

**Projeto Assistido por Computador**  
**T7037A**  
**Prof. Franklin**  
**Projeto de um Sistema Electro-mecânico Utilizando**  
**Microcontrolador PIC**

## **1 Introdução**

Micro-controladores se tornaram o modo mais conveniente e econômico de controlar dispositivos electro-mecânicos. Neste projeto os estudantes são apresentados aos micro-controladores PIC e pretendem seguir uma seqüência de passos para o desenvolvimento e construção de vários circuitos necessário para o uso do PIC. Ao término deste projeto os estudantes terão o conhecimento para construir e programar os circuitos baseados em microcontrolador para a implementação de uma aplicação de controle de um braço robótico.

Para usar um microcontrolador em uma aplicação mecatrônica, o código a ser desenvolvido deve ser escrito, testado, e armazenado na ROM do microcontrolador. Normalmente o software é escrito e compilado em um PC e então carregado a ROM como código de máquina. Se o programa é escrito em linguagem assembler, o PC tem que ter um compilador para gerar o código de máquina para o microcontrolador. Para microcontroladores PIC a programação é realizada mediante o aplicativo MPLAB. Este software está disponível para download grátis na internet ([www.microchip.com](http://www.microchip.com)). Programar um PIC utilizando linguagem assembler pode ser no princípio muito difícil e poderiam resultar em um extenso procedimento de depuração. Felizmente compiladores de linguagem de alto nível estão disponíveis e nos permitem programar o PIC a um nível mais amigável. Neste projeto é utilizado um compilador C chamado CCS. ([www.ccsinfo.com](http://www.ccsinfo.com)). O compilador C CCS inclui uma biblioteca de programas de exemplo para muitas aplicações comuns. Cada programa de exemplo contém um cabeçalho com instruções de como rodar o exemplo, e se necessário, as instruções de instalação elétrica para conectar dispositivos externos. Na programação do PIC este arquivo compilado aparecerá como um arquivo '.hex'.

No PIC convencional o programa compilado é carregado ao CI mediante um circuito especial (firmware). Este procedimento exige colocar o microcontrolador em um local especial do circuito “gravador” para carregar o programa gerado no computador. A desvantagem deste método é que quando o programa é modificado, o microcontrolador deve ser tirado da placa da aplicação (PCB) e colocado no circuito de gravação. Muitas repetições poderiam danificar os pinos do microcontrolador assim como o PCB. Parça evitar este procedimento a funcionalidade RS232 no PIC é explorada para carregar os programas por uma porta serial. Este processo requer a instalação de um programa especial chamado *bootloader* de forma que o PIC pode ser reprogramado comunicando o circuito da aplicação pelo conector de porta serial do PC. O procedimento de *bootloading* é facilmente realizado no circuito, com o microcontrolador de PIC ainda inserido no PCB. Os passos exigidos ativar este programa estão disponíveis na internet <http://www.microchip.com/PIC16bootload/>.

Os programas do PIC normalmente são desenvolvidos em um ambiente Windows chamado MPLAB. O programa está disponível na Internet para download gratuito. O MPLAB na sua versão 6.x exige instalar um plug-in adicional para rodar o compilador CCS. O plug-in comentado pode ser obtido no endereço [www.ccsinfo.com/aboutccs.shtml](http://www.ccsinfo.com/aboutccs.shtml).

O MPLAB versão 5.70 é compatível para rodar e compilar o programa de *bootloader*. O programa de *bootloader* só é escrito para família de PIC16F87x e não pode ser usado para outros tipos de PIC.

Os passos seguintes devem ser executados:

- Programar o microcontrolador PIC no circuito da aplicação com o código de *bootloader*. Esta é uma operação que deve ser realizada com o PIC sem alimentação externa.
- Conectar o adaptador serial RS232 para seu PC. São utilizados só 3 pinos no microcontrolador PIC.
- Inserir uma linha extra em seu programa para reservar os primeiros 255 bytes em memória para o código *bootloader*. São providas instruções na parte final deste documento.
- Clique no ícone *write* no software MPLAB, e resetar o seu circuito de aplicação.
- O *bootloader* permanece ativo durante 0.2 segundos depois da reinicialização, para lhe dar a opção de transferir o novo código. Depois de outros 0.2 segundos (ou depois de completar a transferência) você pode usar a porta serial normalmente.
- Opcionalmente é possível fixar seu IDE/editor para auto-carregar o arquivo .hex compilado no PIC16F87x, usando as opções de linha de comandos no ambiente Windows.

## 2 Passos detalhados para carregar e configurar o programa *bootloader*

Antes de seguir estes passos é necessário conferir se o MPLAB esta instalado no computador e um circuito PIC circuito da aplicação está conectado ao computador.

<http://www.microchip.com/1010/pline/tools/picmicro/devenv/mplabi>

1. Carregue o programa *bootloader* do PIC16F87x disponível no endereço <http://www.microchip.com>.
2. Executar o ambiente MPLAB.
3. Selecione o programa compatível com o circuito da aplicação (por exemplo PICSTART Plus). Do menu de tarefas do MPLAB selecionar *Options/Programmer Options/Select Programmer* para definir o tipo de circuito escolhido.
4. No menu *Projeto* na barra de tarefas abrir o projeto de *bootloader* gerado no passo 1 no diretório.../Bootloader/assembly de PIC source/bootldr.pjt.
5. No menu Arquivo da barra de tarefa abrir.../Bootloader/assembly de PIC source/bootldr.asm. O que deve abrir o programa assembler do *bootloader*.
6. Confira o arquivo e entre com os seguintes valores na seção de definições de usuarios (*user setting*).
  - a. Escolher o dispositivo correto (por exemplo 16f873)
  - b. Configurar a frequência do oscilador de em 20000000 (para 20MHz como velocidade do clock)
  - c. Configurar a taxa de bauds para 19200
7. Selecionar *Project* e rodar o programa (*build*).
8. Selecione o circuito designado (por exemplo o PICSTART PLUS) e habilitar o programa. Isto abrirá algumas caixas de diálogo. Também o. arquivo .hex aparecerá em um das janelas.
9. Mude os bits de configuração conforme a seguir
  - a. Oscilador para HS
  - b. Watchdog Timer OFF
  - c. Power Up Timer OFF
  - d. Code Protect OFF
  - e. Brown Out Detect OFF
  - f. Low Voltage Program Disabled

- g. Data EE Project OFF
  - h. Flash Program Write Enabled
  
  - i. Background Debug Disabled
10. Coloque o PIC na posição correta no circuito
  11. Pressionar o botão de programar na janela de Programação do dispositivo.  
Isto carregará o programa bootloader no PIC.

Uma vez o programa de bootloader é instalado, o PIC está pronto para ser programado diretamente pelo conector serial.

### **3 Procedimento para escrever e carregar seu próprio código C CSC**

#### **(a) Compilação do Programa**

O ambiente de desenvolvimento MPLAB permite escrever um programa em código *assembler* assim como compilar o programa para produzir o arquivo *.hex* objetivado. Porém, escrevendo em linguagem *assembler* é bastante complexo precisa de um maior investimento de tempo. Há vários outros compiladores disponíveis de forma que o usuário pode programar o PIC em um idioma de alto-nível, tal como C ou C++. O projeto objetiva utilizar o compilador da CCS (<http://www.ccsinfo.com/picc.shtml>) que nos permite programar usando a linguagem C. O compilador inclui muitas funções em forma de bibliotecas para acessar o hardware de PIC. Detalhes adicionais e muitas notas ajuda pode ser achado em [www.ccsinfo.com](http://www.ccsinfo.com). Os passos mostrados abaixo são pertinentes para correr um programa de PIC que usa o compilador de CCS.

#### Passos a ser seguidos com o MPLAB 6.10

1. Usando o menu de Projeto abrir o arquivo *.hex*.
2. Selecione o dispositivo (PIC 16f873).
3. Selecione o “*active tool set*” como o compilador da CCSC e confira o diretório onde o compilador esta localizado [...\CCSC.exe].
4. Inserir o nome de projeto e local de arquivo.
5. Utilizar um programa existentes ou escreva um programa novo.
6. Abra um arquivo de texto novo do menu de arquivo e digite o programa em C. Salve este arquivo como nome do programa.c.
7. Clickar com o botão direito do mouse em “*source files*” e selecione adicionar arquivo a um programa novo.
8. No menu projeto selecionar “*build options*”. Então selecionar compilador C e selecionar também PCM (PIC compilador médio) para o dispositivo 16f873.
9. Selecionar “*build*” do menu “*project*” e compilar o programa.

### **(b) Download do programa**

N o diretório “.. /Bootloader/Downloader Windows/” e abrir o executável PICdownloader.exe. Isto abrirá uma janela e permitirá o usuário selecionar os arquivos compilados \*.hex do MPLAB. Selecione a porta como COM1 e 19200 Bd como a taxa de bauds. Clique o Write[f4] com o PIC sem alimentação externa.

### **(c) Correndo o programa PIC mediante porta serial**

A conexão serial RS232 também pode ser usada para transferir e apresentar dados da placa PCB. Dados, como leituras de sensor, podem ser transferidos ao computador em tempo real e facilmente podem ser usados para serem exibidos enquanto a aplicação PIC está rodando. Para exibir os dados um *hyperterminal* é utilizado via porta serial COM1. Para habilitar o *hyperterminal* ir para o menu *start* e selecionar *accessories/communications/hyperterminal* para estabelecer uma conexão nova, configurar o PIC como usuário do COM1. Da mesma forma configurar

1. *Bits Per Second* para 9600,
2. *Data Bits* para 8,
3. *Stopbits* para 1 e
4. Flow Control to None.

## **4 PIC16f873**

São mostradas as conexões dos pinos do PIC16f873 na Figura 1. Os manuais do PIC16F87X podem ser obtidos do endereço [www.microchip.com](http://www.microchip.com) como arquivo PDF. Quando um programa é compilado e carregado a um PIC, é armazenado como um jogo de instruções de código de máquina binárias na memória flash de programa. O arquivo .hex produzido mediante a compilação é carregado ao PIC usando um circuito de gravação ou usando o *bootloader*.

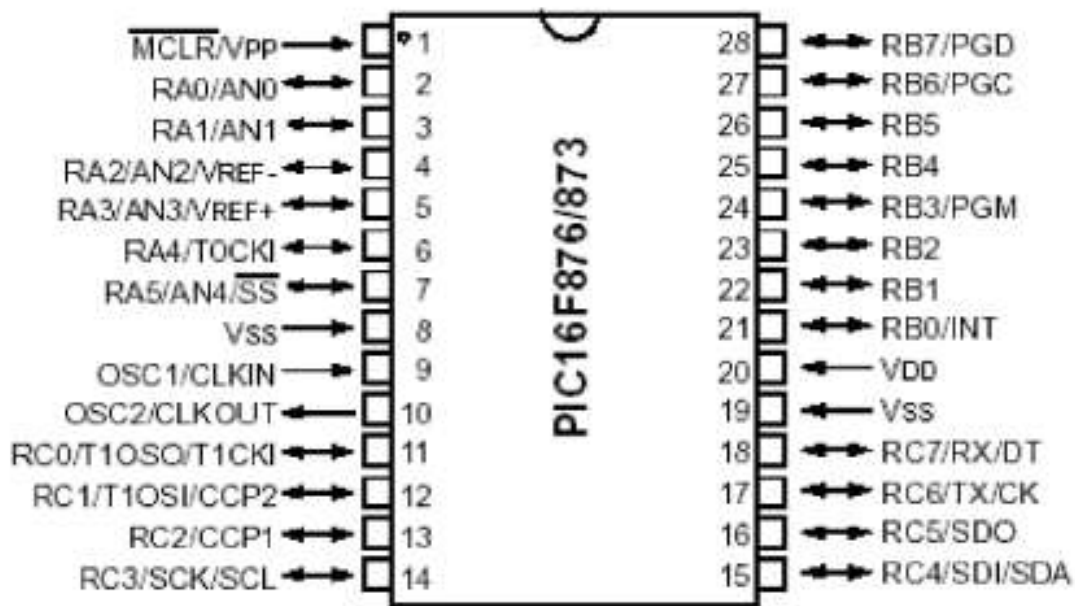


Figura 1: Pinos do PIC

O conjunto mínimo de componentes externos requeridos para o PIC funcionar são;

- a) O pino mestre de *clear* (MCLR) deve ser conectado a terra para prover a possibilidade de reinicialização. Uma vez que o PIC é reinicializado mediante este pino o PIC procura o código armazenado na EEPROM.
- b) Alimentação e terra são conectados ao PIC mediante os pinos V<sub>dd</sub> e V<sub>ss</sub>. Os índices dd e ss se referem a dreno e fonte. No PIC,  $V_{dd} = 5V$  e  $V_{ss} = 0V$ .
- c) A frequência de clock do PIC pode ser determinada mediante um circuito RC externo, ou um cristal de relógio. Neste caso nós usamos um de 4 MHz ou 20 MHz inseridos nos pinos OSC1 e OSC2.

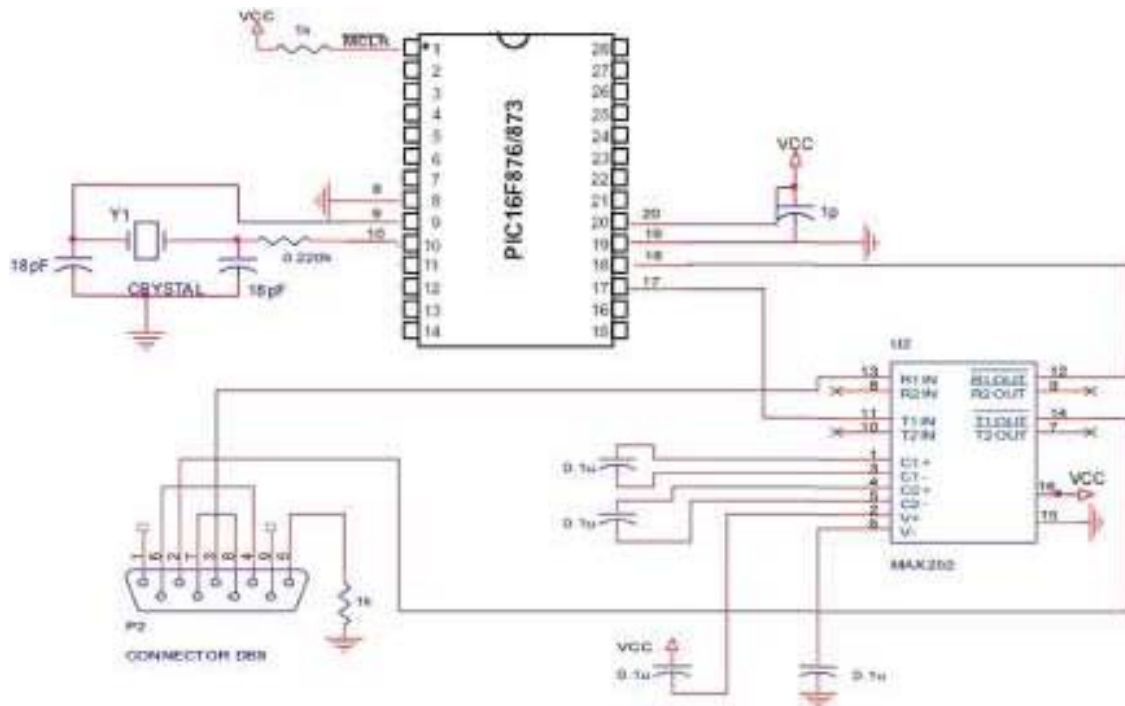


Figura 2: Circuito de PIC Básico

## 5 Programando o PIC utilizando o compilador CCS

A programação básica requer os seguintes cabeçalhos essenciais.

```
#include <16f873.h>

#fuses HS,NOWDT,NOPROTECT
#device PIC16F873 ADC=8
#use delay(clock=20000000)
//assign pin c6 and pin c7 for transmitting, and receiving
// signals for rs232 at a BAUD rate of 9600
#use rs232(BAUD=9600,XMIT=PIN C6,RCV=PIN C7)
#opt 9
//reserve top 255 bytes of memory for the bootloader
#ORG 0x0F00, 0x0FFF
void loader16F873(void) {}

main()
{
    printf("\nTYPE YOUR PROGRAM/n");
}
```

#include <16f873 . h> incluirá o arquivo de cabeçalho associado com o dispositivo PIC16f873. Em alguns casos este arquivo deverá ser copiado no diretório do programa principal para a compilação. A linha "#ORG 0x0F00 , 0x0FFF {}" reservará os 256 bytes iniciais de memória para o programa de bootloader. O 16f873 têm um total de 4k memória e os primeiros 256 endereços começam a partir de 0x0F00 e vão até 0x0FFF. Em decimal os endereços reservados são de e incluindo 3840 a 4095.

## 6 Funções de Controle

### (a) I/O Digital

Os pinos que têm a notação R podem ser utilizados como entradas ou saídas digitais. RA[0-5] configuram seis bits de I/O bidirecional (porta A). De forma semelhante, o conjunto RB[0-7] e RC[0-7] estruturam conjuntos de I/O digitais de 8 bits (porta B e porta C respectivamente). O bit menos significativo se refere à porta 0 de A, B ou C. O valor binário 1 indica que o pino é configurado como entrada e o valor 0 indica que o pino é configurado como saída. A função em CCS funciona para fixar estas portas é

```
//fixará todos os pinos A0-A5 como saídas.
set tris a(0b000000);
//fixará os pinos B[0-3] como saídas e B[4-7] como entradas.
set tris b(0b11110000);
```

**Discussão:** *Achar exemplos de aplicações onde precisamos de pinos digitais para agir como entradas.*

Para fixar o valor digital da porta, usar o comando

```
//fixar o pino A0 em 1 se A0 for saída
output_high(PIN_A0);
De forma semelhante,
    //fixe A0 em 0
    output_low(PIN_A0);
```

Os valores input/output (1 ou 0) de todos os pinos digitais podem ser mudados usando os comandos seguintes simultaneamente

```
input b(0xFF); // fixa todos os pinos de entrada de B
em 1
output b(0xFF); // fixa todos os pinos de saída de B em
1
```

### (b) entradas Analógicas:

Os pinos identificados como AN podem ser usados como entradas analógicas. Neste caso AN[0-4] (pinos 2, 3, 4, 5 & 7). O PIC16f873 tem um ADC interno (conversor analógico digital) que nos permite introduzir valores analógicos mediante a porta A. É possível ficar todos os canais A como analógicos ou podemos definir canais específicos para serem analógicos. Exemplo:

```
// Todos os pinos são analógicos
setup_adc_ports (ALL_ANALOG);

// Pinos A0, A1 e A3 são analógicos e todos os
outros são digitais. +5v é utilizado como
referência de entrada analógica.

setup_adc_ports (RA0 RA1 RA3 ANALOG);

// Pinos A0 e A1 são analógicos. Pino RA3 é
usado para a voltagem de referência e todos os
outros pinos são digitais.

    setup_adc_ports (A0_RA1_ANALOGRA3_REF);

//lendo um valor analógico
//isto fixará a porta AN2 como analógica
set_adc_channel(2);
//esta demora é um imperativo para o PIC trabalhar
delay_us(10);
// leia o valor digital do sinal analógico
value = read_adc();
```

### (c) Controle de PWM:

Os pinos relativos a Modulação por Largura de Pulso (PWM) são CCP1 e CCP2. O PIC pode ter dois drivers com controle de PWM. Também podem ser usados para ler um pulso de uma entrada. Na maioria dos casos o algoritmo PWM é utilizado para

acionamento de motores. Para ativar o controle de PWM primeiro a porta CCP deve ser configurada como saída. Na realidade quando o diagrama de pinos é analisado, os pinos CPP são RC1 e RC2.

Como um exemplo, assumir que nós queremos usar CCP1 como um PWM par acionar um motor. Neste caso é necessário fixar a porta C1 como uma saída utilizando o comando `set_tris_c()`.

Use o comando

```
setup_ccp1(CCP_PWM) // configurar CCP1 como um PWM
```

#### (d) Interrupções:

Uma interrupção é um evento que força uma chamada a uma seqüência de dados que normalmente é definida como rotina ou serviço de interrupção. Fontes típicas de interrupções no PIC incluem uma transição positiva ou negativa na entrada de RB.0/INT, uma mudança em quaisquer das entradas RB4 - RB7 ou a saturação (*overflow*) de algum contador ou timer na mudança de valor de 0xFF(255) para 0x00 (0). Na interrupção, o processador salva o endereço de retorno na pilha e o controle de programa é redirecionado à rotina de serviço de interrupção.

O compilador CCS tem várias rotinas de interrupção disponíveis. Como exemplo, #INT\_EXT é uma interrupção externa que pode ser executada por uma interrupção externa como a ocorrência de um interruptor ou mudança de estado de um pino digital. A seguir é apresentado um exemplo simples onde uma interrupção é fixada para ocorrer para uma mudança de uma entrada digital de 0 a 1.

```
//Este programa ativará uma interrupção a cada
```

```
//6 segundos para exibir "Hello from my interrupt"
```

```
#int ext
```

```
my interrupt()
```

```
{
```

```
    printf("\r\nhello from my interrupt");
```

```
main()
```

```
    // pin B0 is input and pins B[1-7] outputs
```

```
    set_tris_b(0b00000001);
```

```
    //the interrupt set to occur when the value changes
```

```
    //from low to high
```

```
    ext int edge(L TO H);
```

```
    //enables the interrupt at the given level, INT_EXT.
```

```
    enable_interrupts(GLOBAL);
```

```
    enable_interrupts(INT_EXT);
```

```

// fixe o PIC para trabalhar em um loop infinito

//Pin B1 is connected to pin B0
//when pin B1 goes to high, Pin B0 input goes high
//and calls the interrupt service routine
while(true) {
//forçar uma interrupção a cada 6 segundos
delay ms(3 000); output high(PIN B1); delay ms(3 000);
output low(PIN B1);

```

### (e) Cronômetros (*timers*):

Cronômetro é uma rotina de interrupção especialmente disponível com PIC. O PIC tem um número de cronômetros que podem ser usados para executar operações de cronometragem. As operações de cronômetro típicas são Timer\_0, Timer\_1, Timer\_2 ou RTCC. Os cronômetros consideram valores inteiros na forma set\_timer0 ou set\_RTCC (valor). Timer\_0 ou RTCC assume um valor inteiro de 8 bits valor de inteireza (max 255). Timer\_1 e Timer\_3 consideram valores de 16 bits. Todas as contas dos cronômetros são incrementais e quando um cronômetro alcança o valor de máximo deverá retornar ao seu valor inicial. Como exemplo considerar set\_timer0(0) contará 0,1,.....254,255,0,1,2...

A taxa de incremento do cronômetro define o valor de tempo correto. Como uma instrução em PIC leva quatro ciclos de relógio para ser executada, na taxa máxima a velocidade do contador será  $CLOCK/4$ . Mudando o valor do DIV, a taxa de incremento pode ser fixada como; **Counting rate**  $= (CLOCK/4)/RTCC\_DIV$

### Cronômetro de RTCC

Se a velocidade do clock é 4MHz e se fixássemos o RTCC\_DIV a 256 (valor máximo) o contador incrementará a uma taxa  $(4000000/4)/256$  vezes por segundo que é 3906.25 incrementos por segundo ou 256us por incremento.

Nós podemos usar este método para contar um período de tempo pre-especificado, 1 segundo, 2 segundos e assim por diante. Seguindo o anterior exemplo, o número total de contas requeridos para um segundo de duração é dado por,  $(1 \times 1000000/256) = 3906.25$  contas. Se fixamos o cronômetro para contar de 0, temos um total de  $3906.25/256 = 15.25$  interrupções de cronômetro. Ou nós podemos aproximar a 15 interrupções de cronômetro. Se um valor preciso é requerido, o valor de DIV poderia ser diminuído para reduzir o erro.

O programa seguinte imprimirá o tempo em segundos a todo um-segundo intervalo.

```

int g counter = 0; int
g second = 0; //interrupt
for rtcc time

```

```

#int_rtcc
void timer_rtcc ()
{
    g_counter++;

    if(g_counter ==15)

        g_second++;
        g_counter=0;
        printf("\r\n%d seconds",g_second);

main()

    setup_counters(RTCC_INTERNAL,RTCC_DIV_256)
    enable_interrupts (INT_RTCC);
    enable_interrupts (GLOBAL);

    while(true);

```

## TIMER 2

De forma semelhante também podemos usar o *Timer 2* para chamar a rotina de interrupção a cada saturação do contador. Em RTCC a interrupção é chamada a cada saturação do contador. Em Timer2 este valor pode ser variado de 1-16. O `setup_timer_2` leva três valores; `setup_timer_2(mode,period,postscale)`; Mode é o valor DIV para fixar a velocidade do contador, `2_DISABLED`, `T2_DIV_BY_1`, `T2_DIV_BY_4`, `T2_DIV_BY_16`.

**Períod** 0-255 é o valor de relógio para reajustar.

**postscale** é um número 1-16 que determina quantas reinicializações dos timers são necessárias antes de uma interrupção: (1 significa uma reinicialização, 2 significam 2, e assim por diante).

Se escolhermos um valor de DIV igual a 16 (valor máximo para o timer2). A taxa de variação do contador é determinada por,  $(4000000/4)/16=62500$ . Se nós fixamos a saturação (*overflow*) do cronômetro para acontecer a cada 256 contagens, as interrupções por segundo serão  $62500/256 \approx 244$ . Com o `postscale=1` o programa seguinte imprimirá o tempo a cada um segundo intervalo.

Se o valor de `postscale` é mudado a 2, a declaração de impressão se aparecerá a cada dois segundos de tempo.

### Cronômetro 2 exemplo

```

int g_counter = 0;
int g_second = 0;

```

```
//interrupção para cronômetro 2
#int timer2
void timer timer2 ()
{
    g_counter++;

    if(g_counter ==244)

        g_second++;
        g_counter=0;
        printf("\r\n%d seconds",g_second);

main()

    setup timer 2(T2 DIV BY 16,255,1);
    enable interrupts (INT TIMER2);
    enable_interrupts (GLOBAL);

    while(true);
```