# Estudo das Interrupções

Uma interrupção suspende a execução de um programa, salva informações do contexto (principalmente informações dos registradores WREG, STATUS e BSR), desvia o programa para que a interrupção ocorrida seja tratada e, após o tratamento, as informações de contexto salvas, são retornadas e a execução do programa retorna à partir de onde foi interrompido.
O PIC 18F4520 possui 20 fontes de interrupção e uma característica que permite priorizar a execução das interrupções, ou seja, podemos definir as interrupções como de alta ou baixa prioridade.
Somente para constar, o vetor de alta prioridade tem o endereço 00008h e o de baixa prioridade tem o endereço 00018h. Assim, quando ocorre uma interrupção e, se definimos esta interrupção como de alta prioridade, o contador de programa (PC – program counter) apontará para o endereço 00008h e lá será tratada a interrupção ocorrida. Da mesma forma, ao ocorrer uma interrupção de nível baixo, o PC apontará para o endereço 00018h para que a interrupção ocorrida seja tratada.
Se não definirmos nenhuma prioridade para a interrupção, estas, quando ocorrerem, o apontamento do PC será para o endereço 00008h.

**Tipos de interrupções disponíveis:**

***-Interrupção externa;***

Ocorre quando um sinal externo é gerado no pino 0 da porta B. Este pino deve ser configurado como entrada.

***-Interrupção externa 1;***

Ocorre quando um sinal externo é gerado no pino 1 da porta B. Este pino deve ser configurado como entrada.

***-Interrupção externa 2;***

Ocorre quando um sinal externo é gerado no pino 2 da porta B. Este pino deve ser configurado como entrada.

***-Interrupção por mudança de estado;***

Ocorre quando existe a mudança de estado nos pinos 4, 5, 6 e 7 da porta B. Estes pinos devem ser configurados como entrada.

***-Interrupção de porta paralela (PSP);***

Ocorre quando uma operação de escrita ou leitura da porta paralela do tipo escravo (PSP – Parallel Slave Port) é completada.

***-Interrupção dos conversores A/D;***

Ocorre quando uma conversão A/D (analógica/Digital) é completada.

***-Interrupção de recepção da USART;***

Ocorre quando a recepção de um dado pela USART (Universal Synchronous Asynchronous Receiver Transmitter) é terminado.

***-Interrupção de transmissão da USART;***

Ocorre quando a transmissão de uma dado pela USART (Universal Synchronous Asynchronous Receiver Transmitter) é terminado.

***-Interrupção da comunicação serial (SPI e I2C);***

Ocorre sempre que um dados é transmitido ou recebido por estes modos de comunicação serial SPI ou I2C.

***-Interrupção do módulo CCP1(Capture/Compare/PWM);***

Interrupção vinculada ao sistema CCP1.

***-Interrupção do Timer 0;***

Ocorre sempre que acontecer um estouro de contagem no contador TMR0.

***-Interrupção do Timer 1;***

Ocorre sempre que acontecer um estouro de contagem no contador TMR1.

***-Interrupção do Timer 2;***

Ocorre sempre que acontecer um estouro de contagem no contador TMR2.

***-Interrupção do Timer 3;***

Ocorre sempre que acontecer um estouro de contagem no contador TMR3.

***-Interrupção de fim de escrita na EEPROM/FLASH;***

Ocorre sempre que a rotina de escrita na EEPROM ou na memória FLASH é finalizada.

***-Interrupção de colisão de dados (Bus Collision);***

Informa ao sistema sobre colisões de dados na comunicação I2C.

***-Interrupção do CCP2(Capture/Compare/PWM);***

Interrupção vinculada ao sistema CCP1.

***-Interrupção dos comparadores.***

Ocorre quando existe a mudança de estado nos comparadores internos do PIC.

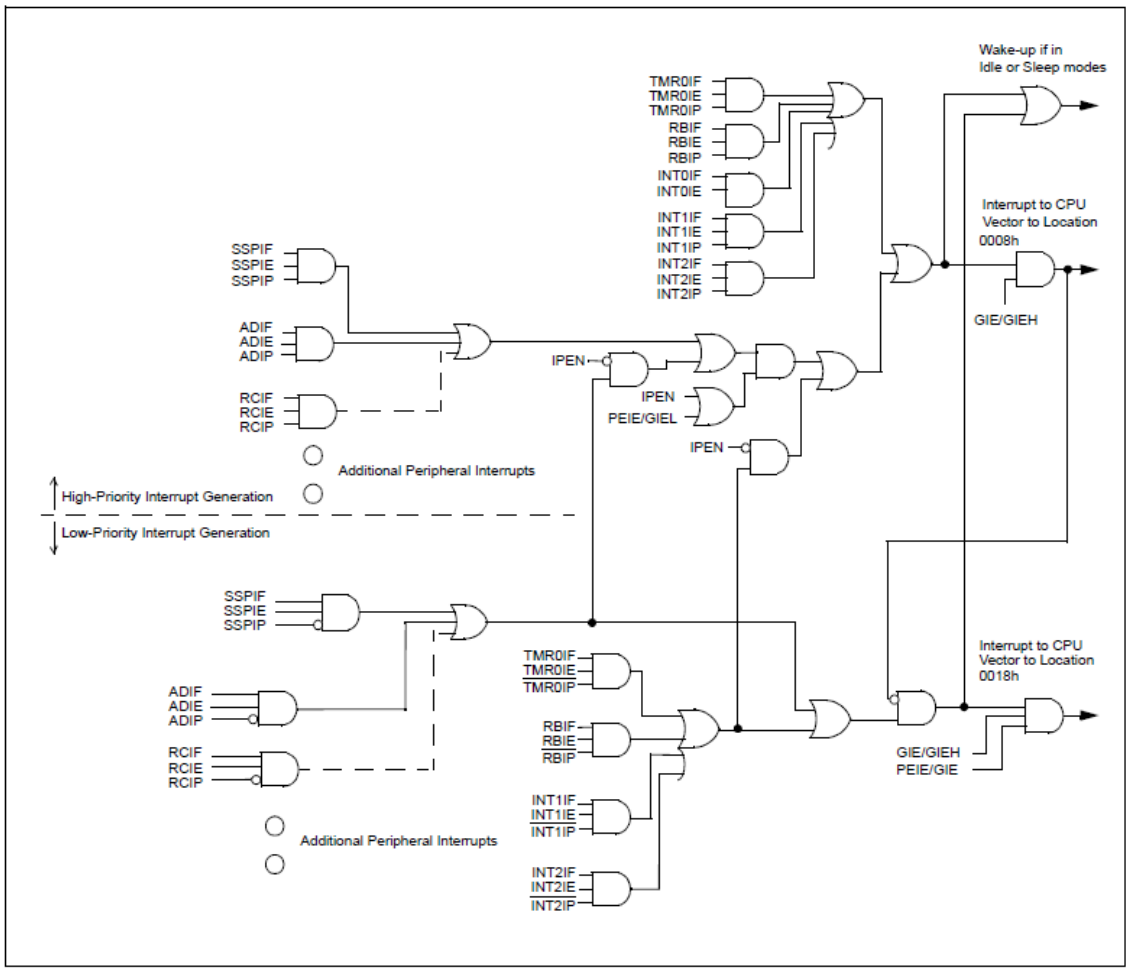
***-Interrupção de detecção de falha no oscilador.***

Ocorre quando ocorre alguma falha no oscilador externo.

***-Interrupção de alta/baixa tensão.***

Ocorre quando existe a detecção de alta ou baixa tensão determinada pelo bit VDIRMAG do registrador HLVDCON.

**Representação da lógica das interrupções:**




| Chaves | Registrador | Bit | Flags | Registrador | Bit |
|---|---|---|---|---|---|
| GIE | INTCON | 7 | - | - | - |
| PEIE | INTCON | 6 | - | - | - |
| TMR0IE | INTCON | 5 | TMR0IF | INTCON | 2 |
| INT0IE | INTCON | 4 | INT0IE | INTCON | 1 |
| RBIE | INTCON | 3 | RBIF | INTCON | 0 |
| INT1IE | INTCON3 | 3 | INT1IF | INTCON3 | 0 |
| INT2IE | INTCON3 | 4 | INT2IF | INTCON3 | 1 |
| TMR1IE | PIE1 | 0 | TMR1IF | PIR1 | 0 |
| TMR2IE | PIE1 | 1 | TMR2IF | PIR1 | 1 |
| TMR3IE | PIE2 | 1 | TMR3IF | PIR2 | 1 |
| CCP1IE | PIE1 | 2 | CCP1IF | PIR1 | 2 |
| CCP2IE | PIE2 | 0 | CCP2IF | PIR2 | 0 |
| CMIE | PIE2 | 6 | CMIE | PIR2 | 6 |
| EEIE | PIE2 | 4 | EEIF | PIR2 | 4 |

| | | | | | |
|---|---|---|---|---|---|
| SSPIE | PIE1 | 3 | SSPIF | PIR1 | 3 |
| TXIE | PIE1 | 4 | TXIF | PIR1 | 4 |
| RCIE | PIE1 | 5 | RCIF | PIR1 | 5 |
| ADIE | PIE1 | 6 | ADIF | PIR1 | 6 |
| HLVDIE | PIE2 | 2 | HLVDIF | PIR2 | 2 |
| SPPIE | PIE1 | 7 | SSPIF | PIR1 | 7 |
| BCLIE | PIE2 | 3 | BCLIF | PIR2 | 3 |
| OSCFIE | PIE2 | 7 | OSCFIF | PIR2 | 7 |

**Função de tratamento das interrupções**

Para que aconteça uma interrupção, devemos ter em mente alguns itens:

-Configuração da interrupção, ou seja, habilitação dos bits referentes à interrupção pretendida, além da habilitação da interrupção global e/ou de periféricos;
-Ao acontecer a interrupção, o *flag* relacionado àquela interrupção é setado (levado para 1);
-Acontece o desvio do programa para a função de tratamento da interrupção;
(Neste momento o compilador faz o salvamento do contexto e também o salvamento ou restauração dos registradores WREG, STATUS e BSR)
-A interrupção é identificada;
-O *flag* relacionado da interrupção acontecida deve ser limpo (levado para 0);
-A interrupção é tratada;
-O programa volta para a condição após a ocorrência da interrupção.

## Tratamento das interrupções no compilador C18

A configuração da interrupção pode ser feita logo na função principal do programa, onde devem ser habilitados os bits referentes à interrupção, se for o caso, habilitar a interrupção dos periféricos e também habilitar a interrupção global.
Toda vez que ocorrer uma interrupção, o programa será desviado para o endereço cujo qual deverá conter a função de tratamento desta (0x00008 para interrupção de alta prioridade e 0x00018 para interrupção de baixa prioridade).

**Diretiva de tratamento das interrupções**

Para declarar uma função de alta prioridade, a diretiva ***#pragma interrupt*** deve ser utilizada e para uma interrupção de baixa prioridade, usa-se a diretiva **#pragma interruptlow**.

Estas diretivas não posiciona as funções de tratamento das interrupções nos endereços corretos referentes aos vetores de interrupção. Então, através da diretiva ***#pragma code***, pode-se informar que a função de tratamento de uma interrupção deverá estar no endereço informado.

**Exemplo:**

```
#pragma code int_h = 0x00008
#pragma interrupt trata_interrupt
void trata_interrupt(void)
{
.....
}
```

A diretiva ***#pragma code*** neste exemplo, informa o endereço 0x00008 para a função de tratamento das interrupções de alta prioridade (trata_interrupt(void).

**Tratando uma interrupção de alta prioridade**

```
#pragma code int_hr = 0x000008      //vetor de interrupção de alta prioridade
#pragma interrupt trata_interrupt   /*define a função trata_interrupt como rotina de
                                    tratamento da interrupção*/
void trata_interrupt(void)          //Função de tratamento da interrupção
{
  //Zera o flag da interrupção
  //Rotina de tratamento.....
}
```

**Tratando uma interrupção de baixa prioridade**

```
#pragma code int_hr = 0x000018        //vetor de interrupção de alta prioridade
#pragma interruptlow trata_interrupt  /*define a função trata_interrupt como rotina
                                      ... de tratamento da interrupção*/
void trata_interrupt(void)            //Função de tratamento da interrupção
{
  //Zera o flag da interrupção
  //Rotina de tratamento.....
}
```

**Tratando várias interrupções de alta prioridade**

```
#pragma interrupt trata_INT_1      /*define a função trata_INT_1 como rotina de
                                   ...tratamento da interrupção*/
void trata_INT_1(void)             //Função trata_INT_1
{
  //Zera o flag da interrupção
  //Rotina de tratamento.....
}

#pragma interrupt trata_INT_2      /*define a função trata_INT_2 como rotina de
                                   ...tratamento da interrupção*/
void trata_INT_2(void)             //Função trata_INT_2
{
   //Zera o flag da interrupção
   //Rotina de tratamento.....
}

#pragma code isr = 0x000008        //vetor de interrupção de alta prioridade
void Mult_INT(void)                //função Mult_INT
{
        if (... flag INT_1 == 1)  _asm BRA trata_INT_1  _endasm
        if (... flag INT_2 == 1)  _asm BRA trata_INT_2  _endasm
}
```

Note que na função Mult_INT, são verificados os flags de cada interrupção habilitada para identificar a fonte da interrupção ocorrida e quando identificado, o programa é desviado para a função de tratamento correta. Este desvio, neste caso, é feito utilizando-se instruções em *assembly* (BRA – desvio incondicional).
O desvio para o tratamento da interrupção é feito para que o salvamento do contexto e

dos registradores STATUS, WREG e BSR sejam corretos, o que acontece quando a diretiva ***#pragma interrupt ....*** é utilizada.

```
void Mult_INT(void)                    //função Mult_INT
{
        if (... flag INT_1 == 1)  _asm BRA trata_INT_1  _endasm
        if (... flag INT_2 == 1)  _asm BRA trata_INT_2  _endasm
}
```

Uma opção funcional, porém não tão eficiente para este tratamento pode ser vista a seguir:

```
#pragma code isr = 0x000008        //vetor de interrupção de baixa prioridade
#pragma interrupt Mult_INT         /*define a função Mult_INT como rotina
                                    ... de tratamento da interrupção*/
void Mult_INT(void)                //função Mult_INT
{
 if (... flag INT_1 == 1)
        {
        //Zera o flag da interrupção
        //Rotina de tratamento.....
        }

if (... flag INT_2 == 1)
        {
        //Zera o flag da interrupção
        //Rotina de tratamento.....
        }
}
```

## Interrupção Externa

O Registrador **INTCON** é o responsável pelo controle de algumas das interrupções no PIC. Abaixo são mostrados os bits de habilitação e o flag de controle para a interrupção externa.

| Registrador INTCON | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Nome | GIE/GIEH | PEIE/GIEL | TMR0IE | INT0IE | RBIE | TMR0IF | INT0IF | RBIF |

**GIE = 1:** Habilita a interrupção global.
**GIE = 0:** Desabilita a interrupção global.

**INT0IE = 1:** Habilita a interrupção externa.
**INT0IE = 0:** Desabilita a interrupção externa.

**INT0IF** – Flag de sinalização da ocorrência da interrupção externa, este Flag deverá ser limpo (levado à 0) quando do tratamento da interrupção.
**INT0IF = 1:** Houve overflow no Timer 0.
**INT0IF = 0:** Não houve overflow no Timer 0.

**Obs.:** Nesta interrupção não será necessário habilitar o bit de interrupção de periféricos, PEIE.
Deve-se configurar a interrupção externa na borda de descida do sinal aplicado ao pino 0 da porta B. Este sinal, devido ao resistor de Pull Up, está em nível alto e quando pressionamos o botão RB0, o nível de sinal aplicado ao pino 0 da porta B irá para 0.
Podemos configurar no registrador **INTCON2,** no bit **INTEDG0,** a borda de ativação da interrupção externa, conforme abaixo:

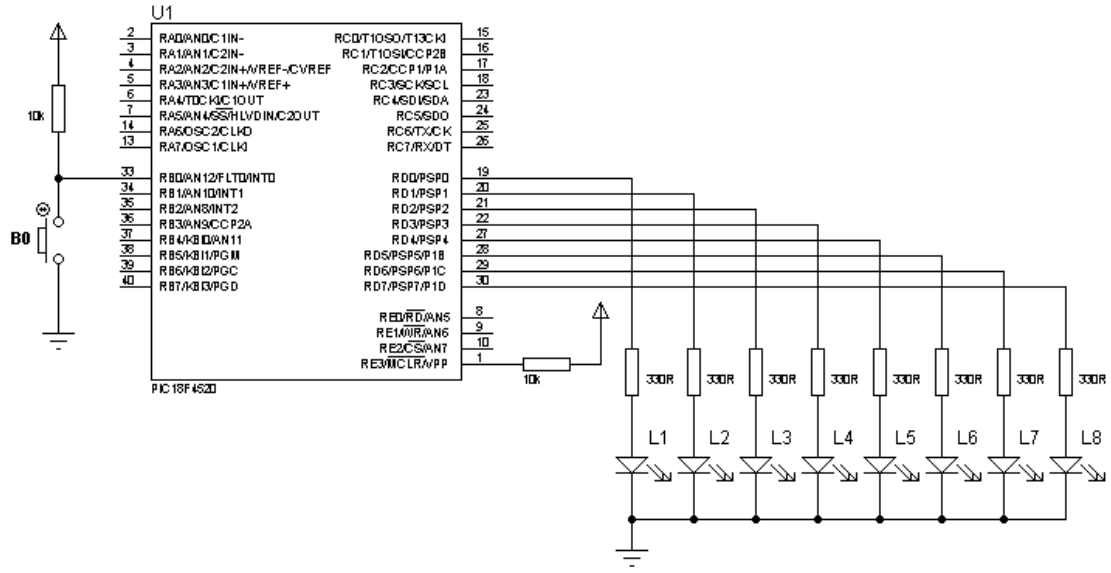
**INTEDG0 = 0:** Poderá ocorrer a interrupção na borda de descida do sinal no pino 0 da porta B (RB0/INT).
**INTEDG0 = 1:** Poderá ocorrer a interrupção na borda de subida do sinal no pino 0 da porta B (RB0/INT).

Para o nosso exemplo, configuraremos o bit INTEDG0 em 0:

**Obs.:** Não podemos esquecer de configurar os bits 0 à 4 da porta B como entrada/saída digital, assim como informado no exemplo de acionamento de botões.

**Projeto:**
Utilizando a interrupção externa, toda vez que o botão B0 for pressionado, o estado dos LEDs conectados à porta D deverão ser alterados, ou seja, se acesos, devem apagar e vice-versa.

**Circuito:**




**Acesso direto aos bits dos registradores:**

Primeiramente, antes de iniciarmos a escrita do código do programa, vamos verificar como podemos acessar somente um bit em um determinado registrador.

**Sintaxe:**
<registrador>**bits.**<bit>

**Exemplos:**

```
TRISDbits.TRISD3 = 1;          //Colocamos em 1 somente o bit 3 do registrador TRISD
INTCONbits.INT0IE = 1;         //Habilita interrupção externa INT0
INTCON2bits.INTEDG0 = 0;       //Habilita interrupção na borda de descida
INTCONbits.GIE = 1;            //Habilita todas as interrupções de alta prioridade
```

**Definições do Projeto:**

-Nome do projeto: Int_Ext0

**Código do programa:**

```
#include <p18F4520.h>

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us

#pragma config OSC = HS          //Configura o oscilador a cristal
#pragma config WDT = OFF         //Desabilita o Watchdog Timer (WDT).
#pragma config PWRT = ON         //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON        //Habilita Brown-out reset
#pragma config BORV = 1          //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF      //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF         //Desabilita o Low Voltage Program.

#pragma code int_h = 0x000008    //vetor de interrupção de baixa prioridade
#pragma interrupt INT_EXT0       /*define a função INT_EXT0 como rotina de
                                ....tratamento da interrupção externa*/
void INT_EXT0(void)              //função INT_TEXT0
{
  INTCONbits.INT0IF = 0;         //Limpa o flag de interrupção externa
  PORTD = ~PORTD;                //inverte o estado da PORTA D
}

void main()
{
TRISBbits.TRISB0 = 1;            //Configura o pino 0 da porta B como entrada
TRISD = 0x00;                    //Faz toda a porta D como saída
PORTD = 0x00;                    //Limpa toda a porta D

INTCONbits.INT0IE = 1;           //Habilita interrupção externa INT0
INTCON2bits.INTEDG0 = 0;         //Habilita interrupção na borda de descida

INTCONbits.GIE = 1;              //Habilita todas as interrupções

while(1)
        {
        }
}
```

**Entendendo o programa:**

Primeiramente, na função principal, configuramos o bit 0 da porta B como entrada (botão) e, em seguida, configuramos, como saída, todos os pinos da porta D através do registrador TRISD e logo zeramos todos os pinos da porta D para que todos os LEDs conectados a ele se apaguem:

```
TRISBbits.TRISB0 = 1;        //Configura o pino 0 da porta B como entrada
TRISD = 0x00;                //Faz toda a porta D como saída
PORTD = 0x00;                //Limpa toda a porta D
```

Em seguida, fazemos a configuração dos bits de controle da interrupção externa, como segue.

Habilitando a interrupção externa:

```
INTCONbits.INT0IE = 1;       //Habilita interrupção externa INT0
```

Definindo que a interrupção deverá ocorrer na borda de descida do sinal no pino RB0:

```
INTCON2bits.INTEDG0 = 0;     //Habilita interrupção na borda de descida
```

Habilitando a interrupção global:

```
INTCONbits.GIE = 1;          //Habilita todas as interrupções
```

Note também que no loop principal, só deixamos a sua indicação que, conforme visto anteriormente, é necessária para que o microcontrolador não fique 'ressetando':

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

**Tratamento da interrupção**

No tratamento da interrupção externa, temos abaixo a diretiva de tratamento da interrupção e a função de tratamento desta:

```
#pragma code int_h = 0x000008      //vetor de interrupção de baixa prioridade
#pragma interrupt INT_EXT0         /*define a função INT_EXT0 como rotina de
                                  ....tratamento da interrupção externa*/
void INT_EXT0(void)                //função INT_TEXT0
{
  INTCONbits.INT0IF = 0;           //Limpa o flag de interrupção externa
  PORTD = ~PORTD;                  //inverte o estado da PORTA D
}
```

Como já informado, a interrupção externa através do bit 0 da porta B (INT0) não suporta nível de prioridade, assim, quando esta ocorre, o programa será desviado para o endereço 0x000008h.
A diretiva **#pragma code int_h = 0x00008** direciona para o endereço do vetor de prioridade alta.
Logo em seguida, alterando a diretiva, ***pragma*** será definida a função de tratamento para a interrupção externa (***#pragma interrupt INT_EXT0***).
Na função INT_EXT0 será então tratada a interrupção, sendo que o flag da interrupção externa será zerado e em seguida o estado da porta D será alterado, ou seja, se todos os bits estiverem em 1, todos passarão para 0 e vice-versa.

Basta então, compilar o projeto e gravar o microcontrolador para verificar o funcionamento.

# TIMER 0

O Timer 0 pode operar como contador ou temporizador de 8 ou 16 bits e a sua fonte de clock pode ser configurada para ser obtida do sinal de clock interno (fosc/4) ou do sinal externo aplicado ao pino RA4/T0CKI.

Diagrama de blocos do Timer 0 operando no modo 8 bits:

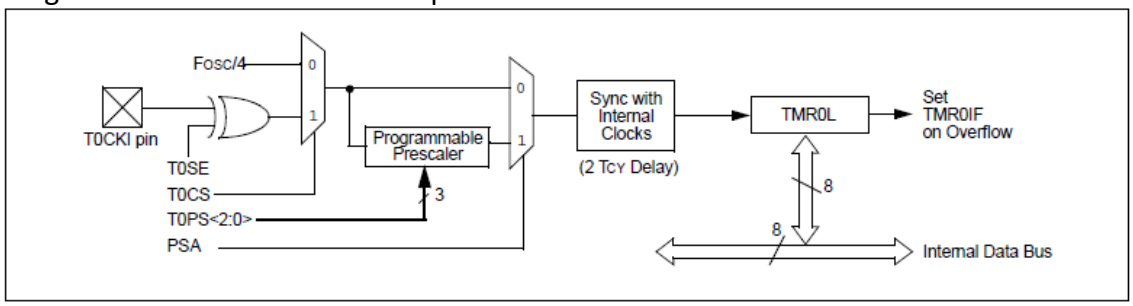
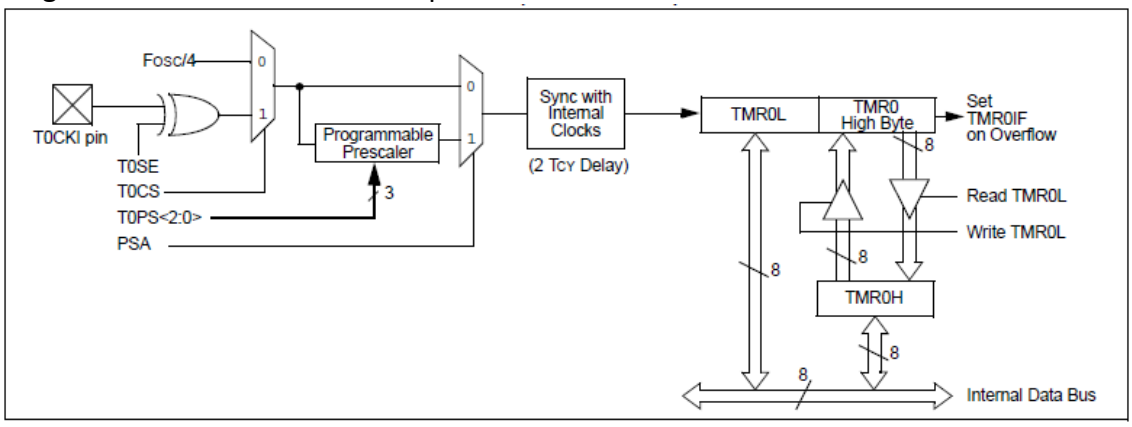



Diagrama de blocos do Timer 0 operando no modo 16 bits:




O Timer 0 possui um ***prescaler*** de 1:2 até 1:256, conforme a tabela apresentada nos bits de configuração T0PS2, T0PS1 e T0PS0 do registrador T0CON.
O ***prescaler*** define o número de vezes que um evento deve ocorrer, antes que o registrador TMR0 (8 bits) ou TMR0L + TMR0 (16 bits) seja incrementado.
Supondo que o módulo TIMER 0 reconheça um evento na borda de descida do sinal e o prescaler está configurado com 1:2, então o incremento no registrador TMR0 se dará somente a cada duas bordas de descida do sinal.
Sendo assim, se configurarmos este *prescaler* para 1:2 e selecionarmos o timer para operação em 8 bits, por exemplo, então haverá um *overflow* (estouro da contagem) a cada 512 contagens, ou seja, a contagem de 0 à 255 (256 no total dos 8 bits) vezes 2 do *prescaler*. Portanto, selecionando o TIMER 0 para operação em 16 bits, então o overflow ocorrerá a cada 131072 contagens, ou a contagem 0 à 65535 (65536 no total dos 16 bits) vezes 2 do *prescaler*.
Quando ocorre o overflow, o bit TMR0IF do registrador INTCON vai para 1, indicando, assim a ocorrência do estouro na contagem do TIMER.

Para a configuração do Timer 0, utilizamos o registrador T0CON.

| Registrador T0CON – Timer 0 Control Register | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Bit** | **Bit 7** | **Bit 6** | **Bit 5** | **Bit 4** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| **Nome** | TMR0ON | T08BIT | T0CS | T0SE | PSA | T0PS2 | T0PS1 | T0PS0 |

**Configuração dos bits de controle do Timer 0:**

**Bit TMR0ON - Liga/desliga Timer 0.**
**TMR0ON = 1:** Habilita Timer 0.
**TMR0ON = 0:** Desabilita Timer 0.

**Bit T08BIT - Seleciona o Timer 0 para operação em 8 ou 16 bits.**
**T08BIT = 1:** Seleciona o Timer 0 para operação em 8 bits.
**T08BIT = 0:** Seleciona o Timer 0 para operação em 16 bits.

**T0CS = 1:** Incremento a cada transição no pino RA4/T0CKI (clock externo).
**T0CS = 0:** Incremento a cada ciclo de máquina (Fosc/4).

**Bit T0SE – Timer 0 Select Edge**
**T0SE = 1:** Incremento na borda de descida.
**T0SE = 0:** Incremento na borda de subida.

**PSA = 1:** Entrada de clock do TIMER0 direta (sem prescaler).
**PSA = 0:** Entrada de clock passa pelo *Prescaler*.

**PS2, PS1 e PS0** - Configuração do *Prescaler.*

| PS2 | PS1 | PS0 | TMR0 |
|---|---|---|---|
| 0 | 0 | 0 | 1:2 |
| 0 | 0 | 1 | 1:4 |
| 0 | 1 | 0 | 1:8 |
| 0 | 1 | 1 | 1:16 |
| 1 | 0 | 0 | 1:32 |
| 1 | 0 | 1 | 1:64 |
| 1 | 1 | 0 | 1:128 |
| 1 | 1 | 1 | 1:256 |

**OBS.:** Quando PSA = 1, podemos entender como um prescaler de 1:1.

## Interrupção do Timer 0

A interrupção do Timer 0 ocorre quando acontece o estouro na contagem do registrador TMR0L (8 bits) e TMR0L + TMR0 (16 bits), ou seja, quando a contagem do registrador ultrapassar o valor máximo possível, sendo o valor máximo igual a 256, se configurado para 8 bits e 65536, se configurado para 16 bits.
A habilitação da interrupção do TIMER 0 é feita colocando-se em 1 o bit 5 (TMR0IE) do registrador INTCON e o bit de controle da interrupção, ou seja, o bit que informará a ocorrência do overflow, é o bit TMR0IF (bit 2 do registrador INTCON).
Juntamente com a habilitação do Timer 0, precisamos também habilitar a interrupção global (GIE, Bit 7 do registrador INTCON).
Quando ocorrer a interrupção, o bit TMR0IF vai para 1 informando a ocorrência, assim o programa, como vimos antes, é desviado para a função de tratamento da interrupção e lá deve ser tratada e, novamente, colocar em nível lógico 0 o bit TMR0IF.

## Calculando o tempo total para a ocorrência da interrupção do TIMER 0 para o modo 8 bits.

Podemos calcular o tempo para a ocorrência da interrupção no TIMER 0, para o modo de funcionamento em 8 bits, seguindo a seguinte fórmula:

**Tempo = 256 x prescaler x ciclo de máquina**

**Exemplo:**
Para um cristal de 8MHz, teremos o ciclo de máquina igual a 0,5 us;
Utilizaremos o prescaler de 1:2, então:

**Tempo = 256 x 2 x 0,5u;**

**Tempo = 256 us.**

Para um prescaler de 256 e mantendo o cristal de 8MHz, teremos:

**Tempo = 256 x 256 x 0,5u;**

**Tempo = 32768 us ou 32,768ms**

## Contando um tempo de 1 segundo no modo 8 bits

Como sabemos, 1 segundo é o equivalente a 1000ms, porém o tempo máximo que conseguimos chegar com o máximo prescaler para o TIMER 0 operando em 8 bits é de 32,768ms. Sendo assim, precisamos então dividir estes 1000ms para que possamos adequar à realidade do nosso PIC.
Bem, a divisão de 1000ms por 32,768ms resulta em 30,5175.... Este resultado, por ser um valor "quebrado" não nos ajudará muito, então vamos dividir o tempo de 1000ms por 25ms. O resultado agora será 40. ...Guardaremos este resultado para ser usado posteriormente.

Então, precisamos fazer com que o estouro na contagem do registrador TMR0 ocorra a cada 25ms... Verificamos também que este tempo se "encaixa" no prescaler 1:256, pois para o prescaler abaixo deste, ou seja igual a 128, o tempo máximo conseguido será de 16,384ms (menor do que o valor desejado).
Basta-nos encontrar o valor de início para o registrador TMR0 a fim de que o estouro na contagem deste ocorra conforme a nossa necessidade.
Podemos lançar mão da fórmula abaixo para definirmos este valor de início para o registrador TMR0:

$$TMR0L = 256 - \left( \frac{\text{tempo desejado}}{\text{ciclo de máquina x prescaler}} \right)$$

Atribuindo os valores na fórmula temos:

$$TMR0L = 256 - \left( \frac{25\ ms}{0{,}5\ us \times 256} \right)$$

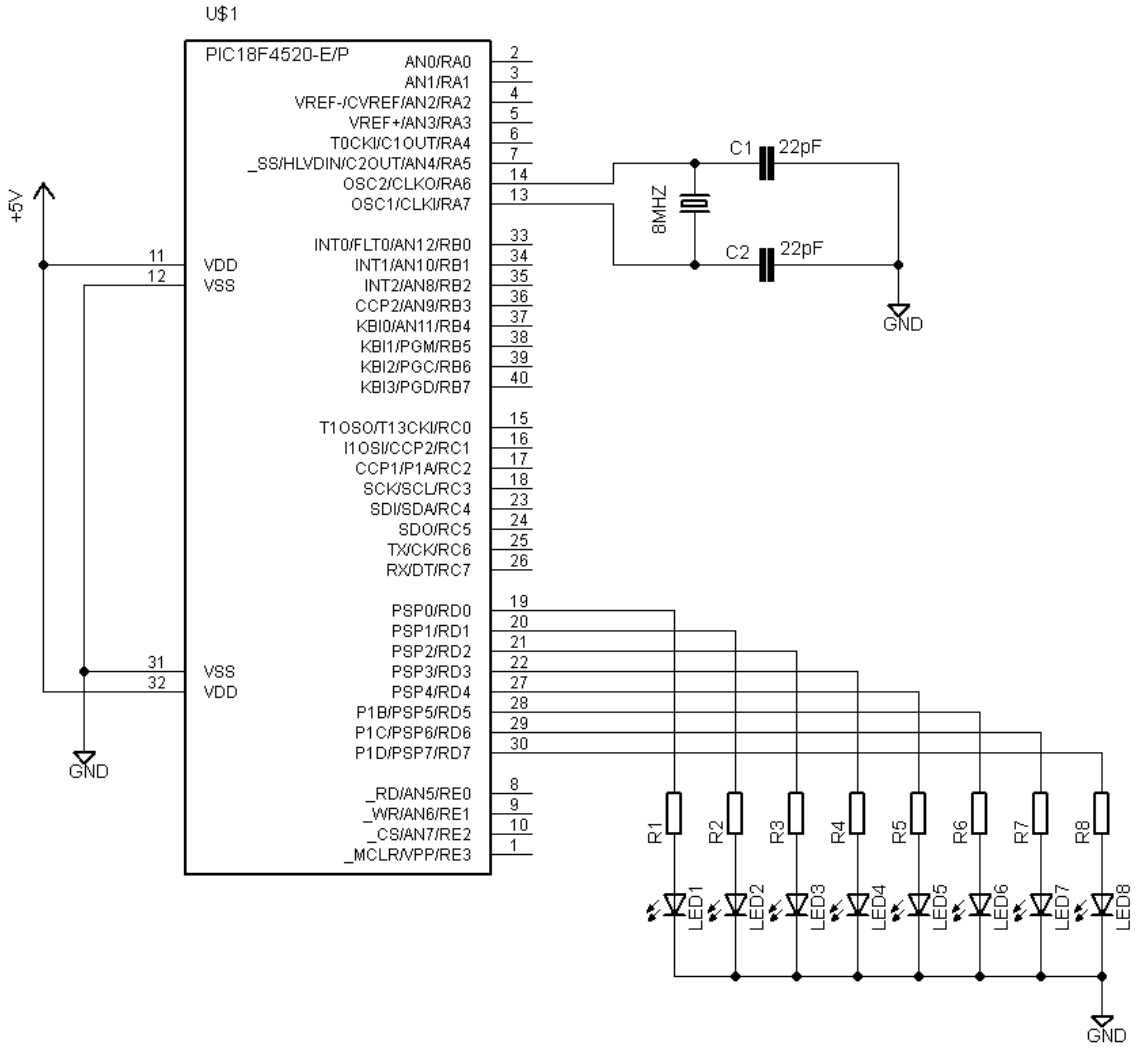
**TMR0L = 60,69**

O resultado é um valor fracionário, mas podemos aproximá-lo para 61. Então o valor a ser atribuído ao registrador TMR0L a fim de iniciá-lo será 61.
De posse dos dados obtidos, podemos iniciar a escrita do código para programação do microcontrolador.

**Utilizando a Interrupção do Timer 0 no modo 8 bits:**

Vamos fazer com que o led L1 (conectado ao pino RD0) pisque a cada 1 segundo e a cada, aproximadamente 500ms, o outro led L2 (pino RD1) pisque também.
Como já calculado antes, o valor 61 será armazenado no registrador TMR0L. Com esse valor, verificamos que a interrupção ocorrerá a cada 25 ms, portanto para obtermos o tempo de 1 segundo, basta multiplicarmos o número de interrupções por 40 e para obtermos 500ms, basta multiplicarmos as interrupções por 20.

**Circuito:**



**Código do projeto:**

```
#include <p18F4520.h>

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us

#pragma config OSC = HS          //Configura o oscilador a cristal
#pragma config WDT = OFF         //Desabilita o Watchdog Timer (WDT).
#pragma config PWRT = ON         //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON        //Habilita Brown-out reset
#pragma config BORV = 1          //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF      //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF         //Desabilita o Low Voltage Program.

char conta_500=0, conta_1000=0;  //Variáveis de controle de tempo

#pragma code isr = 0x000008      //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER0   /*define a função trata_TIMER0 como rotina de
                                ...tratamento da interrupção do Timer 0*/
void trata_TIMER0(void)          //Função trata_TIMER0
{
conta_500++;                 //A cada 25 ms a variável conta_500 é incrementada
if (conta_500>=20)           //Se esta for >= 20, segue o código abaixo
  {
   conta_1000++;             //Incrementa a variável conta_1000
   if (conta_1000>=2)        //Se esta for >= 2, segue abaixo
     {
      PORTDbits.RD0 = ~PORTDbits.RD0;    //Inverte o estado do pino RD0
      conta_1000 = 0;                    //Zera a variável conta_1000
     }
        PORTDbits.RD1 = ~PORTDbits.RD1;  //Inverte a posição do pino RD1
        conta_500 = 0;                   //Zera a variável conta_500
  }
INTCONbits.TMR0IF = 0;                   //Zera o flag de controle da interrupção
TMR0L = 61;                              //Retoma o valor inicial para o timer 0
}


void main()
{
TRISD = 0x00;                //Faz toda a porta D como saída
PORTD = 0x00;                //Zera toda a porta D, apagando os LEDS

T0CON = 0b11000111;          /*Configura o Registrador T0CON
```

```
                        TMR0ON = 1 -> Habilita o TIMER 0
                        T08BIT = 1 -> TIMER 0 no modo 8 bits
                        T0CS = 0 -> Incremento pelo ciclo de máquina
                        ...0,5us para o cristal de 8MHz.
                        T0SE = 0 -> Incremento na orda de subida.
                        PSA = 0 -> Prescale aplicado ao Timer 0
                        PS2, PS1 e PS0 = 1 -> Prescale = 1:256*/

INTCON = 0b10100000;    /*Configura o registrador INTCON
                        GIE = 1 (bit7) -> Habilita a interrupção global
                        TMR0IE = 1 (bit 5)-> Habilita a interrupção do Timer 0
                        TMR0IF = 0 (bit 2)-> Flag de interrupção do Timer 0
                      ...desligado */

TMR0L = 61;             //Valor Inicial para o timer 0

while(1);
}
```

**Entendendo o programa:**

Primeiramente, utilizamos duas variáveis, conta_500 e conta_100, que servirão de controle para que tenhamos os tempos de 500ms e 1000ms respectivamente.
De acordo com os cálculos feitos, sabemos que acontecerá uma interrupção do Timer 0 a cada 25ms e cada vez que esta ocorrer, a variável conta_500 será incrementada até que tenhamos esta variável igual a 20, ou seja, no tempo de 500ms (25ms x 20). Neste tempo também incrementamos a variável conta_1000 até que esta seja igual a 2, portanto em 1000ms (2 x 20 x 25ms).
No tempo de 500ms também invertemos o estado do pino 0 da porta D (RD0), assim, se este estiver em 0 vai para 1 e vice-versa e no tempo de 1000ms, invertemos o estado do pino 1 da porta D (RD1).
Na função principal, são feitas as configurações dos registradores T0CON e INTCON para habilitarmos a interrupção do TIMER 0 de acordo com a nossa necessidade e logo em seguida, atribuímos o valor 61 (valor do cálculo para uma interrupção de 25ms) ao registrador TMR0L. Vale lembrar que que estamos trabalhando aqui no modo de 8 bits, portanto só utilizaremos o registrador TMR0L (Veja Diagrama de blocos do Timer 0 operando no modo 8 bits).

```
T0CON = 0b11000111;        /*Configura o Registrador T0CON
                           TMR0ON = 1 -> Habilita o TIMER 0
                           T08BIT = 1 -> TIMER 0 no modo 8 bits
                           T0CS = 0 -> Incremento pelo ciclo de máquina
                           ...0,5us para o cristal de 8MHz.
                           T0SE = 0 -> Incremento na orda de subida.
                           PSA = 0 -> Prescale aplicado ao Timer 0
                           PS2, PS1 e PS0 = 1 -> Prescale = 1:256*/

INTCON = 0b10100000;       /*Configura o registrador INTCON
                           GIE = 1 (bit7) -> Habilita a interrupção global
                           TMR0IE = 1 (bit 5)-> Habilita a interrupção do Timer 0
                           TMR0IF = 0 (bit 2)-> Flag de interrupção do Timer 0
                         ...desligado */

TMR0L = 61;                //Valor Inicial para o timer 0
```

No tratamento da interrupção, mais uma vez, como estamos trabalhando somente com uma interrupção não há necessidade de identificarmos a ocorrida, pois só teremos a interrupção do Timer 0, ou seja, somente o flag de controle TMR0IF vai pra 1 quando ocorrer o overflow na contagem do Timer 0.
É importante notar que toda vez que ocorrer uma interrupção, independente do dispositivo, o programa sempre será desviado para a função de tratamento das interrupções, cabendo ao programador identificar a interrupção ocorrida.
Note também que estamos aqui fazendo o tratamento da interrupção de alta prioridade, porém o Timer 0 permite a interrupção de baixa prioridade também. Para que isto ocorra é necessário habilitarmos o bit TMR0IP do registrador INTCON2.

```
#pragma code isr = 0x000008          //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER0       /*define a função trata_TIMER0 como rotina de
                                   ...tratamento da interrupção do Timer 0*/
void trata_TIMER0(void)              //Função trata_TIMER0
{
conta_500++;                 //A cada 25 ms a variável conta_500 é incrementada
if (conta_500>=20)           //Se esta for >= 20, segue o código abaixo
  {
   conta_1000++;             //Incrementa a variável conta_1000
   if (conta_1000>=2)        //Se esta for >= 2, segue abaixo
     {
      PORTDbits.RD0 = ~PORTDbits.RD0;     //Inverte o estado do pino RD0
      conta_1000 = 0;                     //Zera a variável conta_1000
     }
        PORTDbits.RD1 = ~PORTDbits.RD1;   //Inverte a posição do pino RD1
        conta_500 = 0;                    //Zera a variável conta_500
```

```
  }
INTCONbits.TMR0IF = 0;                  //Zera o flag de controle da interrupção
TMR0L = 61;                             //Retoma o valor inicial para o timer 0
}
```

Tratada a interrupção, o próximo passo é limpar o flag de sinalização e, como neste caso, retornarmos o valor inicial ao registrador TIMER 0. Tal medida é necessária para que tenhamos outra ocorrência do overflow no mesmo tempo que a anterior.

```
INTCONbits.TMR0IF = 0;                  //Zera o flag de controle da interrupção
TMR0L = 61;                             //Retoma o valor inicial para o timer 0
```

## Calculando o tempo total para a ocorrência da interrupção do TIMER 0 para o modo 16 bits.

O cálculo do tempo para a ocorrência da interrupção no TIMER 0 no modo 16 bits pode ser feito seguindo a fórmula:

**Tempo = 65536 x prescaler x ciclo de máquina**

**Exemplo:**
Para um cristal de 8MHz, como sabemos, teremos o ciclo de máquina igual a 0,5 us;
Utilizando o prescaler de 1:2, então:

**Tempo = 65536 x 2 x 0,5u;**

**Tempo = 65,536 ms.**

Para um prescaler de 256 e mantendo o cristal de 8MHz, teremos:

**Tempo = 65536 x 256 x 0,5u;**

**Tempo = 8388 ms ou 8,388s**

## Contando um tempo de 10 segundos no modo 16 bits

Bem, o tempo máximo que conseguimos chegar, com o máximo prescaler para o TIMER 0 no modo 16 bits, é de 8,388 segundos. Sendo assim, podemos fazer com que a interrupção ocorra a cada 5 segundos e, utilizaremos uma variável de controle para contar duas vezes a ocorrência da interrupção e assim obtermos os 10 segundos desejados.

Então, precisamos fazer com que o estouro na contagem do registrador TMR0 ocorra a cada 5s... Aqui também, ao testar o prescaler 1:128, verificamos que o tempo máximo será menor que o tempo que desejamos, então configuraremos o prescaler em 1:256.
Para que possamos encontrar o valor de início para o registrador TMR0 a fim de que o estouro na contagem deste ocorra conforme a nossa necessidade, podemos utilizar a fórmula abaixo.
Note que ela é bem parecida com a fórmula para o modo 8 bits, alterando somente o valor de 256 dos 8 bits para 65536 dos 16 bits:

$$TMR0 = 65536 - \left( \frac{\text{tempo desejado}}{\text{ciclo de máquina x prescaler}} \right)$$

Atribuindo os valores na fórmula temos:

$$TMR0 = 65536 - \left( \frac{5}{0{,}5 \text{ us x } 256} \right)$$

**TMR0 = 26474**

O resultado é um valor fracionário, mas podemos aproximá-lo para 26474. Porém, agora, estamos trabalhando no modo 16 bits e, portanto, este valor tem que ser divido entre dois registradores (TMR0L que armazenará os bits menos significativos e TMR0 que armazenará os bits mais significativos).
Para facilitar a operação de divisão entre os dois registradores, converteremos o valor 26474 para o seu equivalente em hexadecimal, assim:

**26474 = 0x676A**

Com isso, definimos que os valores para os registradores serão:

**TMR0L = 0x6A;**
e
**TMR0 = 0x67;**

**Utilizando a Interrupção do Timer 0 no modo 16 bits:**

Vamos fazer com que os leds L1, L2, L3 e L4 pisquem a cada 5 segundos e a cada, 10 segundos, os leds L5, L6, L7 e L8 pisquem também.
Como já calculado antes, os valores 0x6A e 0x67 devem ser armazenados nos registradores TMR0L e TMR0, respectivamente.
Assim, a ocorrência do overflow no TIMER 0 será a cada 5 segundos e então utilizaremos uma variável de controle para obtermos os 10 segundos que necessitaremos também.

**Circuito:** O circuito é o mesmo utilizado no exemplo anterior

**Código do programa:**

-Project Name : Timer0_16b

```
#include <p18F4520.h>

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us
#pragma config OSC = HS          //Configura o oscilador a cristal
#pragma config WDT = OFF         //Desabilita o Watchdog Timer (WDT).
#pragma config PWRT = ON         //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON        //Habilita Brown-out reset
#pragma config BORV = 1          //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF      //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF         //Desabilita o Low Voltage Program.

char conta=0;                    //Variável de controle de tempo

#pragma code int_pr = 0x000008   //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER0   /*Define a função trata_TIMER0 p/tratamento da
                                 ...interrupção do Timer 0*/
void trata_TIMER0(void)          //Função trata_TIMER0
{
conta++;                         //A cada 25 ms a variável conta é incrementada
//Inverte a situação dos pinos 0, 1, 2 e 3 da porta D
PORTDbits.RD0 = ~PORTDbits.RD0;
PORTDbits.RD1 = ~PORTDbits.RD1;
```

```
PORTDbits.RD2 = ~PORTDbits.RD2;
PORTDbits.RD3 = ~PORTDbits.RD3;
if (conta>=2)                              //Se esta for >= 2, segue o código abaixo
  {
        //Inverte a situação dos pinos 4, 5, 6 e 7 da porta D
        PORTDbits.RD4 = ~PORTDbits.RD4;
        PORTDbits.RD5 = ~PORTDbits.RD5;
        PORTDbits.RD6 = ~PORTDbits.RD6;
        PORTDbits.RD7 = ~PORTDbits.RD7;
        conta = 0;                          //Zera a variável conta
  }
INTCONbits.TMR0IF = 0;      //Zera o flag de controle da interrupção
TMR0H = 0x67;               //Valor inicial para a parte alta do TIMER 0
TMR0L = 0x6A;               //Valor Inicial para a parte baixa do TIMER 0
}

void main()
{
TRISD = 0x00;               //Faz toda a porta D como saída
PORTD = 0x0F;               //Zera toda a porta D, apagando os LEDS

T0CON = 0b10000111;         /*Configura o Registrador T0CON
                            TMR0ON = 1 -> Habilita o TIMER 0
                            T08BIT = 0 -> TIMER 0 no modo 16 bits
                            T0CS = 0 -> Incremento pelo ciclo de máquina
                            ...0,5us para o cristal de 8MHz.
                            T0SE = 0 -> Incremento na orda de subida.
                            PSA = 0 -> Prescale aplicado ao Timer 0
                            PS2, PS1 e PS0 = 1 -> Prescale = 1:256*/

INTCON = 0b10100000;        /*Configura o registrador INTCON
                            GIE = 1 (bit7) -> Habilita a interrupção global
                            TMR0IE = 1 (bit 5)-> Habilita a interrupção do Timer 0
                            TMR0IF = 0 (bit 2)-> Flag de interrupção do Timer 0
                           ...desligado */

TMR0H = 0x67;               //Valor inicial para a parte alta do TIMER 0
TMR0L = 0x6A;               //Valor Inicial para a parte baixa do TIMER 0
while(1);
}
```

**Entendendo o programa:**

O que temos aqui de diferença para o exemplo anterior é a configuração do registrador T0CON, bit 6, que agora passa a ter o valor zero, habilitando assim o modo de contagem para o registrador TIMER 0 em 16 bits.
Como para os 16 bits, precisamos de dois registradores de 8 bits, temos também o acréscimo do registrador TMR0H.

```
T0CON = 0b10000111;        /*Configura o Registrador T0CON
                           TMR0ON = 1 -> Habilita o TIMER 0
                           T08BIT = 0 -> TIMER 0 no modo 16 bits
                           T0CS = 0 -> Incremento pelo ciclo de máquina
                           ...0,5us para o cristal de 8MHz.
                           T0SE = 0 -> Incremento na orda de subida.
                           PSA = 0 -> Prescale aplicado ao Timer 0
                           PS2, PS1 e PS0 = 1 -> Prescale = 1:256*/

INTCON = 0b10100000;       /*Configura o registrador INTCON
                           GIE = 1 (bit7) -> Habilita a interrupção global
                           TMR0IE = 1 (bit 5)-> Habilita a interrupção do Timer 0
                           TMR0IF = 0 (bit 2)-> Flag de interrupção do Timer 0
                          ...desligado */

TMR0H = 0x67;              //Valor inicial para a parte alta do TIMER 0
TMR0L = 0x6A;              //Valor Inicial para a parte baixa do TIMER 0
```

Para o tratamento da interrupção, só acrescentaremos também o registrador TMR0H conforme explicado, mantendo a mesma forma de tratamento verificada no exemplo anterior para o TIMER0 em modo 8 bits.

```
#pragma code int_pr = 0x000008    //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER0    /*Define a função trata_TIMER0 p/tratamento da
                                  ...interrupção do Timer 0*/

void trata_TIMER0(void)           //Função trata_TIMER0
{
conta++;                          //A cada 25 ms a variável conta é incrementada

//Inverte a situação dos pinos 0, 1, 2 e 3 da porta D
PORTDbits.RD0 = ~PORTDbits.RD0;
PORTDbits.RD1 = ~PORTDbits.RD1;
PORTDbits.RD2 = ~PORTDbits.RD2;
PORTDbits.RD3 = ~PORTDbits.RD3;
```

```
if (conta>=2)                              //Se esta for >= 2, segue o código abaixo
  {
        //Inverte a situação dos pinos 4, 5, 6 e 7 da porta D
        PORTDbits.RD4 = ~PORTDbits.RD4;
        PORTDbits.RD5 = ~PORTDbits.RD5;
        PORTDbits.RD6 = ~PORTDbits.RD6;
        PORTDbits.RD7 = ~PORTDbits.RD7;
        conta = 0;                              //Zera a variável conta
  }
INTCONbits.TMR0IF = 0;      //Zera o flag de controle da interrupção
TMR0H = 0x67;               //Valor inicial para a parte alta do TIMER 0
TMR0L = 0x6A;               //Valor Inicial para a parte baixa do TIMER 0
}
```

No exemplo anterior, utilizamos a inversão de somente 1 bit por vez, aqui estamos invertendo os 4 bits menos significativos em 5 segundos e os 4 bits mais significativos da porta D em 10 segundos.

## Biblioteca de funções do compilador C18

O compilador C18 disponibiliza funções pré-definidas para alguns periféricos, como no caso dos Timers. Estas funções podem facilitar a escrita de um programa, tendo em vista o programador não precisar escrever diretamente nos bits de configuração de cada registrador, ou seja, basta utilizar as funções do compilador e os bits são configurados de acordo com o estabelecido.
Estas funções podem ser vistas no arquivo que acompanha a apostila, MPLAB_C18_Bibliotecas.pdf.

### Funções para o Timer 0

Primeiramente, para que possamos utilizar as funções, é necessária a inclusão do arquivo de cabeçalho '***timers.h***'.

**Função OpenTimer0**
Esta função configura e habilita o Timer 0.

**Protótipo:**
void OpenTimer0(unsigned char ***config***);

Onde ***config*** pode ser 1 ou mais configurações conforme segue:

**Interrupção do Timer 0:**
TIMER_INT_ON - Habilita a interrupção
TIMER_INT_OFF – Desabilita a interrupção

**Modo do Timer 0:**
T0_8BIT - Modo de 8 bits
T0_16BIT –Modo de 16 bits

**Fonte de clock:**
T0_SOURCE_EXT - Fonte de clock externa
T0_SOURCE_INT - Fonte de clock interna

**Trigger para o clock externo (somente para T0_SOURCE_EXT):**
T0_EDGE_FALL – Clock externo na borda de descida do sinal
T0_EDGE_RISE – Clock externo na borda de subida do sinal

**Prescaler:**
T0_PS_1_1 – prescaler 1:1
T0_PS_1_2 – prescaler 1:2
T0_PS_1_4 – prescaler 1:4
T0_PS_1_8 - prescaler 1:8

T0_PS_1_16 – prescaler 1:16
T0_PS_1_32 – prescaler 1:32
T0_PS_1_64 - prescaler 1:64
T0_PS_1_128 – prescaler 1:128
T0_PS_1_256 - prescaler 1:256

**Exemplo:**

```
OpenTimer0( TIMER_INT_OFF &          //Interrupção do Timer 0 desativada
            T0_8BIT &                //Timer 0 no modo 8 bits
            T0_SOURCE_INT &          //Fonte de clock interna
            T0_PS_1_32 );            //Prescaler de 1:32
```

Note que cada item de configuração é separado pelo sinal lógico ‘&’

**Função WriteTimer0**

Para que possamos iniciar o valor do registrador Timer 0, também temos uma função própria:

WriteTimer0(unsigned int **val**)

Onde ***val*** é o valor de início para o registrador Timer 0

**Exemplo:**

```
WriteTimer0(0x676A);      //Timer 0 iniciado com um valor na notação hexadecimal
WriteTimer0(1000);        //Timer 0 iniciado com um valor na notação decimal
```

**Programa de exemplo:**

Vamos fazer um programa onde faremos acender e apagar os LEDs da porta D a cada 1 segundo. No exemplo, utilizaremos o Timer 0 operando no modo de 8 bits, porém utilizando as funções prontas do compilador.
O circuito para este programa também é o mesmo utilizado nos exemplos anteriores.

**Código do programa:**

-Project Name : Timer0_II

```
#include <p18F4520.h>
#include <timers.h>                    //Biblioteca de funções para os Timers

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us

#pragma config OSC = HS                //Configura o oscilador a cristal
#pragma config WDT = OFF               //Desabilita o Watchdog Timer (WDT).
#pragma config PWRT = ON               //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON              //Habilita Brown-out reset
#pragma config BORV = 1                //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF            //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF               //Desabilita o Low Voltage Program.

char conta;                            //Variável de controle de tempo para o Timer 0

#pragma code int_pr = 0x000008         //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER0         /*Define a função trata_TIMER0 p/tratamento da
                                       ...interrupção do Timer 0*/
void trata_TIMER0(void)                //Função trata_TIMER0
{
conta++;                               //Incrementa a variável conta
if (conta>=40)                         //Se a variável conta >= 40 (1000 ms)
  {
        PORTD = ~PORTD;                //Inverte o estado da porta D
        conta = 0;                     //Zera a variável conta
  }
INTCONbits.TMR0IF = 0;                 //Limpa o flag da interrupção do timer 0
WriteTimer0(61);                       //Retoma o valor inicial do Timer 0
}
```

```
void main()
{
TRISD = 0x00;                    //Faz toda a porta D como saída
PORTD = 0x00;                    //Zera a porta D

OpenTimer0( TIMER_INT_ON         //Habilita a interrupção do Timer 0
            &T0_8BIT             //Modo de operação 8 bits
            &T0_SOURCE_INT       //Clock pelo ciclo de máquina
            &T0_PS_1_256);       //prescaler 1:256

INTCONbits.GIEH = 1;          /*Configura o registrador INTCON
                              ...GIE = 1 (bit7) -> Habilita a interrupção global*/

WriteTimer0(61);             //Valor Inicial para o timer 0


while(1);
}
```

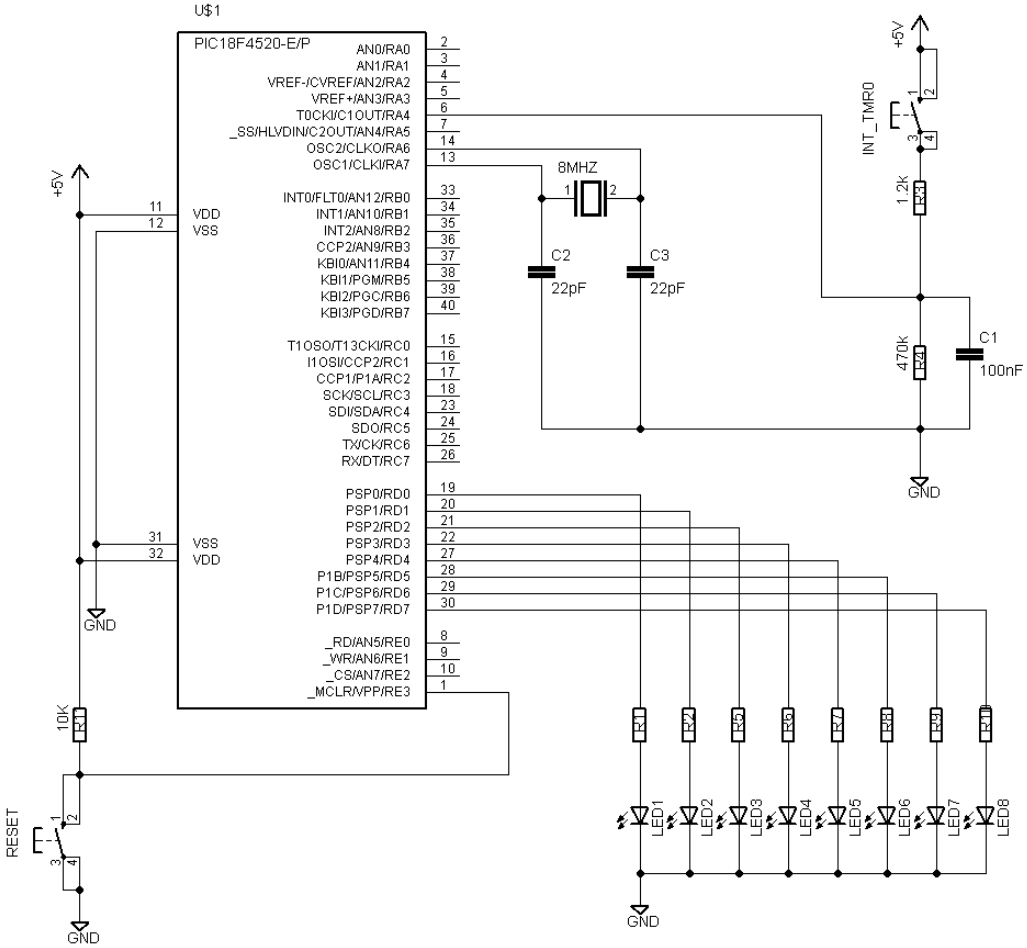
A configuração **TIMER_INT_ON** habilita somente a interrupção do Timer 0, ou seja, somente coloca em 1 o bit 5 (TMR0IE) do registrador INTCON, assim é necessário que habilitemos a interrupção global de alta e/ou baixa prioridade.
No caso de nosso exemplo, habilitamos a interrupção global de alta prioridade colocando em 1 o bit GIEH do registrador INTCON.
Quando utilizamos a função *OpenTimer0,* o compilador busca o arquivo '*t0open.c*', mostrado abaixo.
Este arquivo está localizado na pasta: ...MCC18/src/pmc_common/Timers.

## TIMER 0 com sinal externo

Como já falado anteriormente, este TIMER também pode ter uma fonte de clock externa aplicada ao pino T0CKI (RA4, pino 6) do microcontrolador.
Para isso, deve-se configurar o bit T0CS do registrador INTCON para 1, fazendo assim com que selecionemos o pino T0CKI para entrada de um sinal de clock que servirá de base para a contagem no registrador TIMER 0. Também devemos selecionar o bit T0SE para que o incremento no TIMER 0 seja pela borda de descida (1) ou pela borda de subida (0) do sinal externo.

**Exemplo:**
Vamos fazer um exemplo utilizando um sinal externo aplicado ao pino T0CKI. Veja no circuito abaixo que temos um botão (INT_TMR0) conectado a este pino e que, o sinal aplicado ao pino, sem que o botão esteja pressionado, é um sinal de nível lógico '0'. Portanto, quando o botão é pressionado, o nível lógico no pino vai para '1', pois o botão aplica o sinal VCC ao pino.
Daí, podemos tirar que para o incremento no TIMER 0, devemos selecionar a borda de subida no bit T0SE.

**Circuito:**



Para este exemplo, faremos com que os Leds conectados à porta D acendam ou apaguem (dependendo da última situação) ao pressionarmos o botão por 6 vezes.
Com isso conclui-se que é necessário desativar o prescaler, ou seja, o bit PSA deve ser setado e, como o overflow ocorre quando o contador do TIMER 0 ultrapassar 8 bits, portanto, o valor 255, então, deve-se iniciar o registrador TMR0L com o valor 250.

**Código do programa:**

-Project Name : Timer0_ext

```
#include <p18F4520.h>

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us
#pragma config OSC = HS          //Configura o oscilador a cristal
#pragma config WDT = OFF         //Desabilita o Watchdog Timer (WDT).
#pragma config PWRT = ON         //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON        //Habilita Brown-out reset
#pragma config BORV = 1          //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF      //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF         //Desabilita o Low Voltage Program.

#pragma code int_pr = 0x000008   //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER0   /*Define a função trata_TIMER0 p/tratamento da
                                 ...interrupção do Timer 0*/
void trata_TIMER0(void)          //Função trata_TIMER0
{
        PORTD = ~PORTD;          //Inverte o estado da porta D

        INTCONbits.TMR0IF = 0;   //Limpa o flag de interrupção do Timer 0
        TMR0L = 250;             //Valor Inicial para o timer 0
}

void main()
{
TRISD = 0x00;                //Faz toda a porta D como saída
PORTD = 0x00;                //Zera toda a porta D (Apaga os LEDs)

T0CON = 0b11101000;          /*Configura o Registrador T0CON
                             TMR0ON = 1 -> Habilita o TIMER 0
                             T08BIT = 1 -> TIMER 0 no modo 8 bits
                             T0CS = 1 -> Incremento pelo sinal externo
                             ...0,5us para o cristal de 8MHz.
                             T0SE = 0 -> Incremento na borda de subida.
                             PSA = 1 -> Prescaler habilitado
```

```
                        PS2, PS1 e PS0 = 0*/

INTCON = 0b10100000;    /*Configura o registrador INTCON
                        GIE = 1 (bit7) -> Habilita a interrupção global
                        TMR0IE = 1 (bit 5)-> Habilita a interrupção do Timer 0
                        TMR0IF = 0 (bit 2)-> Flag de interrupção do Timer 0
                      ...desligado */

TMR0L = 250;            //Valor Inicial para o timer 0

while(1);
}
```

**Funcionamento do projeto:**

Se pressionarmos uma vez o botão conectado ao pino RA4/T0CKI, veremos que nada acontece, portanto, se pressionarmos 6 vezes este botão, os LED's acenderão e depois mais 6 vezes, os LED's apagarão e assim sucessivamente.
Analisando o código, verificamos que definimos o *prescaler* em 1:1, ou seja, neste caso, estamos colocando o bit PSA do registrador T0CON em '1', assim cada vez que pressionamos o botão, haverá, no pino RA4/T0CKI, a transição do sinal de nível '0' para nível '1', fazendo assim com que haja um incremento no registrador TIMER0.
Como iniciamos o registrador TIMER0 em 250 e havendo 6 incrementos neste, haverá o overflow na contagem do Timer, uma vez que definimos também que o Timer trabalhará no modo 8 bits. A cada overflow, ocorrerá a interrupção e, no seu tratamento, faremos a inversão do estado dos pinos da porta D, assim acendendo ou apagando os leds conectados a estes pinos.
Se modificarmos o *prescaler* para 1:2, consequentemente o bit PSA do registrador T0CON deverá ser alterado para para '0' e o incremento no registrador TIMER0 se dará a cada 2 transições, ou seja, a cada 2 vezes que pressionarmos o botão INT_TIMER0.

Veja o mesmo exemplo anterior, porém utilizando a biblioteca de funções para os Timers.

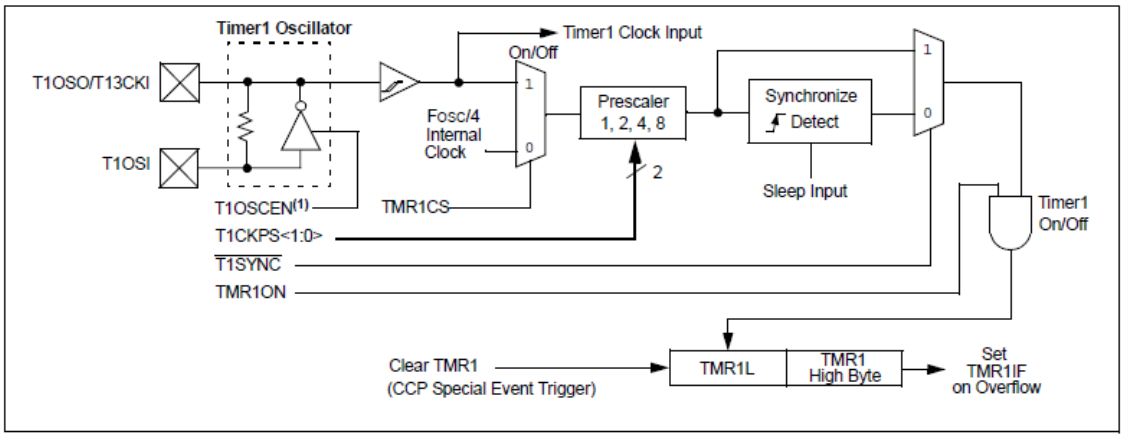
**Código do programa:** -Project Name : Timer0_ext_II

```
#include <p18F4520.h>
#include <timers.h>                    //Inclui a biblioteca de função para os timers

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us
#pragma config OSC = HS                //Configura o oscilador a cristal
#pragma config WDT = OFF               //Desabilita o Watchdog Timer (WDT).
#pragma config PWRT = ON               //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON              //Habilita Brown-out reset
#pragma config BORV = 1                //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF            //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF               //Desabilita o Low Voltage Program.

#pragma code int_pr = 0x000008         //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER0         /*Define a função trata_TIMER0 p/tratamento da
                                       ...interrupção do Timer 0*/
void trata_TIMER0(void)                //Função trata_TIMER0
{
        PORTD = ~PORTD;                //Inverte o estado da porta D
        INTCONbits.TMR0IF = 0;         //Limpa o flag de interrupção do Timer 0
        WriteTimer0(250);              //Retorna o valor inicial ao registrador Timer 0
}

void main()
{
TRISD = 0x00;                          //Faz toda a porta D como saída
PORTD = 0x00;                          //Zera toda a porta D (Apaga os LEDs)
OpenTimer0(  TIMER_INT_ON              //Habilita a interrupção do Timer 0
             &T0_8BIT                  //Habilita o modo de operação em 8 bits
             &T0_SOURCE_EXT            //Fonte de clock externa
             &T0_EDGE_RISE             //Incremento na borda de subida
             &T0_PS_1_1);              //Prescaler de 1:1 ou PSA = 1

INTCONbits.GIEH = 1;           /*Configura o registrador INTCON
                               ...GIE = 1 (bit7) -> Habilita a interrupção global*/
WriteTimer0(250);              //Valor Inicial para o timer 0

while(1);
}
```

# TIMER 1

O TIMER 1 tem suas características muito parecidas com as do Timer 0, sendo que ele também pode operar como contador (de um evento externo através do pino T13CKI – RC0) ou temporizador (obtido pelo clock interno do PIC), mas agora, somente com 16 bits.
Desta forma, podemos concluir que um ciclo de contagem do Timer 1 se incia em 0 e termina em 65535 (equivalente aos 16 bits do contador com prescaler de 1:1).

Diagrama de blocos do Timer 1:




O TIMER 1 é controlado através do registrador T1CON, conforme segue:

| Registrador T1CON | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Nome | RD16 | T1RUN | T1CKPS1 | T1CKPS0 | T1OSCEN | /T1SYNC | TMR1CS | TMR1ON |

**RD16 = 1:** Habilita modo de escrita/leitura em 16 bits;
**RD16 = 0:** Habilita modo de escrita/leitura em 8 bits.

**T1RUN = 1:** Oscilador do Timer 1 é a fonte de clock;
**T1RUN = 0:** Oscilador do Timer 1 não é fonte de clock.

**T1CKPS0 e T1CKPS1**, são utilizados para selecionar 1 dos 4 fatores de prescaler.

| T1CKPS1 | T1CKPS0 | Prescaler |
|---|---|---|
| 0 | 0 | 1:1 |
| 0 | 1 | 1:2 |
| 1 | 0 | 1:4 |
| 1 | 1 | 1:8 |

**T1OSCEN = 1:** Habilita oscilador externo. O pino RC0 e RC1 são configurados como entrada e não poderão ser utilizados como I/O;
**T1OSCEN = 0:** Desabilita oscilador externo. RC0 opera como entrada para sinal externo para contagem (T13CKI), RC1 opera como I/O.
Este timer possui um sistema de sincronismo do clock externo com o clock interno;

**/T1SYNC = 1:** Sincronismo desligado;
**/T1SYNC = 0:** Sincronismo ligado.

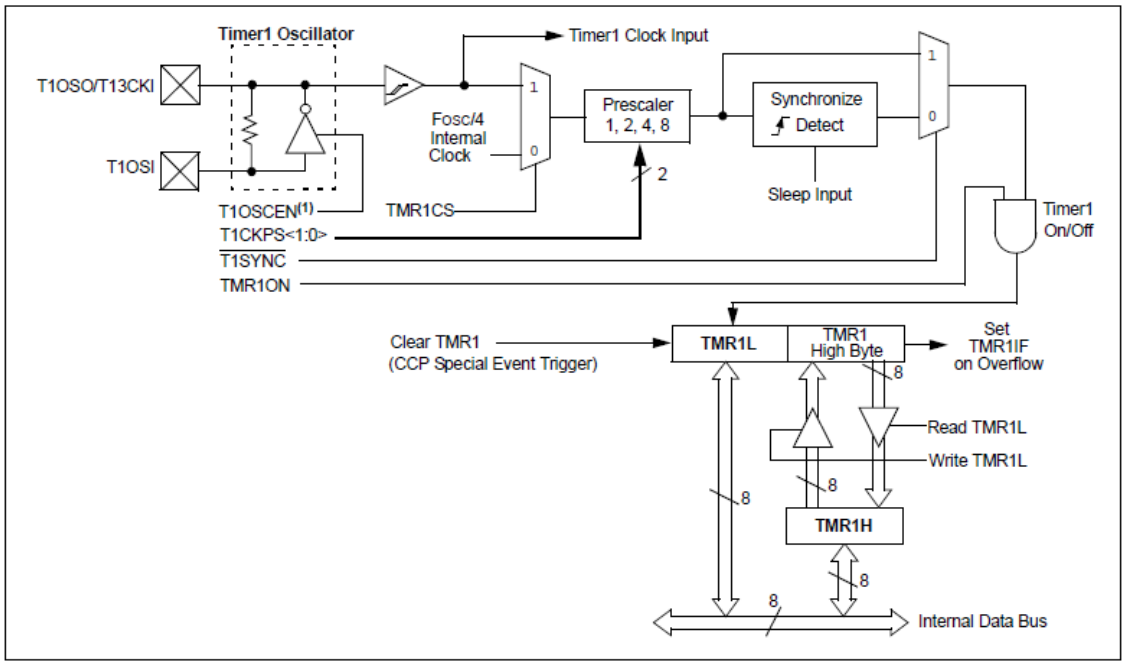
**TMR1CS = 1:** Clock externo via RC0/T13CKI (na borda de subida);
**TMR1CS = 0:** Clock interno através do ciclo de máquina.

**TMR1ON = 1:** Contagem do timer 1 habilitada;
**TMR1ON = 0:** Contagem do timer 1 desabilitada.

**Modo de escrita e leitura em 16 bits:**

O TIMER 1 pode ser configurado para escritas e leituras em seu registro em 8 ou 16 bits. Se o bit RD16 estiver em 1, o registrador TMR1H será mapeado e uma leitura no registrador TMR1L retornará o conteúdo também dos 8 bits do primeiro registrador, ou seja o retorno será de 16 bits no total (8 bits do registrador TMR1H e 8 bits do registrador TMR1L).

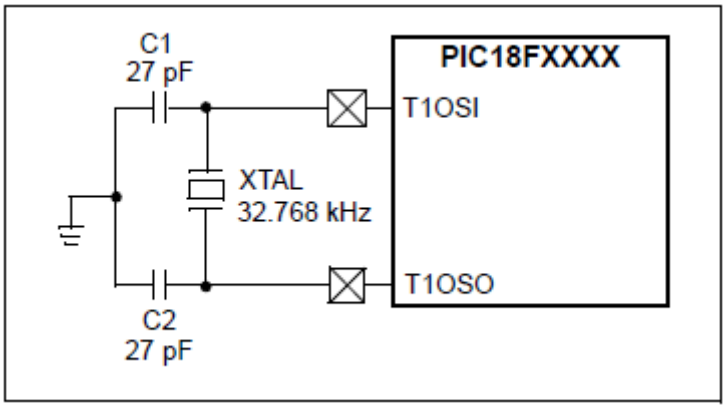
Diagrama de blocos do Timer 1 (Modo de escrita e leitura em 16 bits):



**Circuito oscilador (Low Power)**

Este TIMER também possui um circuito oscilador (low-power) para a inclusão de cristal de 32,768 KHz entre os pinos T1OSI e T1OSO. Isto faz com que o sistema proporcione as funcionalidade de um Relógio de Tempo Real (Real Time Clock – RTC) para as aplicações com uma quantidade mínima de componentes externos.

Componentes externos para o circuito oscilador do Timer 1:




**Interrupção do Timer 1**

A interrupção do timer 1 ocorre toda vez que acontece um estouro na contagem dos registradores TMR1L e TMR1H que compõem a contagem total para o Timer 1. Este Timer, como vimos anteriormente é capaz de contar de 0 à 65535 (16 bits) e após este valor, ou seja na próxima contagem depois do 65535, os valores dos contadores são zerados e o flag TMR1IF no registrador PIR1 é setado (levado para 1) informando a ocorrência do estouro na contagem e caso a interrupção esteja habilitada (Bit TMR1IE do registrador PIE1 em 1), ocorrerá, então a interrupção.

Para que a interrupção esteja habilitada corretamente, também são necessários que os bits **GIE** e **PEIE** do registrador **INTCON**, interrupção global e interrupção de periférico, respectivamente, estejam setados.

**Registrador PIE1**

O registrador **PIE1** é o responsável por habilitar a interrupção dos periféricos individualmente, sendo que, a habilitação da interrupção de cada periférico se dará juntamente com a habilitação do bit 6 (PEIE/GIEL- Habilita interrupção de periféricos) e do bit 7 (GIE/GIEH – habilita interrupção global) do registrador INTCON.

| **Registrador PIE1** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Nome | PSPIE | ADIE | RCIE | RXIE | SSPIE | CCP1IE | TMR2IE | TMR1IE |

**TMR1IE = 1**: Habilita a interrupção do timer 1;
**TMR1IE = 0**: Desabilita a interrupção do timer 1.

**Registrador PIR1**

No Registrador **PIR1**, estão todos os flags de ocorrência da interrupção dos periféricos.

| Registrador PIR1 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Nome | PSPIF | ADIF | RCIF | RXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF |

**TMR1IF = 1:** Ocorreu estouro (overflow) na contagem do Timer 1;
**TMR1IF = 0:** Não ocorreu o estouro na contagem do Timer 1.

**Registradores TMR1L e TMR1H**

Os registradores **TMR1L** e **TMR1H** são os contadores, ou seja, são neles que a contagem do timer 1 ocorrerá, perceba que são 2 contadores de 8 bits cada, onde a parte menos significativa estará no registrador TMR1L e a parte mais significativa estará no registrador TMR1H.

| Registrador TMR1L |
|---|
| Contador de 8 bits (parte menos significativa) |
| **Registrador TMR1H** |
| Contador de 8 bits (parte mais significativa) |

## Calculando o tempo total para ocorrência da interrupção do TIMER 1

Bem, estamos utilizando em nossos exemplos um cristal externo de 8MHz, então podemos concluir que o tempo total para a ocorrência do estouro na contagem do TIMER1, utilizando o maior prescaler **1:8** e clock interno pelo ciclo de máquina, será:

**Tempo = (65536 – valor do contador) x (ciclo de máquina) x prescaler**

**Onde: ciclo de máquina = 0,5 us**

No caso, o valor do contador será 0 (queremos o tempo total), então:

**Tempo = (65536 – 0) x 0,5us x 8**
**Tempo = 262,144ms**

Tempo máximo com prescaler **1:1**:

**Tempo = (65636 – 0) x 0,5us x 1**
**Tempo = 32,768ms**

## Contando um tempo de 1 segundo com o Timer 1:

Vamos fazer um pequeno projeto para que os LED's conectados à porta D do microcontrolador pisquem a cada 1 segundo utilizando o timer 1.

Vimos acima que com o prescaler 1:8, conseguiremos no máximo 262,144ms, então podemos utilizar esta configuração de prescaler e ajustar os tempos em 4 contagens de 250ms cada, fazendo assim, um total de 1000ms (1 segundo).

Os cálculos serão então:

**Tempo = (65536 – TMR1) x (1/(FOSC/4) x prescaler**
**250ms = (65536 – TMR1) x 0,5us x 8**

$$TMR1 = 65536 - \left( \frac{tempo}{\left( \frac{1}{(FOSC/4)} \right) \times prescaler} \right)$$

**TMR1 = 3036**

Para que o valor 3036 seja “colocado” no registrador Timer 1, sem que utilizemos a biblioteca do C18, devemos separá-lo em 2 partes de 8 bits cada. Vamos, primeiramente, modificar a notação numérica de decimal para hexadecimal ou binária para facilitar.

Teremos então:

**Hexadecimal** = 0X0BDC
**Binário** = 0b0000101111011100

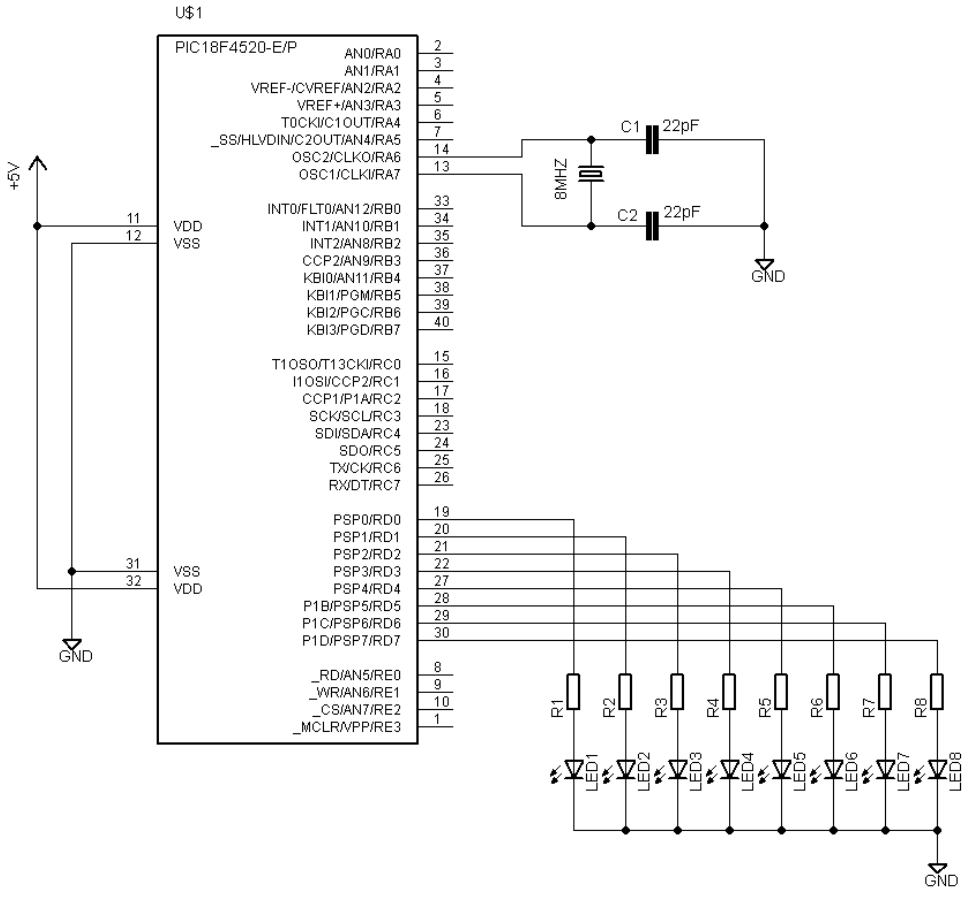
Separando em duas partes de 8 bits

**Hexadecimal** = 0X0B e 0XDC
**Binário** = 0b00001011 e 0b11011100

Iniciando os registradores de contagem do Timer 1:

**TMR1L** = 0XDC ou 0b11011100
**TMR1H** = 0X0B ou 0b00001011

**Circuito:**



**Definições do Projeto:**

-Project Name : Timer1

```
#include <p18F4520.h>

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us
#pragma config OSC = HS          //Configura o oscilador a cristal
#pragma config WDT = OFF         //Desabilita o Watchdog Timer (WDT).
#pragma config PWRT = ON         //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON        //Habilita Brown-out reset
#pragma config BORV = 1          //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF      //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF         //Desabilita o Low Voltage Program.

char conta = 0;                  //Variável de controle para o Timer 1

#pragma code int_pr = 0x000008   //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER1   /*Define a função trata_TIMER1 p/tratamento da
                                 ...interrupção do Timer 1*/
void trata_TIMER1(void)          //Função trata_TIMER1
{
conta++;                         //Incrementa a variável conta
if(conta>=4)                     //Se conta >= 4
  {
   PORTD = ~PORTD;               //Inverte o estado da porta D
   conta=0;                      //zera a variável conta
  }
PIR1bits.TMR1IF = 0;             //Limpa o flag de interrupção do Timer 1
TMR1L = 0XDC;                    //Retoma os valores iniciais aos registradores
TMR1H = 0X0B;                    //... do timer 1
}

void main()
{
TRISD = 0x00;                    //Faz toda a porta D como saída
PORTD = 0x00;                    //Zera toda a porta D (Apaga os LEDs)

T1CON = 0b00110101;              /*RD16 = 0 -> Leitura e escrita em 8 bits
                                 T1RUN = 0
                                 T1CKPS1 e T1CKPS0 = 1 -> Prescaler 1:8
                                 T1OSCEN = 0 -> Oscilador externo desabilitado
                                 T1SYNC = 1 -> sinscronismo desligado
                                 TMR1CS = 0 -> incremento pelo ciclo de máquina
```

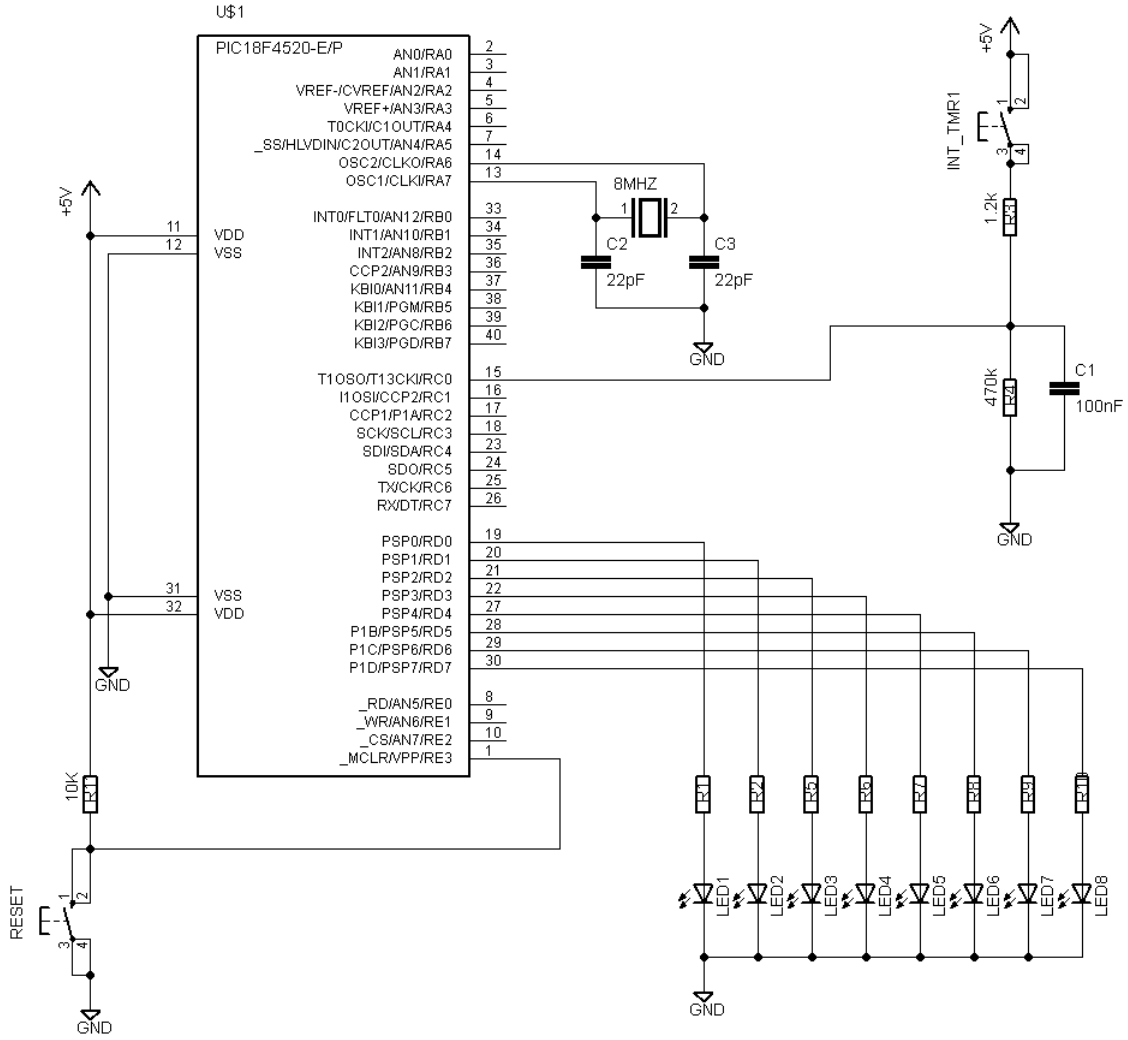
```
                                    TMR1ON = 1 -> habilita a operação do Timer 1*/

PIE1bits.TMR1IE = 1;                //Habilita a interrupção do Timer 1

INTCONbits.GIEH = 1;                /*Configura o registrador INTCON
                                    ...GIE = 1 (bit7) -> Habilita a interrupção global*/
INTCONbits.PEIE = 1;                //Habilita a interrupção de periféricos

TMR1L = 0XDC;                       //Valores iniciais para os registradores
TMR1H = 0X0B;                       //... do timer 1

while(1);
}
```

## Utilizando o Timer 1 com um sinal externo

Assim como no Timer 0, também é possível incrementar o valor do Timer 1 utilizando um sinal externo aplicado ao pino RC0/T1OS0/T13CKI do microcontrolador.

Para que o Timer 1 seja incrementado com um sinal externo, devemos colocar em 1 o bit TMR1CS do registrador T1CON.

**Circuito:**




No nosso projeto faremos com que o contador do Timer 1 seja incrementado a cada pressão no botão INT_TMR1, conectado ao pino RC0/T1OS0/T13CKI e a cada 6 vezes, o estado da porta D mudará, acendendo ou apagando os LED's conectados à ela.
Utilizaremos, neste caso, um *prescaler* de 1:1, ou seja, a cada pressão no botão ocorrerá um incremento no Timer 1. Sendo assim, iniciaremos o contador com o valor 65536 (16 bits) – 6 = 65530 ou ainda, em hexadecimal, 0XFFFA.

**Definições do Projeto:**

-Project Name : Timer1_ext

```
#include <p18F4520.h>

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us
#pragma config OSC = HS          //Configura o oscilador a cristal
#pragma config WDT = OFF         //Desabilita o Watchdog Timer (WDT).
#pragma config PWRT = ON         //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON        //Habilita Brown-out reset
#pragma config BORV = 1          //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF      //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF         //Desabilita o Low Voltage Program.

#pragma code int_pr = 0x000008   //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER1   /*Define a função trata_TIMER1 p/tratamento da
                                 ...interrupção do Timer10*/
void trata_TIMER1(void)          //Função trata_TIMER1
{
  PORTD = ~PORTD;                //Inverte o estado da porta D
  PIR1bits.TMR1IF = 0;           //Limpa o flag de interrupção do Timer 1
  TMR1L = 0XFA;                  //Valores iniciais para os registradores
  TMR1H = 0XFF;                  //... do timer 1
}

void main()
{
TRISD = 0x00;                    //Faz toda a porta D como saída
PORTD = 0x00;                    //Zera toda a porta D (Apaga os LEDs)

T1CON = 0b00000111;              /*RD16 = 0 -> Leitura e escrita em 8 bits
                                 T1RUN = 0
                                 T1CKPS1 e T1CKPS0 = 0 -> Prescaler 1:1
                                 T1OSCEN = 0 -> Oscilador externo desabilitado
                                 T1SYNC = 1 -> sinscronismo desligado
                                 TMR1CS = 1 -> incremento pelo sinal externo
                                 TMR1ON = 1 -> habilita a operação do Timer 1*/

PIE1bits.TMR1IE = 1;             //Habilita a interrupção do Timer 1

INTCONbits.GIEH = 1;             /*Configura o registrador INTCON
                                 ...GIE = 1 (bit7) -> Habilita a interrupção global*/
INTCONbits.PEIE = 1;             //Habilita a interrupção de periféricos
```

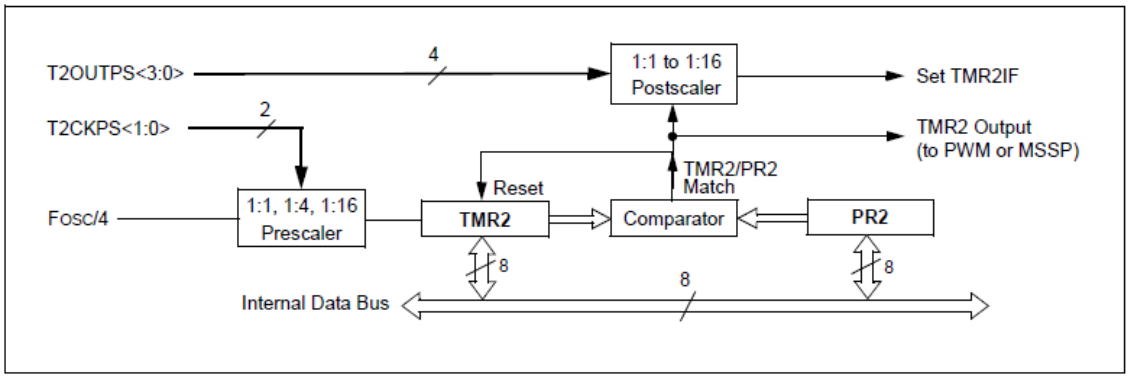
```
TMR1L = 0XFA;                    //Valores iniciais para os registradores
TMR1H = 0XFF;                    //... do timer 1

while(1);
}
```

Perceba que em nosso cálculo, utilizamos o valor 65536 que seria exatamente o valor de estouro do Timer 1, ou seja, quando o contador chegar a 65535 e ocorrer mais um incremento, o contador voltará para 0, mas para o cálculo devemos usar esse incremento como base e considerá-lo como 65536.

# TIMER 2

O timer 2 também é um contador de 8 bits, sendo relacionado somente ao clock interno e possui um *prescaler* e um *postcaler*, a diferença deste timer é que o estouro (overflow) na contagem não se dá quando o contador passa do seu limite de contagem em 255 para 0. Quem impõe o limite de contagem agora é o registrador PR2. Então sempre que o registrador TMR2 tiver o mesmo valor que PR2, o seu valor é zerado.
Quando isto ocorre, o *postcaler* é incrementado e, somente depois do término da contagem do *postcaler,* é que uma interrupção será gerada.

Diagrama de blocos do Timer 2:




Para a configuração do Timer 2, também são utilizados vários registradores. Alguns dos registradores utilizados aqui são comuns aos utilizados pelo Timer 1, somente alterando, obviamente, os bits de controle destes.

### Registrador T2CON

Os bits de configuração para o Timer 2 estão no registrador **T2CON**, conforme segue;

| Registrador T2CON | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Nome | - | TOUTPS3 | TOUTPS2 | TOUTPS1 | TOUTPS0 | TMR2ON | T2CKPS1 | T2CKPS0 |

A seleção do *postcaler* se dá através dos bits abaixo:

| TOUTPS3 | TOUTPS2 | TOUTPS1 | TOUTPS0 | POTSCALER |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1:1 |
| 0 | 0 | 0 | 1 | 1:2 |
| 0 | 0 | 1 | 0 | 1:3 |

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1:4 |
| 0 | 1 | 0 | 0 | 1:5 |
| 0 | 1 | 0 | 1 | 1:6 |
| 0 | 1 | 1 | 0 | 1:7 |
| 0 | 1 | 1 | 1 | 1:8 |
| 1 | 0 | 0 | 0 | 1:9 |
| 1 | 0 | 0 | 1 | 1:10 |
| 1 | 0 | 1 | 0 | 1:11 |
| 1 | 0 | 1 | 1 | 1:12 |
| 1 | 1 | 0 | 0 | 1:13 |
| 1 | 1 | 0 | 1 | 1:14 |
| 1 | 1 | 1 | 0 | 1:15 |
| 1 | 1 | 1 | 1 | 1:16 |

**TMR2ON = 1:** Contagem do Timer 2 habilitada;
**TMR2ON = 0:** Contagem do Timer 2 desabilitada.;

Abaixo verificamos a seleção do *prescaler:*

| T2CKPS1 | T2CKPS0 | PRESCALER |
|---|---|---|
| 0 | 0 | 1:1 |
| 0 | 1 | 1:4 |
| 1 | 0 | 1:16 |
| 1 | 1 | 1:16 |

*Obs.: As duas últimas opções são realmente iguais.*

**Interrupção do Timer 2**

A Interrupção do Timer 2 ocorre, portanto, quando a contagem do *postcaler* estourar. Então, o flag TMR2IF no registrador PIE1 será colocado em 1 e se o bit de configuração da interrupção para este Timer (bit TMR2IE do registrador PIR1) estiver habilitado, ocorrerá a interrupção.

Não podemos esquecer que para que a interrupção ocorra e o programa seja desviado corretamente para o tratamento, é necessário habilitarmos os bits **GIE/GIEH** e **PEIE/GIEL** do registrador **INTCON**.

**Registrador PIE1**

O registrador **PIE1**, como vimos anteriormente, é o responsável por habilitar a interrupção dos periféricos individualmente, sendo que, a habilitação da interrupção de cada periférico se dará juntamente com a habilitação do bit 6 (PEIE/GIEL- Habilita interrupção de periféricos) e do bit 7 (GIE – habilita interrupção global) do registrador INTCON.

| Registrador PIE1 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Nome | PSPIE | ADIE | RCIE | RXIE | SSPIE | CCP1IE | TMR2IE | TMR1IE |

**TMR2IE = 1**: Habilita a interrupção do timer 2;
**TMR2IE = 0:** Desabilita a interrupção do timer 2.

**Registrador PIR1**

No Registrador **PIR1**, estão todos os flags de ocorrência da interrupção dos periféricos.

| Registrador PIR1 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Nome | PSPIF | ADIF | RCIF | RXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF |

**TMR2IF = 1:** Ocorreu estouro (overflow) na contagem do Timer 2;
**TMR2IF = 0:** Não ocorreu o estouro na contagem do Timer 2.

**Registrador TMR2**

O registrador **TMR2** é o contador, ou seja, nele é armazenado o valor de início e a contagem para o Timer 2.

| Registrador TMR2 |
|---|
| Contador de 8 bits |

## Calculando o tempo total de estouro do Timer 2:

Calculando o tempo total para o Timer 2 com um cristal externo de 8MHz, prescaler de 1:1 e postcaler também de 1:1 e PR2 = 255;

**Tempo = prescaler x postcaler x PR2 x (1/(FOSC/4))**
**Tempo = 1 x 1 x 255 x 0,5us**
**Tempo = 0,1275ms**

Calculando o tempo total para o Timer 2 com um cristal externo de 8MHz, prescaler de 1:16 e postcaler também de 1:16 e PR2 = 255;

**Tempo = prescaler x postcaler x PR2 x (1/(FOSC/4))**
**Tempo = 16 x 16 x 255 x 0,5us**
**Tempo = 32,64ms**

## Contando um tempo de 1 segundo com o Timer 2:

Vamos fazer um projeto para que os LED's conectados à porta D do microcontrolador pisquem a cada 1 segundo utilizando o timer 2.

Como acabamos de calcular, o tempo máximo utilizando o Timer 2 é de 32,64ms com o *prescaler* e *postcaler* em 1:16 e PR2 = 255, portanto para que possamos atingir 1 segundo podemos utilizar, como no timer 0, o arredondamento de 40 vezes a contagem de 25ms.
Então, mantendo o *prescaler* em **1:16** e calculando o tempo para um *postcaler* de:
**1:15**: Tempo total = 30,60 ms;
**1:14:** Tempo total = 28,56ms;
**1:13:** Tempo total = 26,52ms;
**1:12:** Tempo total = 24,48ms;

Com isso, verifica-se que pode ser utilizado o *postcaler* de 1:13, pois o tempo máximo é de 26,52ms, ou seja ainda é maior que os 25ms necessitados.

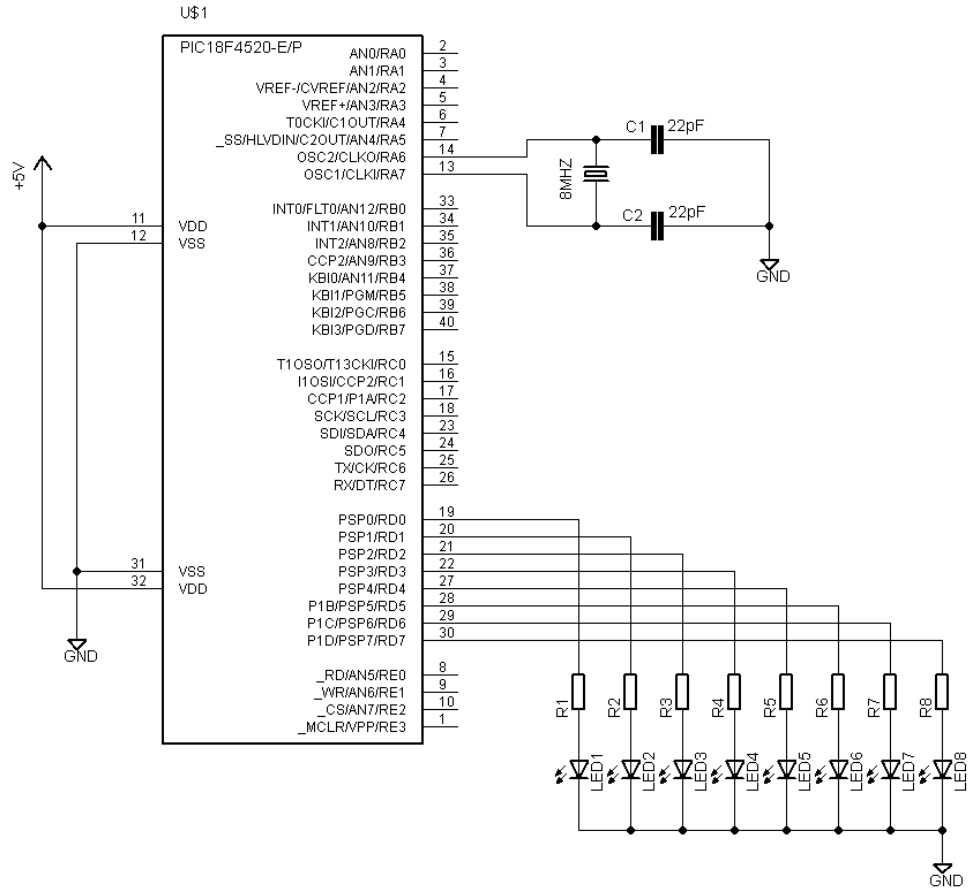
Assim como nos timers anteriores, temos uma fórmula para facilitar o cálculo do valor a ser armazenado inicialmente, agora no registrador PR2, uma vez que o valor no registrador TMR2 será mantido em 0:

$$PR2 = \left( \frac{\text{tempo desejado}}{(1/(FOSC/4)) \text{ x prescale x postcaler}} \right)$$

$$PR2 = \left( \frac{25 \text{ ms}}{0{,}5 \text{ us x } 16 \text{ x } 13} \right)$$

**PR2 = 240**

**Circuito:**



**Definições do Projeto:**

-Project Name : timer2

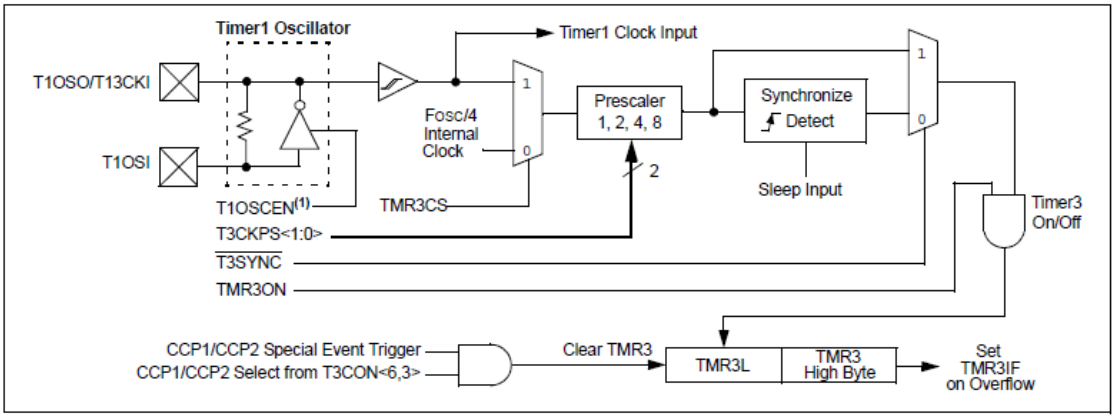
```
#include <p18F4520.h>

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us
#pragma config OSC = HS          //Configura o oscilador a cristal
#pragma config WDT = OFF         //Desabilita o Watchdog Timer (WDT).
#pragma config PWRT = ON         //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON        //Habilita Brown-out reset
#pragma config BORV = 1          //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF      //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF         //Desabilita o Low Voltage Program.

char conta = 0;                  //Variável de controle para o tempo de 1 segundo

#pragma code int_pr = 0x000008   //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER2   /*Define a função trata_TIMER2 p/tratamento da
                                 ...interrupção do Timer 2*/
void trata_TIMER2(void)          //Função trata_TIMER2
{
conta++;                         //Incrementa a variável conta
if(conta>=40)                    //Se conta >= 40
  {
   PORTD = ~PORTD;               //Inverte o estado da porta D
   conta=0;                      //Zera a variável conta
  }
PIR1bits.TMR2IF = 0;             //Limpa o flag de interrupção do Timer 2
TMR2 = 0;                        //Retorna o valor inicial ao registrador Timer 2
}

void main()
{
TRISD = 0x00;                    //Faz toda a porta D como saída
PORTD = 0x00;                    //Zera toda a porta D (Apaga os LEDs)

T2CON = 0b01100111;              /*Postcaler = 1:13
                                 TOUTPS3, TOUTPS2 = 1
                                 TOUTPS1, TOUTPS0 = 0
                                 TMR2ON = 1 -> Habilita o timer 2
                                 T2CKPS1, T2CKPS0 = 1 -> Prescaler = 1:16*/
PIE1bits.TMR2IE = 1;             //Habilita a interrupção do Timer 2
```

```
PR2 = 240;                  //Valor inicial para o registrador PR2


INTCONbits.GIEH = 1;        /*Configura o registrador INTCON
                            ...GIE = 1 (bit7) -> Habilita a interrupção global*/
INTCONbits.PEIE = 1;        //PEIE= 1(bit6) -> Habilita a interrupção de periféricos

TMR2 = 0;                   //Valor Inicial para o timer 0

while(1);
}
```

# TIMER 3

O Timer 3 também é um contador e temporizador de 16 bits e tem suas características bem próximas do Timer 1. Este Timer é composto por dois registradores (**TMR3H** e **TMR3L**) de 8 bits que podem ser lidos ou escritos e podem ser incrementados através do ciclo de máquina ou através de um sinal externo aplicado ao pino **RC0/T1OS0/T13CKI** ou ainda através de um cristal oscilador em **RC0/T1OS0/T13CKI** e **RC1/T1OS1/CCP2**, tal qual o Timer 1.

Diagrama de blocos do Timer 3:




O TIMER 3 é controlado através do registrador T3CON, conforme segue:

| Registrador T3CON | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Nome | RD16 | T3CCP2 | T3CKPS1 | T3CKPS0 | T3CCP1 | /T3SYNC | TMR3CS | TMR3ON |

**RD16 = 1:** Habilita modo de escrita/leitura em 16 bits;
**RD16 = 0:** Habilita modo de escrita/leitura em 8 bits.

**T3CCP2** e **T3CCP1** são utilizados como base de tempo para o módulo CCP (Capture/Compare/PWM)

**T3CKPS0 e T3CKPS1**, são utilizados para selecionar 1 dos 4 fatores de prescaler.

| T3CKPS1 | T3CKPS0 | Prescaler |
|---|---|---|
| 0 | 0 | 1:1 |
| 0 | 1 | 1:2 |
| 1 | 0 | 1:4 |
| 1 | 1 | 1:8 |

**/T3SYNC = 1:** Sincronismo desligado;
**/T3SYNC = 0:** Sincronismo ligado.

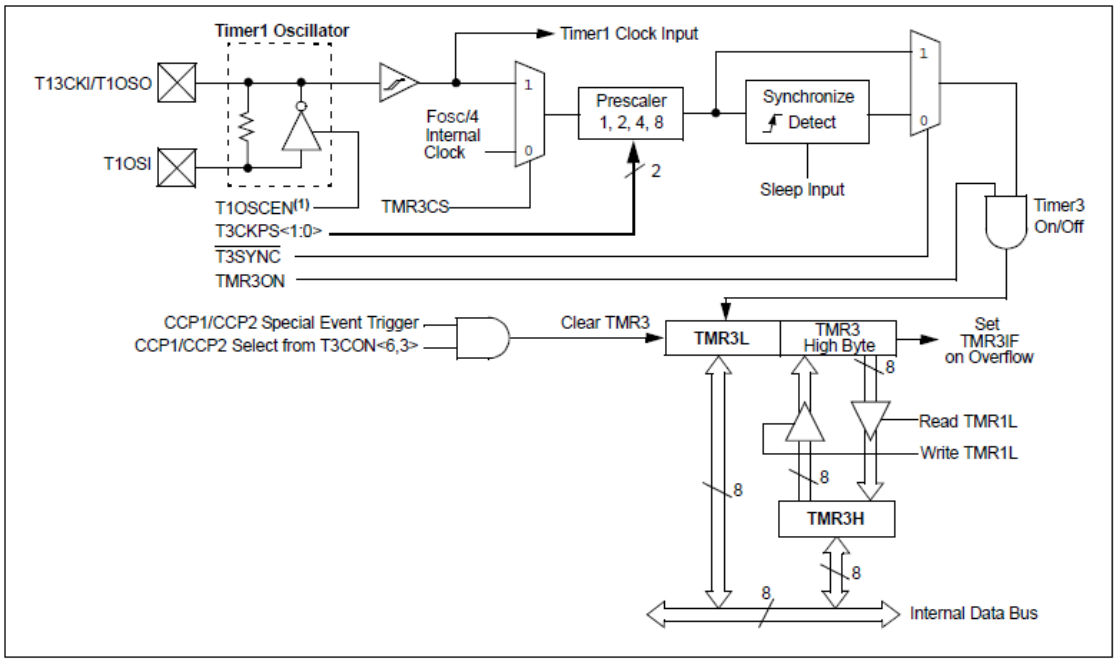
**TMR3CS = 1:** Clock externo via RC0/T13CKI (na borda de subida);
**TMR3CS = 0:** Clock interno através do ciclo de máquina.

**TMR3ON = 1:** Contagem do timer 1 habilitada;
**TMR3ON = 0:** Contagem do timer 1 desabilitada.

**Modo de escrita e leitura em 16 bits:**

O TIMER 3 também pode ser configurado para escritas e leituras em seu registro em 8 ou 16 bits. Se o bit RD16 estiver em 1, o registrador TMR3H será mapeado e uma leitura no registrador TMR3L retornará o conteúdo também dos 8 bits do primeiro registrador, ou seja o retorno será de 16 bits no total (8 bits do registrador TMR3H e 8 bits do registrador TMR3L).

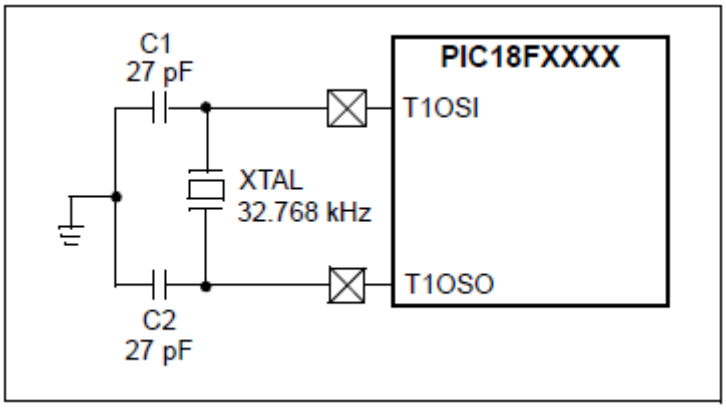
Diagrama de blocos do Timer 3 para o modo de escrita e leitura em 16 bits:



**Circuito oscilador (Low Power)**

Este TIMER, assim, como o Timer 1, também possui um circuito oscilador (low-power) para a inclusão de cristal de 32,768 KHz entre os pinos T1OSI e T1OSO. Isto faz com que o sistema proporcione as funcionalidade de um Relógio de Tempo Real (Real Time Clock – RTC). Para acionar o circuito oscilador é necessário setar o bit T1OSCEN do registrador T1CON.

Componentes externos para o circuito oscilador do Timer 1:




**Interrupção do Timer 3**

A interrupção do Timer 3 ocorre toda vez que acontece um estouro na contagem dos registradores TMR3L e TMR3H que compõem a contagem total para o Timer 3. Este Timer, assim, como o Timer 1 é capaz de contar de 0 à 65535 (16 bits) e após este valor, ou seja na próxima contagem, os valores dos contadores são zerados e o flag TMR3IF no bit 1 do registrador PIR2 é setado, informando a ocorrência do estouro na contagem e caso a interrupção esteja habilitada (Bit TMR3IE (bit 1) do registrador PIE2 em 1), ocorrerá, então a interrupção.

Para que a interrupção esteja habilitada corretamente, também são necessários que os bits **GIE/GIEH** e **PEIE/GIEL** do registrador **INTCON**, interrupção global e interrupção de periférico, respectivamente, estejam setados.

**Registrador PIE2**

O registrador **PIE2** é o responsável por habilitar a interrupção dos periféricos individualmente, sendo que, a habilitação da interrupção de cada periférico se dará juntamente com a habilitação do bit 6 (PEIE/GIEL- Habilita interrupção de periféricos) e do bit 7 (GIE/GIEH – habilita interrupção global) do registrador INTCON.

| Registrador PIE2 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Nome | OSCFIE | CMIE | - | EEIE | BCLIE | HLVDIE | TMR3IE | CCP2IE |

**TMR3IE = 1**: Habilita a interrupção do timer 3;
**TMR3IE = 0**: Desabilita a interrupção do timer 3.

**Registrador PIR2**

No Registrador **PIR2**, estão todos os flags de ocorrência da interrupção dos periféricos.

| **Registrador PIE2** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| Nome | OSCFIE | CMIF | - | EEIF | BCLIF | HLVDIF | TMR3IF | CCP2IF |

**TMR3IF = 1:** Ocorreu estouro (overflow) na contagem do Timer 1;
**TMR3IF = 0:** Não ocorreu o estouro na contagem do Timer 1.

**Registradores TMR3L e TMR3H**

Os registradores **TMR3L** e **TMR3H** são os contadores, ou seja, são neles que a contagem do timer 1 ocorrerá, perceba que são 2 contadores de 8 bits cada, onde a parte menos significativa estará no registrador TMR1L e a parte mais significativa estará no registrador TMR1H.

| **Registrador TMR3L** |
|---|
| Contador de 8 bits (parte menos significativa) |
| **Registrador TMR3H** |
| Contador de 8 bits (parte mais significativa) |

## Calculando o tempo total para ocorrência da interrupção do TIMER 3

O Cálculo do tempo total para a ocorrência do overflow para os registradores do Timer 3 é feito da mesma maneira que o cálculo do Timer 1. Assim, para um *prescaler* de **1:8**, temos:

**Tempo = (65536 – valor do contador) x (ciclo de máquina) x prescaler**

**Onde: ciclo de máquina = 0,5 us**

No caso, o valor do contador será 0 (queremos o tempo total), então:

**Tempo = (65536 – 0) x 0,5us x 8**
**Tempo = 262,144ms**

Tempo máximo com *prescaler* **1:1**:

**Tempo = (65636 – 0) x 0,5us x 1**
**Tempo = 32,768ms**

## Contando um tempo de 1 segundo com o Timer 3:

Vamos fazer o mesmo projeto que fizemos anteriormente para o Timer 1, ou seja, um pequeno projeto onde os LED's conectados à porta D do microcontrolador pisquem a cada 1 segundo utilizando, agora, o Timer 3.

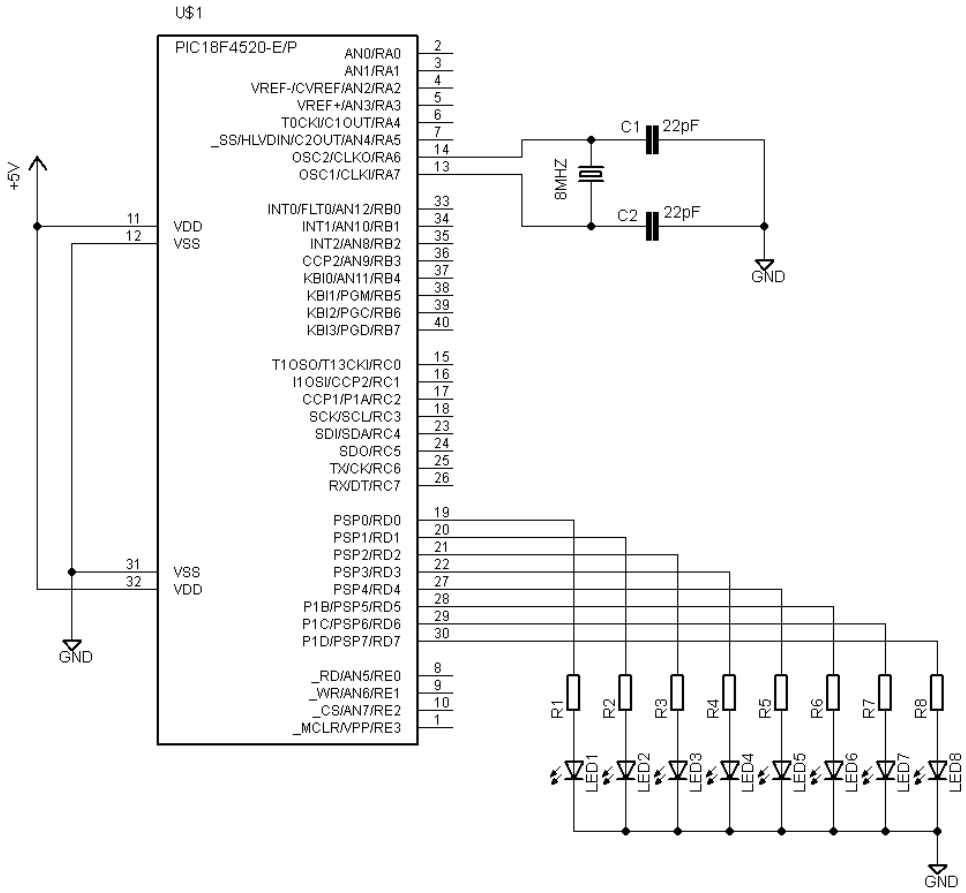
Assim como já explicado para o Timer 1, utilizaremos o *prescaler* de 1:8.

**Tempo = (65536 – TMR1) x (1/(FOSC/4) x prescaler**
**250ms = (65536 – TMR1) x 0,5us x 8**

$$TMR3 = 65536 - \left( \frac{tempo}{\left( \frac{1}{(FOSC/4)} \right) \times prescaler} \right)$$

**TMR3 = 3036**

**Circuito:**




**Definições do Projeto:**

-Project Name : Timer3

```
#include <p18F4520.h>

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us
#pragma config OSC = HS          //Configura o oscilador a cristal
#pragma config WDT = OFF         //Desabilita o Watchdog Timer (WDT).
#pragma config PWRT = ON         //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON        //Habilita Brown-out reset
#pragma config BORV = 1          //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF      //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF         //Desabilita o Low Voltage Program.
```

```
char conta = 0;                 //Variável de controle para o tempo de 1 segundo

#pragma code int_pr = 0x000008  //vetor de interrupção de alta prioridade
#pragma interrupt trata_TIMER3  /*Define a função trata_TIMER3 p/tratamento da
                                ...interrupção do Timer 3*/
void trata_TIMER3(void)         //Função trata_TIMER3

{
conta++;                        //Incrementa a variável conta
if(conta>=4)                    //Se conta >= 4
 {
  PORTD = ~PORTD;               //Inverte o estado da porta D
  conta=0;                      //Zera a variável conta
 }
PIR2bits.TMR3IF = 0;            //Limpa o flag de interrupção do Timer 3

TMR3L = 0xDC;                   //Valor Inicial para o timer 3
TMR3H = 0x0B;
}

void main()
{
TRISD = 0x00;                   //Faz toda a porta D como saída
PORTD = 0x00;                   //Zera toda a porta D (Apaga os LEDs)

T3CON = 0b00110101;             /*RD16 = 0 -> Modo de escrita/leitura em 8 bits
                                T3CCP2 = 0;
                                T3CKPS1, T3CKPS0 = 1 -> Prescaler 1:8
                                T3CCP1 = 0;
                                /T3SYNC = 1 -> Sincronismo desligado
                                TMR3CS = 0 -> Incremento pelo ciclo de máquina
                                TMR3ON = 1 -> Habilita a contagem do timer 3*/

PIE2bits.TMR3IE = 1;            //Habilita a interrupção do timer 3

INTCONbits.GIEH = 1;        /*Configura o registrador INTCON
                            GIE = 1 (bit7) -> Habilita a interrupção global*/
INTCONbits.PEIE = 1;        //PEIE= 1(bit6) -> Habilita a interrupção de periféricos

TMR3L = 0xDC;                //Valor Inicial para o timer 3
TMR3H = 0x0B;

while(1);
}
```

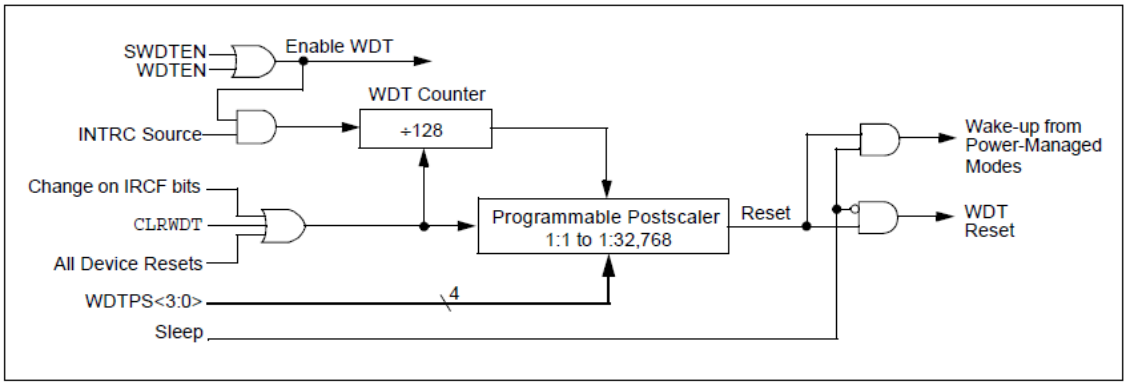
## Utilizando o Timer 3 com um sinal externo

Para que possamos utilizar o Timer 3 com um sinal externo, podemos utilizar o mesmo exemplo feito para o Timer 1, obviamente, alterando-se os registradores PIR1 e PIE1 para PIR2 e PIE2 e utilizando-se a função *OpenTimer3().*

# Watchdog Timer (WDT)

O Watchdog Timer é um temporizador cuja função é dar um reset no microcontrolador, caso seu contador não seja inicializado num determinado tempo. Essa função tem o objetivo de evitar que o processador fique parado numa determinada rotina ou que seja desviado para outro local, ou seja, evita que o programa 'trave'.
O Watchdog do PIC18F4520 é alimentado pelo clock interno e seu período mínimo é de 4ms. Este período é multiplicado por um *postcaler* programável de 16 bits. Assim podemos ter períodos de 4ms à 131.072ms ou 2 minutos e 18 segundos.

Diagrama de blocos do Watchdog Timer:




**Tempo para o watchdog de acordo com o *postcaler.***

| Postcaler | Tempo do Watchdog |
|---|---|
| 1:1 | 4ms |
| 1:2 | 8ms |
| 1:4 | 16ms |
| 1:8 | 32ms |
| 1:16 | 64ms |
| 1:32 | 128ms |
| 1:64 | 256ms |
| 1:128 | 512ms |
| 1:256 | 1024ms |
| 1:512 | 2048ms |
| 1:1024 | 4096ms |
| 1:2048 | 8192ms |
| 1:4096 | 16384ms |
| 1:8192 | 32768ms |
| 1:16384 | 65536ms |
| 1:32768 | 131072ms |

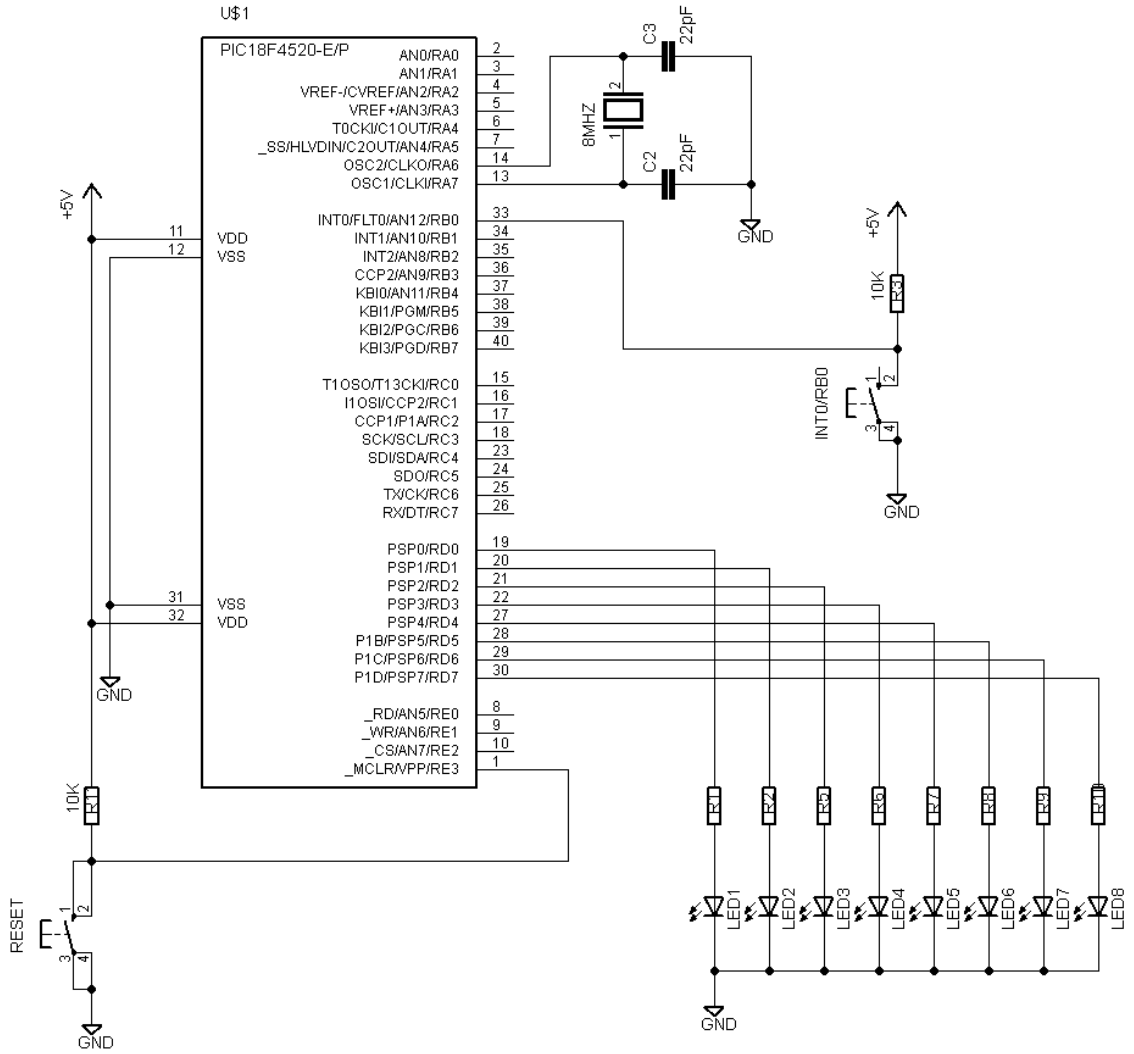
Para que possamos reiniciar o watchdog, ou seja, limpar o seu registrador, utilizaremos

a função **ClrWdt().**

## Exemplo:

Vamos fazer um programa de exemplo utilizando o watchdog. O programa deve reiniciar (ressetar) o microcontrolador se o botão INT0/RB0 não for pressionado dentro de um período de 8192ms (8,192 segundos).
Para verificarmos o ocorrido, quando pressionarmos o botão, os Led's da porta D deverão acender e permanecer acesos se o contador do Watchdog não estourar. Caso aconteça o estouro no contador, ou seja, caso o botão não seja pressionado no tempo correto, os Led' apagarão indicando que houve o reset no microcontrolador.

**Circuito:**




**Definições do Projeto:**

-Project Name : WDT

```
#include <p18F4520.h>

// Frequencia do oscilador = 8MHz
// Ciclo de máquina  = 1/(Fosc/4) = 0,5us
#pragma config OSC = HS          //Configura o oscilador a cristal
#pragma config WDT = ON          //Habilita o Watchdog Timer (WDT).
#pragma config WDTPS = 2048      //Postcaler referente ao tempo de 8192ms
#pragma config PWRT = ON         //Habilita o Power-up Timer (PWRT).
#pragma config BOREN = ON        //Habilita Brown-out reset
#pragma config BORV = 1          //Tensão do BOR é 4,33V.
#pragma config PBADEN = OFF      //RB0,1,2,3 e 4 configurado como I/O digital.
#pragma config LVP = OFF         //Desabilita o Low Voltage Program.


void main()
{
TRISBbits.TRISB0 = 1;        //Faz com que o pino 0 da porta D seja entrada
TRISD = 0x00;                //Faz toda a porta D como saída
PORTD = 0x00;                //Apaga toda a porta D (Apaga os LEDs)

while(1)
   {
    if (PORTBbits.RB0 == 0)          //Se botão INT0/RB0 for pressionado
       {
        if(PORTD == 0x00)            //Se PORTD = 0 (leds apagados)
                PORTD = 0xFF;        //Faz PORTD = 0xFF (leds acesos)
        ClrWdt();                    //Limpa o watchdog.
      }
   }
}
```