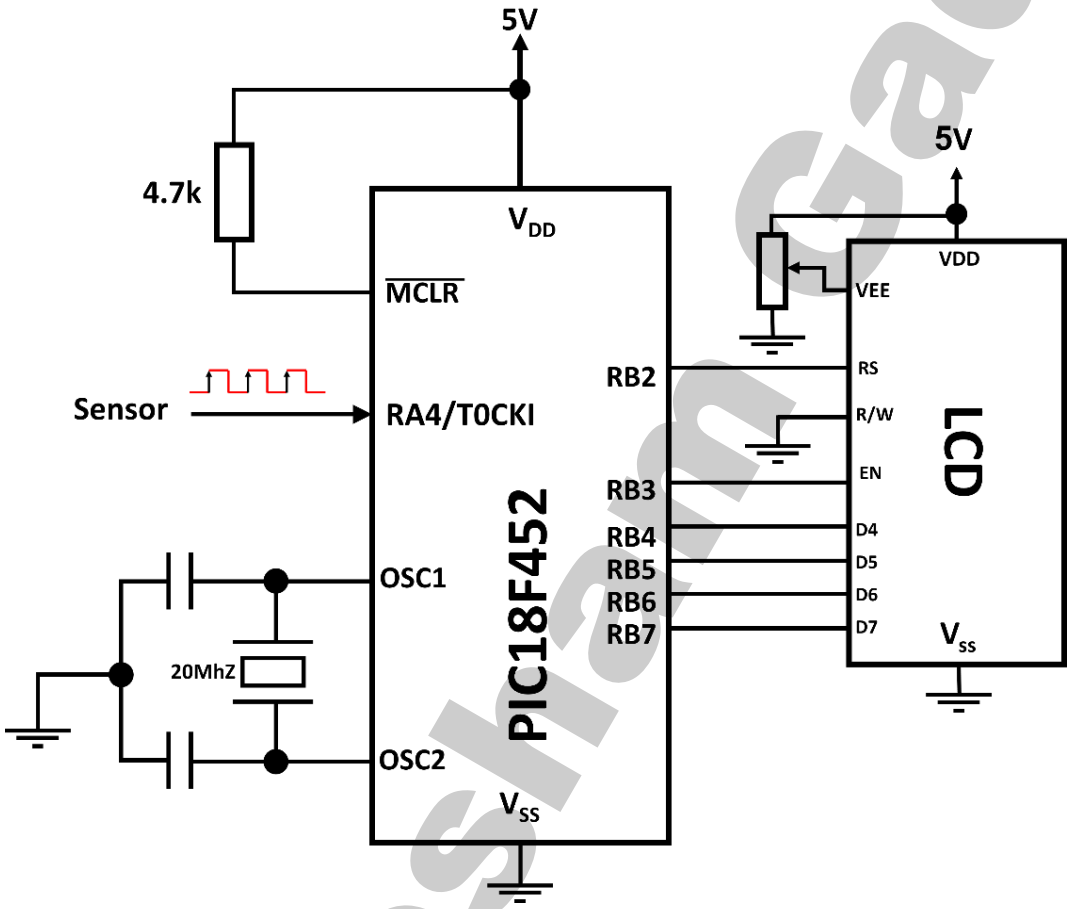
إذا أردنا تصفير قيمة الـ TMR0L والـ TMR0H فإننا نتبع الترتيب الآتى فى الكود:

```
TMR0H=0;
TMR0L=0;
```

وإذا أردنا قراءة محتويات الـ TMR0L والـ TMR0H فإننا نتبع الترتيب الآتى فى الكود:

res=TMR0L;

res=res+TMR0H*256;



Program:

```
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;

sbit LCD_RS_Direction at TRISB2_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;
```

```
void main()
{
    float  res;
    char txt[15];
   TRISD=0x00;
   PORTD=0;
   TRISA.b4=1;
   TRISB=0;

   lcd_Init();                          // إعداد الشاشة
   lcd_cmd(_lcd_clear);                 //مسح الشاشة
   lcd_cmd(_lcd_cursor_off);            //إطفاء المؤشر الذى على الشاشة
   T0CON=0xa8;                          // ضبط وتشغيل التايمر
   while (1)
   {
       TMR0H=0;
       TMR0L=0;
       delay_ms(1000);

       res=(float)TMR0L;
       res=res+(float)TMR0H*256.0;
       res=res/1000.0;

       lcd_cmd(_lcd_clear);
       lcd_out(1,1,"Rotation speed=");
       floatToStr(res, txt);
       lcd_out(2,1,txt);
   }
}
```

تصفير العداد

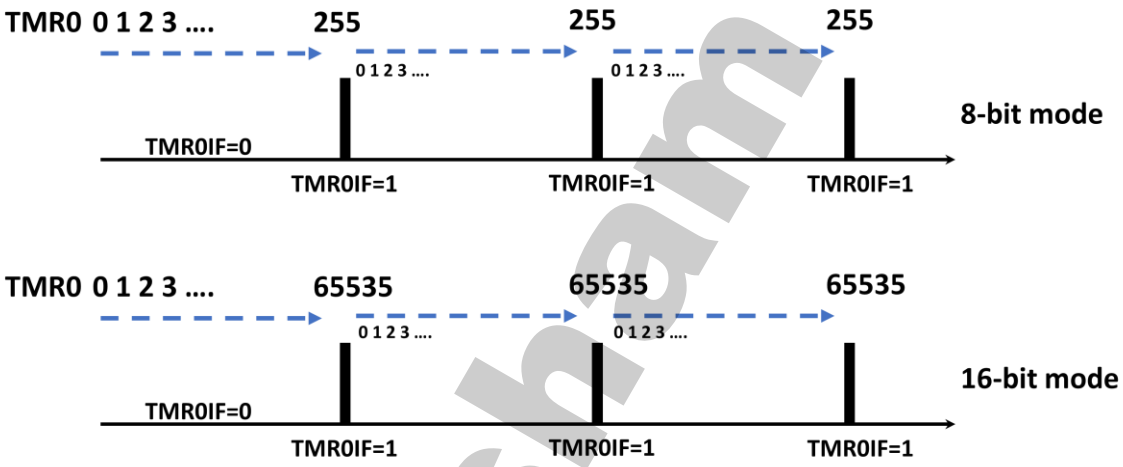
أنتظر لمدة ثانية لكى نعرف عدد النبضات القادمة من الحساس فى الثانية الواحدة

يتم تخزين قيمة العداد فى المخزن res مع قسمة الناتج على 1000 للحصول على عدد اللفات فى الثانية والواحدة

1. مسح الشاشة
2. عرض رسالة Rotation speed على السطر الأول
3. تحويل سرعة الدوران إلى characters
4. عرض سرعة الدوران على الشاشة فى السطر الثانى

6.1.2 Operating Timer0 as a timer

عندما نستخدم الـ Timer0 كمؤقت Timer، فإننا نستخدم النبضات الخاصة بالـ Crystal أى أننا نجعل مصدر النبضات داخلى Internal وذلك عن طريق جعل T0CS=0. كما أننا لا يهمنا قيمة الـ Counter هنا بقدر إهتمامنا بالـ Overflow bit الخارج من هذا الـ Counter والمسمى بإسم TMR0IF. وهى عبارة عن bit تصبح قيمتها تساوى 1 عندما يصل الـ Counter الخاص بالـ Timer 0 إلى أقصى قيمة له (أى 255 عند 8-bit mode) و (65535 عند 16-bit mode) كما هو موضح فى شكل (4-6).



شكل (4-6): طريقة عمل الـ Timer0 فى الـ 8-bit والـ 16-bit.

والـ Overflow bit الخاص بالـ Timer0 موجود بداخل الـ INTCON register ويقع تحت أسم TMR0IF. ويلاحظ أنه ينتهى بحرفين وهم IF أو Interrupt flag ولكن لماذا؟

ذلك لأن هذا الـ bit يُمكن أن يُستخدم لعمل Interrupt للـ CPU الخاص بالـ Microcontroller كل فترة زمنية وذلك لتنفيذ برنامج معين وليكن كل نصف ثانية أو كل ثانية بشكل آلى دون الإستعانة بأمر الـ delay. والفترة بين كل Overflow والأخر تسمى بإسم Overflow time. ولكن يتبقى السؤال هنا، كيف يمكن ظبط الـ Overflow time الخاص بالـ Timer0؟

المسئول عن التحكم فى سرعة الوصول إلى أعلى قيمة للـ Counter أو بالمعنى الأصح التحكم فى قيمة الـ Overflow time ثلاثة أشياء أساسية وهم:

1. تردد الـ Crystal oscillator الواصل بالـ Microcontroller.

2. قيمة الـ Prescaler.

3. القيمة الإبتدائية Initial value للـ Counter الخاص الـ Timer0.

تردد الكريستالة Crystal oscillator هو المسئول الأول عن التحكم سرعة الـ Counter، فكلما كان تردد الكريستالة أعلى كلما كانت سرعة العداد أعلى وبذلك يقل الـ Overflow time. وأيضاً الـ Prescaler له دور كبير فى سرعة العداد فكلما كانت قيمته أكبر كلما كان العداد أبطأ فى العد أى أن الـ Overflow time سوف يصبح أكبر. وأخير القيمة الأبتدائية لها دور فى قيمة الـ Overflow time، حيث أنه تم بدء العد من قيمة 125 بدلاً من الصفر ، فإن زمن الـ Overflow سوف يقل للنصف.

ويتم الربط بين الثلاثة معاملات السابقة عن طريق المعادلة الآتية:

$$\text{Overflow time (sec)}\big|_{8-bit} = 4 \times T_{osc} \times \text{Prescaler} \times (256 - \text{initial value of TMR0}) \qquad (6.1)$$

Or

$$\text{Overflow time (sec)}\big|_{16-bit} = 4 \times T_{osc} \times \text{Prescaler} \times (65536 - \text{initial value of TMR0}) \qquad (6.2)$$

حيث أن الـ T_{osc} هو الـ Period الخاص بالـ Crystal oscillator الواصل بالـ Microcontroller ويساوى:

$$T_{osc} = \frac{1}{F_{osc}}$$

فعلى سبيل المثال، إذا كان تردد الـ Crystal oscillator الواصل بالـ Microcontroller يساوى 4Mhz. وكانت قيمة الـ Prescaler تساوى 1:4 ، والقيمة الإبتدائية للـ Timer0 تساوى 100 وكان الـ Timer0 فى وضع الـ 8-bit mode أحسب زمن الـ Overflow ؟

أولاً : يتم حساب الـ T_{osc} كالآتى:

$$T_{osc} = \frac{1}{F_{osc}} = \frac{1}{4*10^{6}} = 0.25*10^{-6}\ \text{sec}$$

ثانياً: يتم التعويض فى المعادلة (6.1) للحصول على الـ Overflow time:

$$\text{Overflow time (sec)}\big|_{8-bit} = 4\times 0.25\times 10^{-6}\times 4\times(256-100) = 624\times 10^{-6}\ \text{sec}$$

ولكن فى المعتاد تكون قيمة الـ Initial value الخاصة بالـ Timer0 هى المجهولة. حيث أنه يكون معلوم قيمة الـ Overflow time، كما أنه يتم فرض قيمة الـ Prescaler كما أن قيمة تردد الـ Crystal oscillator تكون معلومة. لذلك يتم حساب الـ Initial value الخاص بالـ Timer 0 كالآتى:

$$\text{Initial value of TMR0}\big|_{8-bit} = 256 - \frac{\text{Overflow time (sec)}}{4\times T_{osc}\times \text{Prescaler}} \qquad (6.3)$$

or

$$\text{Initial value of TMR0}\big|_{16-bit} = 65536 - \frac{\text{Overflow time (sec)}}{4\times T_{osc}\times \text{Prescaler}} \qquad (6.4)$$

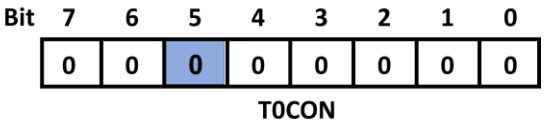
Example (4.3)

By using Timer0, design a microcontroller circuit and write a program to flash a LED with 0.01 second and 0.01 second OFF, which is connected to RD0 pin. The Timer 0 is adjusted in 8-bit mode with a prescaler value of 1:128. The crystal oscillator frequency is 4 Mhz.

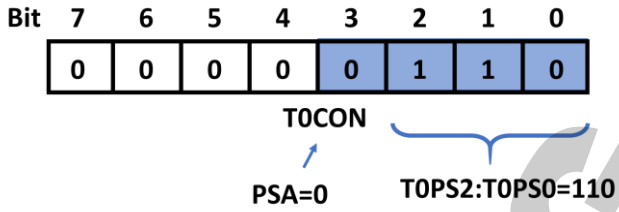
Solution:

فى هذا المثال مطلوب عمل دائرة Flasher لتشغيل وإطفاء LED بمعدل 0.01 ثانية وذلك بدون إستخدام أمر delay_ms. والمطلوب هنا إستخدام الـ Timer0 لعمل هذه المهمة. لذلك يتم إتباع الخطوات التالية:

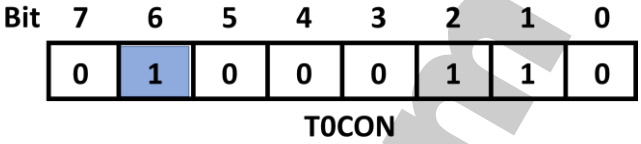
1. يتم ضبط مصدر النبضات Clock source لكى يكون من الداخل Internal عن طريق جعل T0CS=0 الموجود بداخل الـ T0CON register.



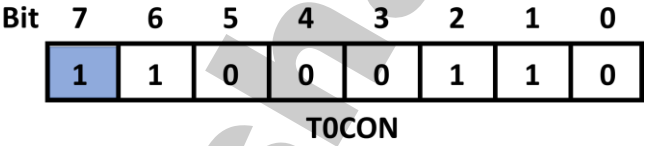
2. حيث أننا نريد إستخدام الـ Prescaler، فإنه يجب جعل الـ PSA=0 مع جعل قيمة الـ T0PS2:T0PS0=110 لضبط قيمة الـ Prescaler لكى تكون 1:128.



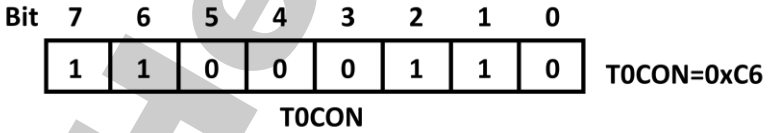
3. يتم ضبط الـ Timer0 لكى يعمل فى الـ 8-bit mode عن طريق جعل الـ T08BIT=1.



4. يتم تشغيل الـ Timer 0 عن طريق جعل الـ TMR0ON=1.



5. وبذلك تكون المحصلة النهائية لقيمة الـ T0CON register كالآتى:

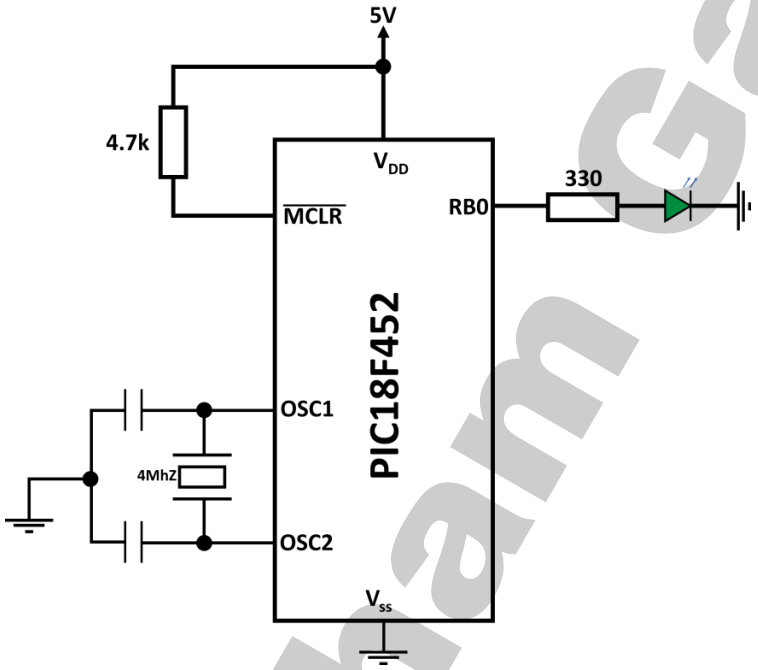


6. يتم حساب القيمة الإبتدائية الخاصة الـ Timer0 وتحميلها على TMR0L كالآتى:

$$\text{Initial value of TMR0}\big|_{8-bit} = 256 - \frac{\text{Overflow time (sec)}}{4 \times T_{osc} \times \text{Prescaler}} =$$

$$= 256 - \frac{0.01}{4 \times 0.25 \times 10^{-6} \times 128} = 177.87 \approx 178 = 0xB2$$

7. يتم تصفير الـ TMR0IF ثم يتم الإنتظار حتى تنتهى المدة المطلوبة وذلك عن طريق التحقق بشكل دورى على قيمة الـ TMR0IF، فإذا أصبحت تساوى 1 فهذا يعنى إنتهاء المدة المطلوبة.



Program:

```
void main()
{
    TRISD.b0=0;                 //ضبط طرف الخرج
    PORTD.b0=0;                 //تصفير طرف الخرج
    T0CON=0xc6;                 //ضبط التايمر
    while(1)
    {
        TMR0L=0xB2;
        INTCON.TMR0IF=0;
        while(INTCON.TMR0IF==0);

        PORTD.b0=~PORTD.bo;                 // إعكس حالة الليد
    }
}
```

1. تحميل القيمة الإبتدائية بالعداد
2. تصفير الـ TMR0IF
3. الإنتظار حتى ينتهى الـ Timer0

Example (6.4)

Without using delay_ms command (i.e. by using Timer0), design a microcontroller circuit and write a program to flash a LED with 1 second and 1 second OFF, which is connected to RD0 pin. (Crystal oscillator 4Mhz)

Solution:

فى هذا المثال مطلوب عمل دائرة Flasher لتشغيل وإطفاء LED بمعدل 1 ثانية وذلك بدون إستخدام أمر delay_ms. ولكن هذه المرة لم يحدد قيمة الـ Prescaler ولا الـ Mode الخاص بالتايمر. فكيف يمكن حساب هذه الأشياء؟

يتم أولاً فرض فيمة الـ Prescaler ولتكن 1:64، كما يتم فرض أن الـ Mode سوف يكون 8-bit. المطلوب هنا إيجاد الـ Initial value للـ Timer0.

$$\text{Initial value of TMR0}\big|_{8-bit} = 256 - \frac{\text{Overflow time (sec)}}{4 \times T_{osc} \times \text{Prescaler}} =$$

$$= 256 - \frac{1}{4 \times 0.25 \times 10^{-6} \times 64} = -15369$$

يُلاحظ هنا أن الـ Initial value للـ Timer0 سالبة وهذا غير مقبول. كما يُلاحظ أيضاً أنها أكبر بكثير من الـ 256. وحتى إذا تم تجربة أعلى قيمة للـ Prescaler ولتكن 1:256، فإن الـ Initial value للـ Timer0 سوف تكون:

$$\text{Initial value of TMR0}\big|_{8-bit} = 256 - \frac{\text{Overflow time (sec)}}{4 \times T_{osc} \times \text{Prescaler}} =$$

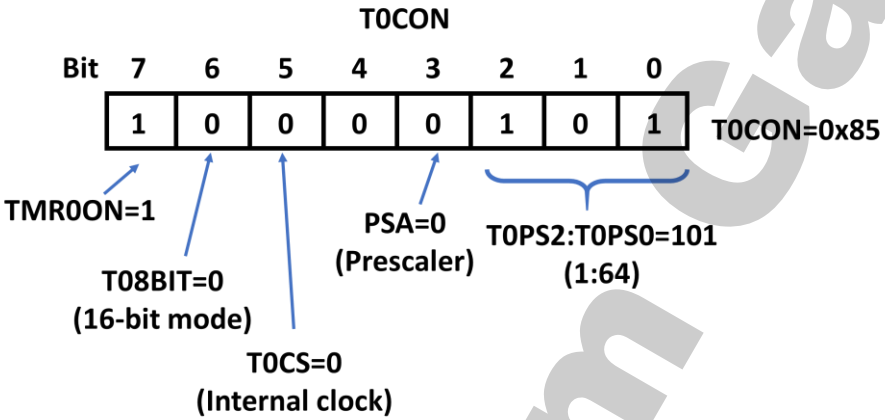
$$= 256 - \frac{1}{4 \times 0.25 \times 10^{-6} \times 256} = -3650.25 \approx -3650$$

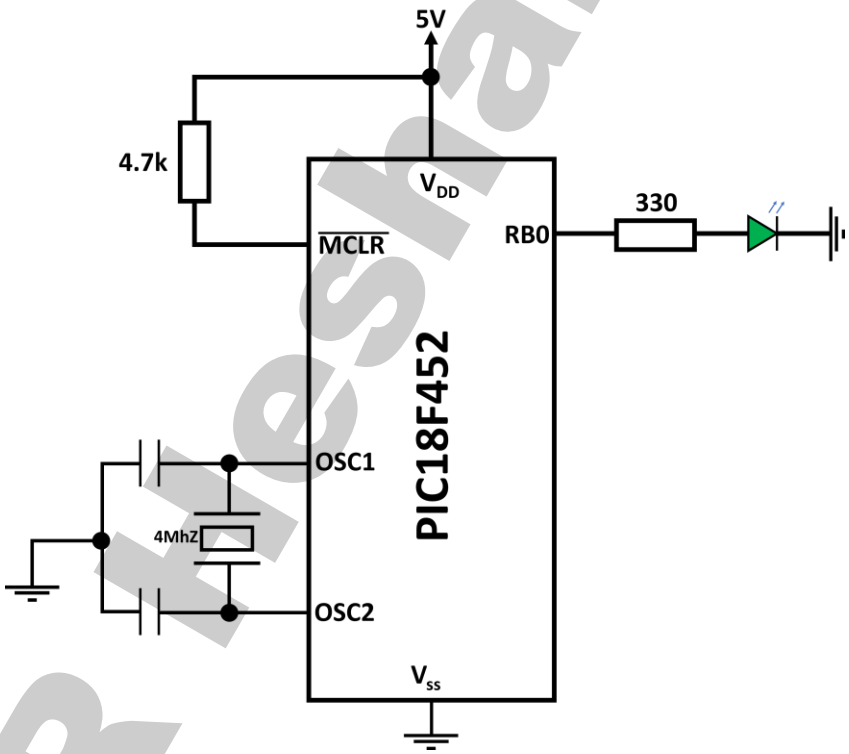
وهى قيمة مازالت سالبة وأكبر بكثير من الـ 255 ، لذلك سوف نجرب الـ 16-bit mode مع إستعمال 1:64 كقيمة للـ Prescaler.

$$\text{Initial value of TMR0}\big|_{16-bit} = 65536 - \frac{\text{Overflow time (sec)}}{4 \times T_{osc} \times \text{Prescaler}} =$$

$$= 65536 - \frac{1}{4 \times 0.25 \times 10^{-6} \times 64} = 49911 = 0xC2F7$$

نُلاحظ أن القيمة التى حصلنا عليها موجبة كما أنها أقل من 65535 وهذا يعنى أنا الـ Timer0 يجب أن يكون فى الـ 16-bit mode. وبذلك سوف تكون قيمة الـ TMR0L=0xF7 والـ TMR0H=0xC2. كما أن قيمة الـ T0CON register سوف تكون كالآتى:





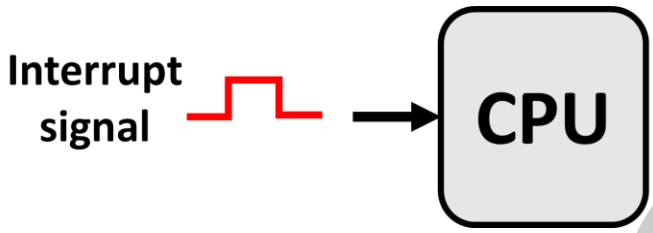
Program:

```
void main()
{
TRISD.b0=0;
PORTD.b0=0;

T0CON=0x85;                    //ضبط التايمر

while (1)
    {
    TMR0H=0xC2;
    TMR0L=0xF7;
    INTCON.TMR0IF=0;
    while (INTCON.TMR0IF==0);

    PORTD.b0=~ PORTD.b0;
    }
}
```

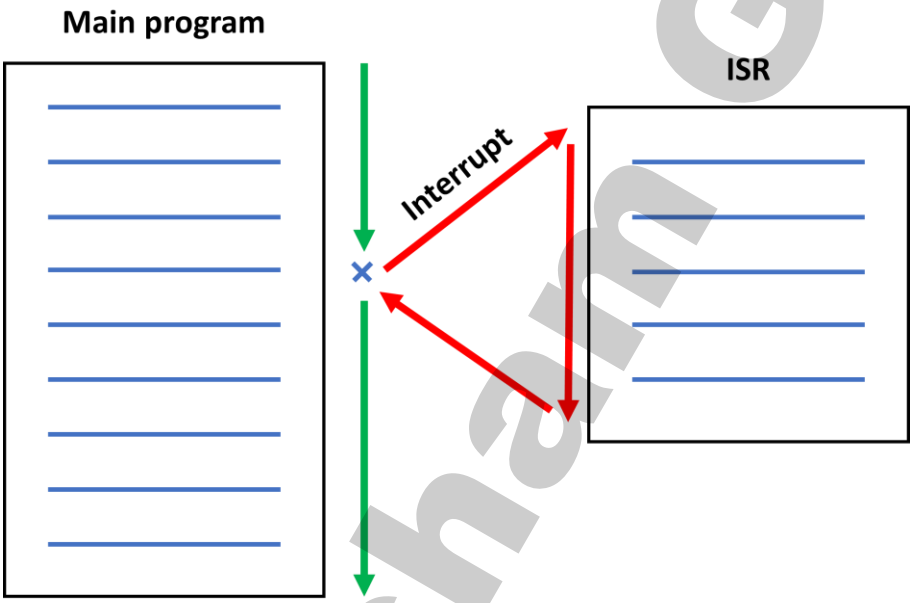
1. تحميل القيمة الإبتدائية بالعداد الخاص بالـ Timer0
2. تصفير الـ TMR0IF
3. الإنتظار حتى ينتهى الـ Timer0

6.1.2 Operating Timer0 as a timer with interrupt

قبل أن نبدأ فى طريقة إستخدام الـ Timer 0 مع الـ interrupt يجب أولاً نعرف ما هو الـ Interrupt . الـ Interrupt هو حدث مفاجئ يجبر الـ CPU بترك تنفذ البرنامج الرئيسى والذهاب إلى برنامج فرعى يدعى بإسم الـ Interrupt service routine (ISR) لكى يتم تنفيذ هذا البرنامج ثم العودة مرة أخرى إلى البرنامج الرئيسى كما هو موضح فى شكل (5-6). ومن مزاياه أنه لا يجعل الـ CPU يعمل على مراقبة شئ ما بشكل مستمر وخاصة فى مراقبة الأشياء الخطرة (مثل أنظمة إنذار الحريق).



شكل (6-5): إشارة الـ Interrupt.



شكل (6-6): ترك البرنامج الرئيسى وتنفيذ برنامج الـ ISR عند حدوث الـ Interrupt.

ولكن ما فائدة ربط الـ Interrupt بالـ Timer0؟

عرفنا فيما سبق أنه لعمل دائرة Flasher ، فإنه يجب إستخدام امر الـ delay_ms أو ضبط الـ Timer0 على الوقت المطلوب ثم مراقبة الـ TMR0IF للتحقق من إنتهاء الوقت بشكل مستمر. ولكن المشكلة هنا أن كلا الطريقتين تعملان على إستهلاك وقت الـ CPU الثمين الذى يمكن إستخدامه لتنفيذ برامج أخرى أكثر أهمية. فمثلاً، بدلاً من أن يعمل الـ CPU على تنفيذ برنامج أخر مثل التحكم فى درجة حرارة التكييف الكهربائى Air conditioner أو التحكم بفرامل السيارة Brake system ، فإنه يعمل على تضيع هذا الوقت فى تنفيذ أمر الـ delay_ms أو مراقبة الـ TMR0IF.

وهنا تأتى فائدة الـ Interrupt، حيث أنه يمكن يتم توصيل الـ TMR0IF بطرف الـ Interrupt الخاص بالـ CPU، وبذلك بمكن أن نجعل الـ CPU يعمل على تنفيذ أى برنامج أخر أكثر أهمية من

الإنتظار حتى ينتهى الوقت وإذا أصبحت قيمة الـ TMR0IF=1 ، فإنه سوف يحدث Interrupt للـ CPU وسوف يترك البرنامج الرئيسى Main program لكى يذهب إلى ISR والذى سوف يحتوى على الكود الذى نريد تنفيذه عند إنتهاء المدة المطلوبة.

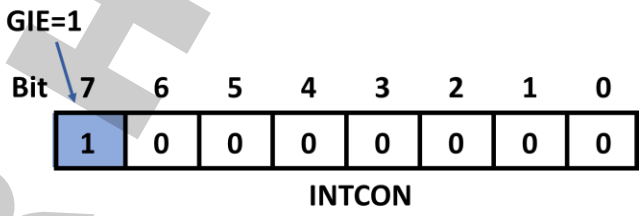
خطوات ضبط الـ Timer0 مع الـ Interrupt الخاص بالـ PIC18F452:

1. يتم كتابة الكود الذى نريد تنفيذه عند حدوث الـ Interrupt الخاص بالـ Timer0 فى برنامج الـ mikroC كالآتى:

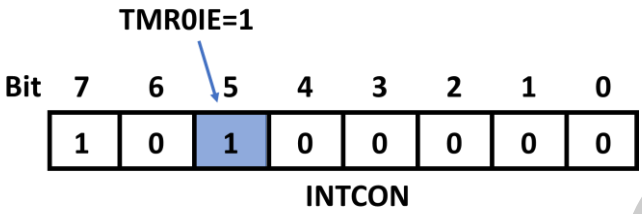
```
void interrupt ()
{
   ……..
   ……..
   ……..
}
void main()
{
   ……..
   ……..
}
```

يتم كتابة الكود المطلوب تنفيذه هنا عند حدوث Interrupt

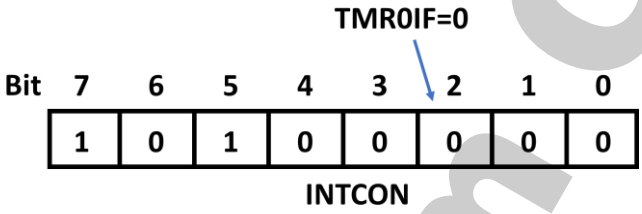
2. يتم ضبط وتشغيل الـ Timer 0 للحصول على الوقت المطلوب كما تم ذكر سابقاً فى المثال (6-4) و (6-5).
3. يتم تشغيل آلية الـ Interrupt وذلك عن طريق جعل الـ GIE bit=1 حيث أن هذا الـ Bit موجود فى الـ INTCON register.



4. يتم تشغيل آلية الـ Interrupt الخاصة بالـ Timer 0 وذلك عن طريق جعل الـ TMR0IE=1 لكى يتم السماح بوصول إشارة الـ TMR0IF إلى الـ CPU.



5. يتم تصفير الـ TMR0IF.



INTCON Register

GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
bit 7							bit 0

bit 7 **GIE/GIEH:** Global Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked interrupts
0 = Disables all interrupts

When IPEN = 1:
1 = Enables all high priority interrupts
0 = Disables all interrupts

bit 6 **PEIE/GIEL:** Peripheral Interrupt Enable bit

When IPEN = 0:
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

When IPEN = 1:
1 = Enables all low priority peripheral interrupts
0 = Disables all low priority peripheral interrupts

bit 5 **TMR0IE:** TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 overflow interrupt
0 = Disables the TMR0 overflow interrupt

bit 4 **INT0IE:** INT0 External Interrupt Enable bit
1 = Enables the INT0 external interrupt
0 = Disables the INT0 external interrupt

bit 3 **RBIE:** RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF:** TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INT0IF:** INT0 External Interrupt Flag bit
1 = The INT0 external interrupt occurred (must be cleared in software)
0 = The INT0 external interrupt did not occur

bit 0 **RBIF:** RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

ملحوظة مهمة:

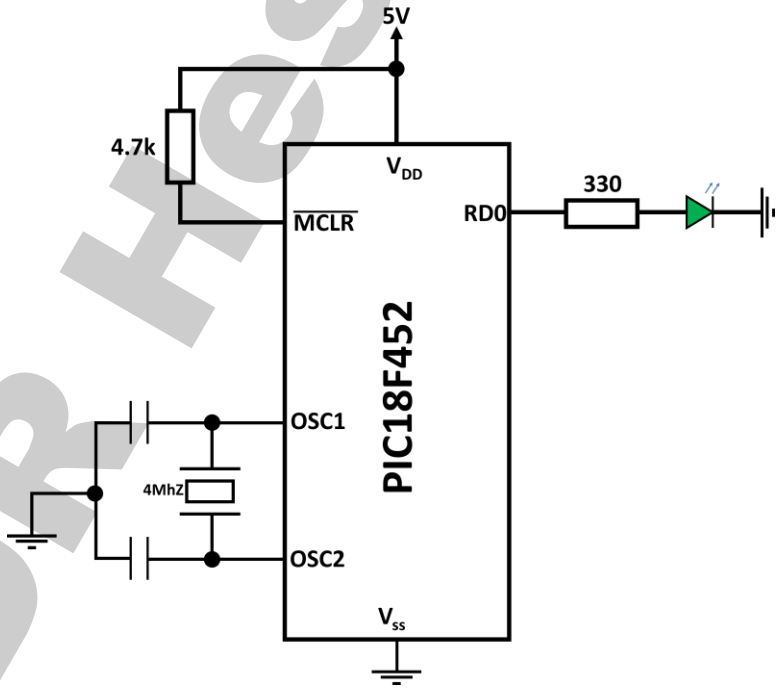
عندما يذهب الـ CPU لتنفيذ كود الـ Interrupt، فإنه يجب تحميل الـ Initial value الخاصة بالـ Timer 0 مع تصفير الـ TMR0IF فى نهاية كود الـ Interrupt. ولذلك لكى يتم برمجة الـ Timer 0 لكى يبدأ من Initial value التى فى كل مرة يحدث فيها الـ Interrupt.

Example (6-6)

By using Timer 0 interrupt, design a microcontroller circuit and write a program to flash a LED with 1 second ON and 1 second OFF, which is connected to RD0 pin. The microcontroller is connected to 4 MHz crystal oscillator.

Solution:

فى هذا المثال مطلوب عمل دائرة Flasher لتشغيل وإطفاء LED كل 1 ثانية، ولكن هذه المرة بإستخدام الـ Interrupt وذلك بدلاً من جعل الـ CPU ينتظر الـ TMR0IF فى كل مرة. وحيث أن القيمة الإبتدائية للـ TMR0 تم حسابها سابقاً فى المثال (6-5)، فإننا سوف نكتب البرنامج مباشرة كالتالى.



Program:

```
void interrupt()
{
    portd.b0=~portd.b0;

    TMR0H=0xC2;
    TMR0L=0xF7;
    INTCON.TMR0IF=0;
}

void main()
{
    TRISD.b0=0;
    PORTD.b0=0;

    T0CON=0x85;
    TMR0H=0xC2;
    TMR0L=0xF7;

    INTCON.GIE=1;
    INTCON.TMR0IE=1;
    INTCON.TMR0IF=0;

    while(1)
    {
            //لا يتم كتابة أى شئ هنا
    }

}
```

هذا الكود سوف ينفذ بشكل آلى كل ثانية

1. يتم تنفيذ المطلوب وهو عكس إشارة الـ LED
2. تحميل القيمة الإبتدائية للعداد
3. تصفير Overflow

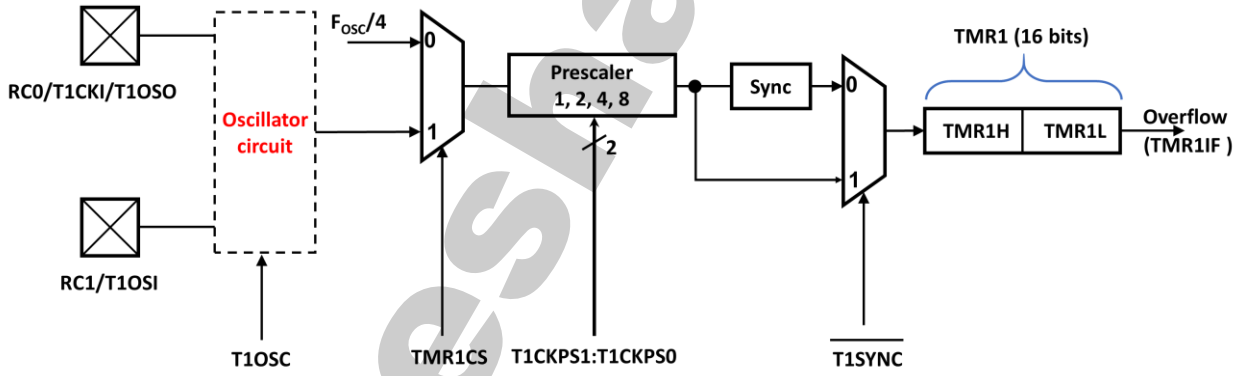
1. ضبط التايمر
2. تحميل القيمة الإبتدائية للعداد

1. تشغيل Interrupt
2. تشغيل الـ Interrupt الخاص بالـ Timer0

6.2 Timer1

على عكس الـ Timer0، يعمل الـ Timer1 بنظام الـ 16-bits mode فقط. حيث أن الـ Counter الخاص بالـ Timer1 يعد من 0 إلى 65535. ويتم ضبط خصائص هذا التايمر عن طريق الـ T1CON register. شكل (6-7) يوضح تركيب الـ Timer1 وهو مشابه جداً لتركيب الـ Timer0. فى هذا الشكل يُلاحظ أنه يوجد Register يسمى الـ TMR1 وهو عبارة عن الـ Counter الخاص بالـ Timer0. كما يوجد الـ Prescaler لكى يتم فى التحكم فى سرعة عد الـ TMR1.

كما يُلاحظ بالشكل أنه يوجد طرفان خارجيان External pins للـ Timer1 وهم T1OSO و T1OSI كما توجد دائر تسمى Oscillator circuit. وذلك لأن الـ Timer1 به إمكانية غير موجودة بالـTimer0 وهى توصيل Crystal oscillator كمصدر للنبضات External pulses للعداد الخاص بالـ Timer1 كما هو موضح بشكل (6-7). وتستخدم هذه الإمكانية لعمل توقيتات دقيقة بهذا الـ Timer عن طريق إستخدام كريستالة خارجية إضافية.



شكل (6-7): تركيب الـ Timer1

وعند الوصول إلى نهاية العد (أى عند الوصول إلى 65535)، فإن TMR1IF bit (الخارج من TMR1) سوف يساوى 1. ويوجد الـ TMR1IF فى الـ PIR1 register.

PIR1 register

PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

شرح الـ T1CON Register

RD16 bit:

يستخدم هذا الـ bit لتحديد طريقة قراءة القيمة الموجودة على الـ TMR1 register. فإذا كانت قيمة الـ RD16=1، فإنه يتم قراءة أو تعديل قيم الـ TMR1L والـ TMR1H بنفس الترتيب الذى تم فى الـ Timer0 (ترتيب القراءة أو تعديل قيمة الـ TMR1L والـ TMR1H). وإذا كانت قيمة الـ RD16=0، فإن الترتيب غير عن القراءة أو الكتابة على الـ TMR1L والـ TMR1H. **ويفضل** جعل قيمة الـ RD16=1، وذلك لأن الطريقة الثانية تؤدى إلى إستخراج قيم غير صحيحة عند عند قراءة قيمة الـ TMR1.

T1CKPS2:T1CKPS0

تستخدم هذه الـ Bits لتحديد قيمة الـ Presaler مع العلم أنه أعلى قيمة للـ Prescaler فى الـ Timer1 هى 1:8

T1OSCEN bit:

تستخدم لتحديد إمكانية توصيل كريستالة خارجية على أطراف الـ Timer1 من عدمة.

T1SYNC bit:

يستخدم هذا الـ Bit لعمل تزامن Synchronization بين النبضات الخارجية External pulses والنبضات الداخلية للـ Microcontroller. أى أن هذا الـ Bit يستخدم فقط عندما نريد إستخدام الـ Timer1 كعداد Counter لعد النبضات الخارجية أى عندما تكون الـ TMR1CS=1. ويجب أن تعرف أن هذا الـ bit معكوس Inverted أى أنه عندما يكون T1SYNC=0 فإن هذه الإمكانية سوف تكون فعالة Active .

TMR1CS bit:

يستخدم لتحديد مصدر النبضات Clock source من حيث كونها External pulses أو Internal pulses. فإذا كانت الـ TMR1CS=1 فهذا يعنى أننا سوف نستخدم النبضات القادمة من خارج الـ Microcontroller وإذا كانت الـ TMR1CS=0 فهذا يعنى أننا سوف نستخدم الداخلية Internal pulses والتى هى $F_{osc}/4$.

TMR0ON bit:

يستخدم هذا الـ Bit لتشغيل وإطفاء الـ Timer1 فعندما تكون قيمة هذا الـ Bit تساوى 0 فإن الـ Timer1 سوف يكون OFF وعندما يكون 1 فإن هذا التايمر سوف يكون ON.

T1CON Register

RD16	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7 **RD16:** 16-bit Read/Write Mode Enable bit
1 = Enables register Read/Write of Timer1 in one 16-bit operation
0 = Enables register Read/Write of Timer1 in two 8-bit operations

bit 6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Enable bit
1 = Timer1 Oscillator is enabled
0 = Timer1 Oscillator is shut-off
The oscillator inverter and feedback resistor are turned off to eliminate power drain.

bit 2 **$\overline{\text{T1SYNC}}$:** Timer1 External Clock Input Synchronization Select bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T13CKI (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

6.2.2 Operating Timer1 as a Counter

Example (6.7)

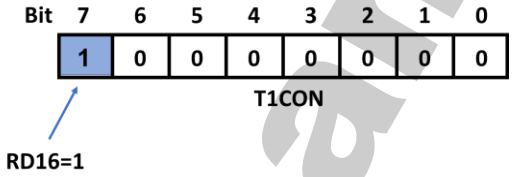
By using Timer1, design a microcontroller circuit to count the number of bottles in a production line. If the number of bottles equals 20, then turn ON a wrapping mechanism for 30 seconds by using a relay connected to RD0 pin and repeat the procedure.

Solution:

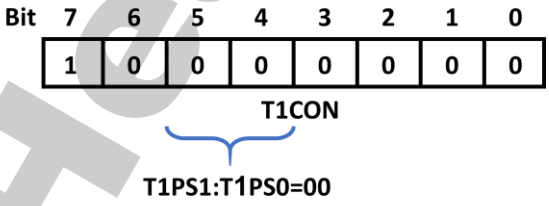
فى هذا المثال يريد أن يستخدم الـ Timer1 لكى يعمل كـ Counter. حيث أنه يريد عد الزجاجات الموجودة على خط الأنتاج، فإذا كان عدد الزجاجات يساوى 20 زجاجة، فإنه سوف يتم تشغيل ماكينة التغليف لمدة 30 ثانية وذلك عن طريق ريلاى موصل بالطرف RD0.

يجب أولاً أن تتذكر أن الطرف RC0/T1CKI هو الخاص بالـ External clock pulses الخاص بالـ Timer1. ولكى يعمل الـ Timer 1 كـ Counter لعد النبضات الخارجية External pulses يجب أن يتم عمل الخطوات الآتية:

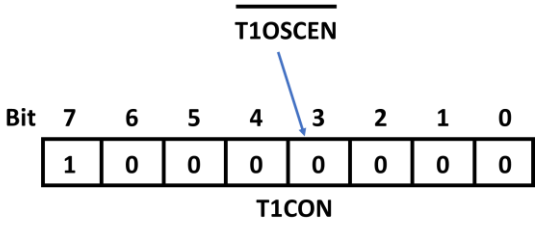
1. يتم ضبط الطرف RC0/T1CKI لكى يعمل كدخل Input عن طريق الـ TRISC register.
2. يتم جعل قيمة الـ RD16=1، وذلك لذيادة دقة القراءة من الـ TMR1.



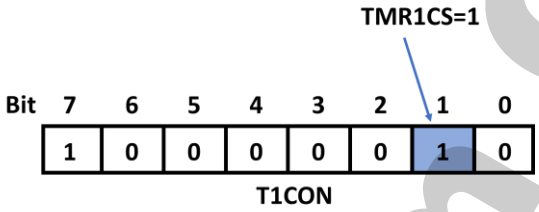
3. حيث أنه فى هذا المثال لم يحدد قيمة الـ Prescaler، كما أن الشئ الذى سوف يتم عده ليس عالى التردد، فإنه سوف يتم جعل قيمة الـ Prescaler=1:1 وذلك عن طريق T1PS1:T1PS0=00.



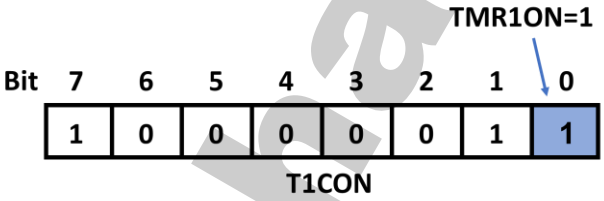
4. وحيث أننا لن نستخدم كريستالة خارجية موصلة بأطراف الـ Timer1 فإننا سوف نجعل الـ T1OSCEN=0.
5. كما يتم جعل الـ T1SYNC=0 لتشغيل التزامن Synchronization.



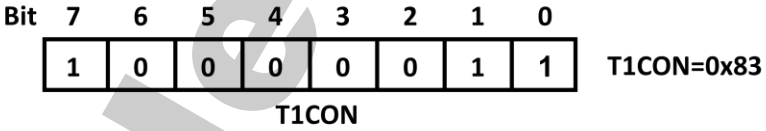
6. يتم جعل مصدر النبضات External وذلك عن طريق جعل TMR1CS=1.



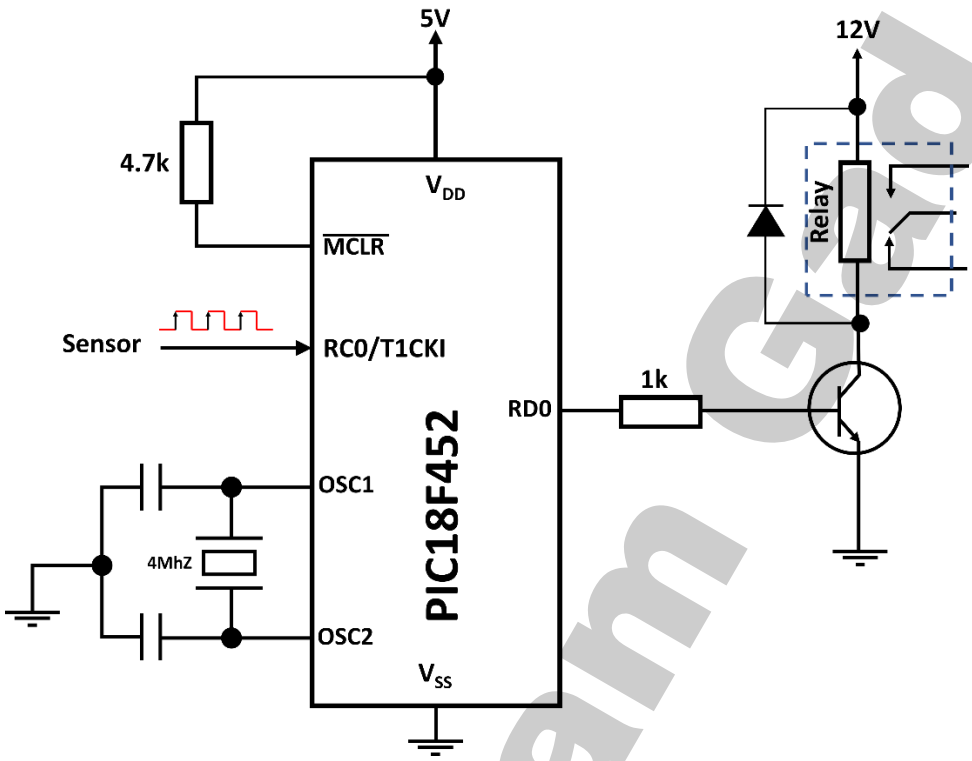
7. وأخيراً يتم تشغيل الـ Timer1 عن طريق جعل الـ TMR1ON=1.



8. وبذلك تكون المحصلة النهائية لقيمة الـ T1CON register كالآتى:



9. يتم قراءة قيمة الـ Counter الخاص الـ Timer1 عن طريق الـ TMR1 والذى يتكون من TMR1L و TMR1H. وحيث أن العدد الذى نريد التحقق منه لن يزيد عن 255 فإننا سوف نتحقق فقط من قيمة الـ TMR1L بشكل مستمر إلى أن تصل قيمته إلى 20 وذلك لتشغيل ماكينة التغليف.

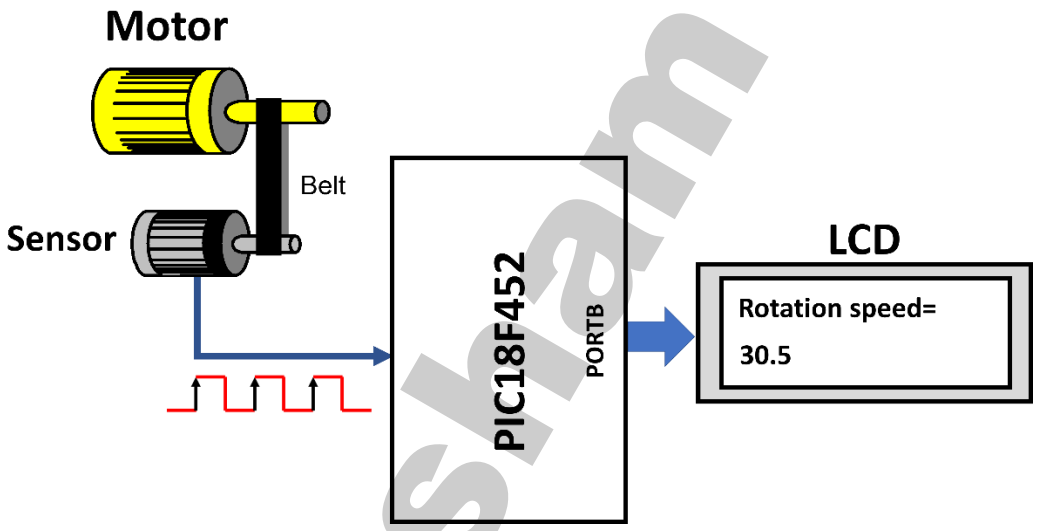


Program:

```
void main()
{
    TRISD.b0=0;
     PORTD.b0=0;
     TRISC.b0=1;
     T1CON=0x83;

     while(1)
     {
          TMR1L=0x00;
          while(TMR1L<12);
          PORTD.b0=1;
          delay_ms(30000);
          PORTD.b0=0;
     }
}
```

Example (6.8)

By using Timer1, design a microcontroller circuit to measure the rotation speed of a motor shaft in (revolutions/sec). The motor shaft position is measured using optical encoder sensor. The resolution of the sensor is (1000 pulse/rev). The rotation speed should be displayed on the LCD screen, which is connected to PORTB. (The number of pulses/second is less than 60000 pulses).



Solution:

فى هذا المثال يريد أن يستخدم الـ Timer1 لكى يعمل كـ Counter، ولكن هذه المرة يريد إستخدامه لقياس سرعة دوران ماتور (عدد اللفات/الثانية) وعرض هذه القيمة على شاشة LCD عن طريق الـ Optical encoder الـ كما تم فعل ذلك سابقاً فى المثال (4.2).

فكرة هذا البرنامج هى نفس فكرة البرنامج السابق ولكن سوف نقرأ قيمة العداد من TMR1L والـ TMR1H وذلك بسبب أن عدد الـ Pulses القادمة من الـ Optical encoder فى الثانية الواحدة غالباً سوف تكون أكبر من 255 pulse فى الثانية الواحدة.

ملحوظة مهمة:

بالرغم من الـ Timer1 يعمل فى الـ 16-bit mode فقط، إلا أنه يجب يُعامل نفس معاملة الـ Timer0 وذلك عند تسجيل قيمة على TMR1L أو TMR1H أو القراءة منهما. أى أنه عند تسجيل قيمة على الـ TMR1L والـ TMR1H فإن الـقيمة التى سوف تسجل على الـ TMR1H يجب أن تسبق القيمة التى سوف تسجل على الـ TMR1L والعكس صحيح عن القراءة منهما.

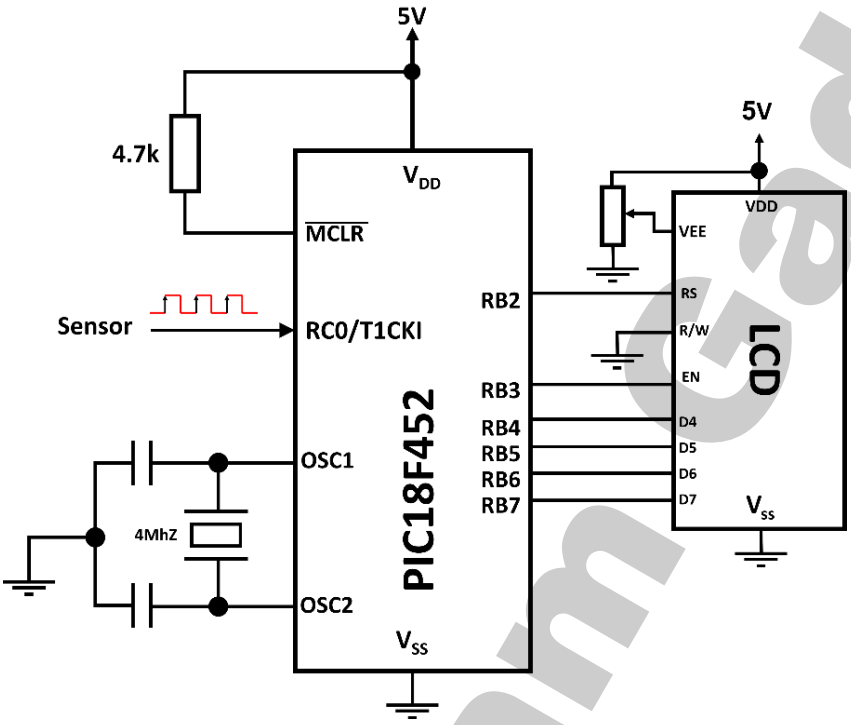
فعلى سبيل المثال:

إذا أردنا تصفير قيمة الـ TMR1L والـ TMR1H فإننا نتبع الترتيب الآتى فى الكود:

```
TMR1H=0;
TMR1L=0;
```

وإذا أردنا قراءة محتويات الـ TMR1L والـ TMR1H فإننا نتبع الترتيب الآتى فى الكود:

```
res=TMR1L;
res=res+TMR1H*256;
```



Program:

```
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;

sbit LCD_RS_Direction at TRISB2_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;
```

```
void main()
{
    float  res;
    char txt[15];

   TRISC.b4=1;
   TRISB=0;
   lcd_Init();                          // إعداد الشاشة
   lcd_cmd(_lcd_clear);                 //مسح الشاشة
   lcd_cmd(_lcd_cursor_off);            //إطفاء المؤشر الذى على الشاشة
   T1CON=0x83;
   while (1)
   {
       TMR1H=0;
       TMR1L=0;
       delay_ms(1000);      // أنتظر لمدة ثانية حتى ينتهى العد

       res=(float)TMR1L;
       res=res+(float)TMR1H*256.0;
       res=res/1000.0;

       lcd_cmd(_lcd_clear);
       lcd_out(1,1,"Rotation speed=");
       floatToStr(res, txt);
       lcd_out(2,1,txt);
   }
}
```

تصفير العداد

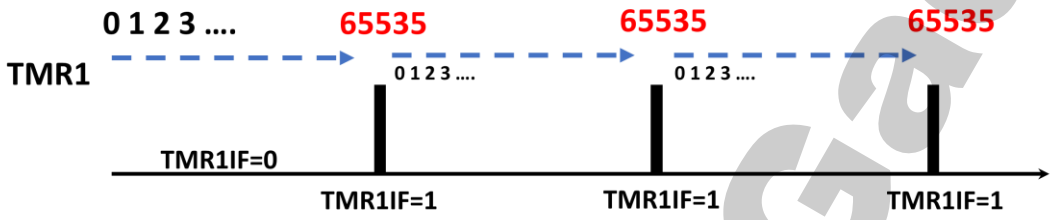
يتم تخزين قيمة العداد فى المخزن res
مع قسمة الناتج على 1000 للحصول
على عدد اللفات فى الثانية والواحدة

1. مسح الشاشة
2. عرض رسالة Rotation speed على السطر الأول
3. تحويل سرعة الدوران إلى characters
4. عرض سرعة الدوران على الشاشة فى السطر الثانى

6.2.2 Operating timer1 as a timer

مثل الـ Timer0، فإنه عند إستخدم الـ Timer1 كمؤقت، فإننا نستخدم النبضات الخاصة بالـ Crystal أى أننا نجعل مصدر النبضات داخلى Internal وذلك عن طريق جعل TMR1CS=0. كما أننا لا نهتم بقيمة الـ Counter هنا بقدر إهتمامنا بالـ Overflow bit الخارج من هذا الـ Counter والمسمى

بإسم TMR1IF. وهى عبارة عن bit تصبح قيمتها تساوى 1 وذلك عندما يصل الـ Counter الخاص بالـ Timer1 إلى أقصى قيمة (65535) كما هو موضح فى الشكل.



والـ Overflow bit الخاص بالـ Timer1 موجود بداخل الـ PIR1 register ويقع تحت أسم TMR1IF.

PIR2 register

PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

ولكن كيف يمكن ظبط الـ Overflow time؟

مثل الـ Timer0، فإن المسؤول عن التحكم فى سرعة الوصول إلى أعلى قيمة للـ Counter ثلاثة أشياء أساسية وهم:

1. تردد الـ Crystal oscillator الواصل بالـ Microcontroller.
2. قيمة الـ Prescaler.
3. القيمة الإبتدائية Initial value للـ Counter الخاص الـ Timer 1.

ويتم الربط بين الثلاثة معاملات السابقين عن طريق المعادلة الآتية:

$$\text{Overflow time (sec)}\big|_{16-bit} = 4 \times T_{osc} \times \text{Prescaler} \times (65536 - \text{initial value of TMR1})$$

وحيث أنه فى المعتاد تكون قيمة الـ Initial value الخاصة بالـ Timer 1 هى المجهولة. لذلك يتم حساب الـ Initial value الخاص بالـ Timer 1 كالآتى:

$$\text{Initial value of TMR1}\big|_{16-bit} = 65536 - \frac{\text{Overflow time (sec)}}{4 \times T_{osc} \times \text{Prescaler}}$$

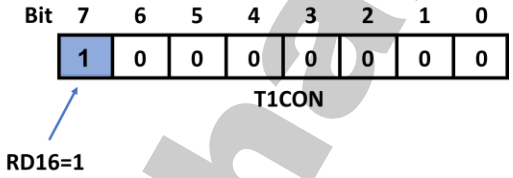
Example (6.9)

By using Timer1, design a microcontroller circuit and write a program to flash a LED with 0.01 second and 0.01 second OFF, which is connected to RD0 pin. The Timer1 is adjusted with a prescaler value of 1:1. The crystal oscillator frequency is 4 Mhz.

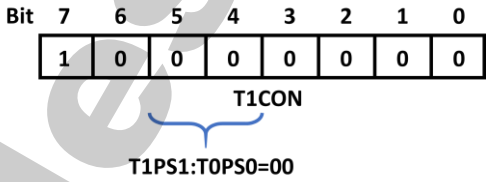
Solution:

فى هذا المثال مطلوب عمل دائرة Flasher لتشغيل وإطفاء LED بمعدل 0.01 ثانية وذلك بدون إستخدام أمر delay_ms. والمطلوب هنا إستخدام الـ Timer 1 لعمل هذه المهمة. لذلك يتم إتباع الخطوات التالية:

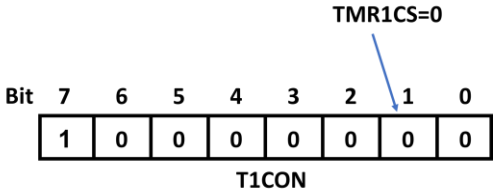
1. يتم جعل قيمة الـ RD16=1، وذلك لذيادة الدقة عند القراءة من الـ TMR1.



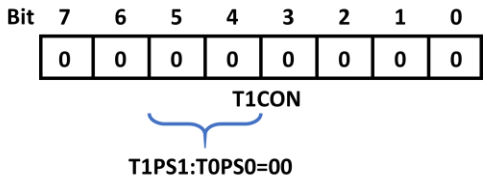
2. يتم جعل قيمة الـ Prescaler=1:1 وذلك عن طريق T1PS1:T1PS0=00 .



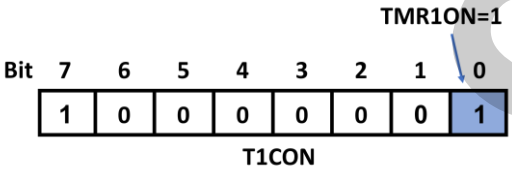
3. يتم ضبط مصدر النبضات Clock source لكى يكون من الداخل Internal عن طريق جعل TMR1CS=0 الموجود بداخل الـ T1CON register.



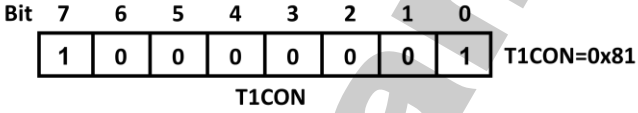
4. يتم جعل قيمة الـ Prescaler=1:1 وذلك عن طريق الـ T1PS1:T1PS0.



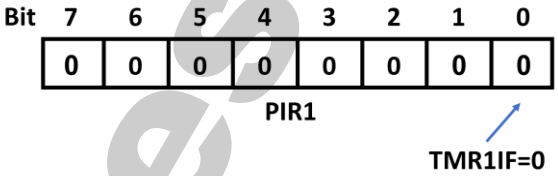
5. يتم تشغيل الـ Timer 1 عن طريق جعل الـ TMR1ON=1.



6. وبذلك تكون المحصلة النهائية لقيمة الـ T1CON register كالآتى:



7. يتم عمل Clear أو مسح لقيمة الـ Overflow bit الخاصة بالـ Timer 1 والتى تسمى TMR1IF عن طريق جعل TMR1IF=0 والموجود داخل الـ PIR1 register.

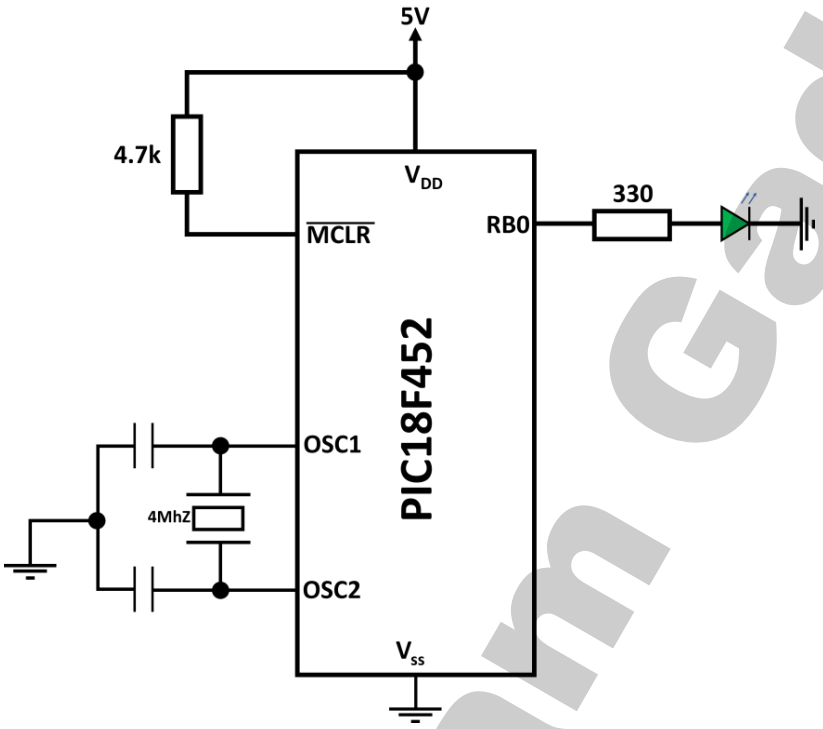


8. يتم حساب القيمة الإبتدائية الخاصة الـ Timer 1 كالآتى:

$$\text{Initial value of TMR1}|_{16-bit} = 65536 - \frac{\text{Overflow time (sec)}}{4 \times T_{osc} \times \text{Prescaler}} =$$

$$= 65536 - \frac{0.01}{4 \times 0.25 \times 10^{-6} \times 1} = 55536 = 0xD8F0$$

9. يتم الإنتظار حتى تنتهى المدة المطلوبة وذلك عن طريق التحقق بشكل دورى على قيمة الـ TMR1IF، فإذا أصبحت تساوى 1 فهذا يعنى إنتهاء المدة المطلوبة.



Program:

```
void main()
{
    TRISD.b0=0;                 //ضبط طرف الخرج
    PORTD.b0=0;                 //تصفير طرف الخرج
    T1CON=0x81;                 //ضبط التايمر

    while(1)
    {
        TMR1H=0xD8;
        TMR1L=0xF0;
        PIR1.TMR1IF=0;
        while(PIR1.TMR1IF==0);

        PORTD.b0=~PORTD.bo;                    // إعكس حالة الليد
    }
}
```

1. وضع القيمة الإبتدائية فى عداد التايمر
2. تصفير الـ TMR1IF
3. الإنتظار حتى يصل التايمر إلى نهايته

Example (6.10)

By using Timer1, design a microcontroller circuit and write a program to flash a LED with 1 second and 1 second OFF, which is connected to RD0 pin.

Solution:

فى هذا المثال مطلوب عمل دائرة Flasher لتشغيل وإطفاء LED بمعدل 1 ثانية وذلك بدون إستخدام أمر delay_ms. ولكن هذه المرة لم يحدد قيمة الـ Prescaler الخاص بالتايمر. فكيف يمكن حساب هذه الأشياء؟

يتم أولاً فرض فيمة الـ Prescaler ولتكن 1:1 المطلوب هنا إيجاد الـ Initial value للـ Timer 1. يُلاحظ هنا أن الـ Initial value للـ Timer 1 سالبة وهذا غير مقبول.

$$\text{Initial value of TMR1}\big|_{16-bit} = 65536 - \frac{\text{Overflow time (sec)}}{4\times T_{osc}\times \text{Prescaler}} =$$

$$= 65536 - \frac{1}{4\times 0.25\times 10^{-6}\times 1} = -934464$$

وحتى إذا تم تجربة أعلى قيمة للـ Prescaler ولتكن 1:8، فإن الـ Initial value للـ Timer 1 سوف تكون:

$$\text{Initial value of TMR1}\big|_{16-bit} = 65536 - \frac{\text{Overflow time (sec)}}{4\times T_{osc}\times \text{Prescaler}} =$$

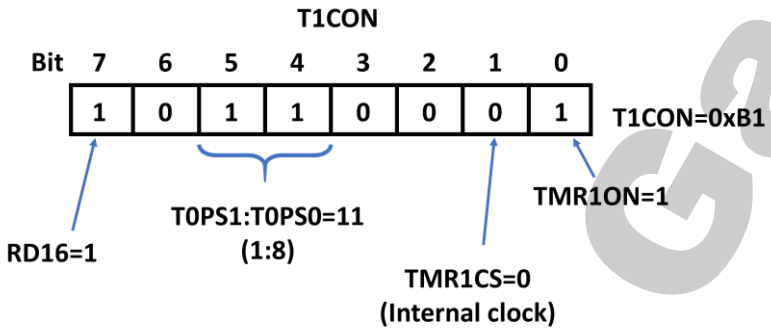
$$= 65536 - \frac{1}{4\times 0.25\times 10^{-6}\times 8} = -59464$$

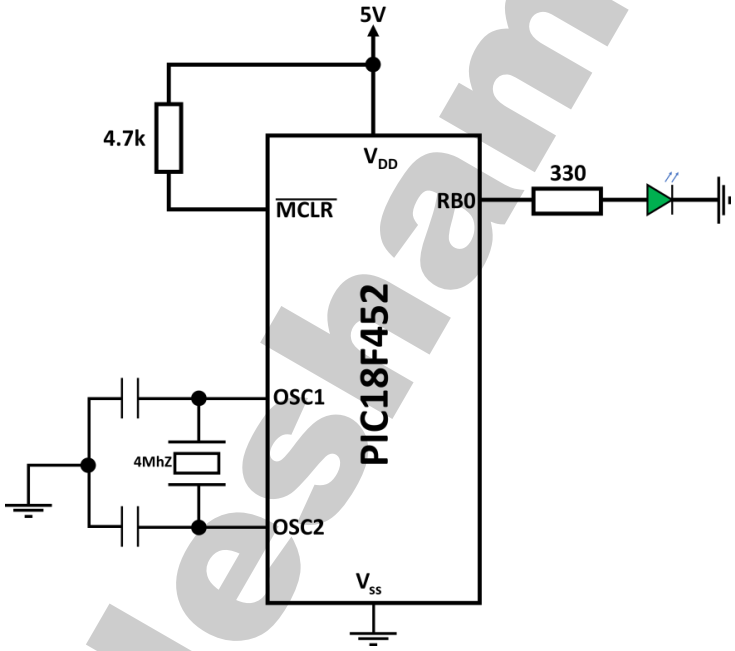
القيمة التى تم الحصول عليها قيمة سالبة، وهذا يعنى أن الـ Timer1 لايمكنه إعطاء 1 second (بوجود الـ 4 Mhz crystal). لذلك سوف يتم تجربة أزمنه أخرى أقل ولتكن 0.5 sec مع فرض أن قيمة prescaler=1:8.

$$\text{Initial value of TMR1}\big|_{16-bit} = 65536 - \frac{\text{Overflow time (sec)}}{4\times T_{osc}\times \text{Prescaler}} =$$

$$= 65536 - \frac{0.5}{4\times 0.25\times 10^{-6}\times 8} = 3036 = 0x0BDC$$

وبذلك سوف تكون قيمة الـ TMR1L=0xDC والـ TMR1H=0x0B. كما أن قيمة الـ T1CON register سوف تكون كالآتى:





Program:

```
void main()
{
int i;
TRISD.b0=0;
PORTD.b0=0;

T1CON=0xB1;
```

```
    while (1)
    {
        for(i=0;i<=1;i++)
        {
           TMR1H=0x0B;
           TMR1L=0xDC;
           PIR1.TMR1IF=0;
           while (PIR1.TMR1IF==0);
        }

        PORTD.b0=~ PORTD.b0;
    }
}
```

1. وضع القيمة الإبتدائية فى عداد التايمر
2. تصفير الـ TMR1IF
3. الإنتظار حتى يصل التايمر إلى نهايته