Tratamento de interrupções com o compilador C18

Prof. Rodrigo A. Romano

1 Objetivo

- Tratamento de interrupções do microcontrolador 18FXXX em linguagem C;
- Configuração do `timer0` usando funções da biblioteca do compilador C18.

2 Funções para tratamento de interrupções

Nesta experiência a interrupção será tratada como de alta prioridade*. O primeiro passo para elaborar um software que utilize o recurso de interrupções é criar um protótipo da função de tratamento. Isto é feito introduzindo a linha de código:

```
void ISR_High_Priority(void);
```

em que `ISR_High_Priority` é o identificador da função para tratamento da interrupção. A prototipagem da função de tratamento de interrupções encontra-se na linha 5 do software `sw0.c`, listado em anexo.

Em seguida, deve-se colocar um desvio para a função de tratamento de interrupção a partir do endereço 0x08 da memória de programa. As linhas de código a seguir (linhas 9-11 de `sw0.c`) implementam tal finalidade.

```
#pragma code vec_int_high_priority = 0x08
void vec_int_high_priority(void)
{
_asm GOTO ISR_High_Priority _endasm
}
```

Por fim, é inserida a função de tratamento da interrupção propriamente dita, denominada `ISR_High_Priority`. Note que a função é precedida da diretiva `#pragma interrupt` (no caso de interrupções de baixa prioridade utiliza-se `#pragma interruptlow`) que tem a função de mostrar ao compilador que uma determinada função é usada no tratamento de interrupções.

```
#pragma code     //retorna para a seção de código padrão
#pragma interrupt ISR_High_Priority
void ISR_High_Priority(void)
{
...
}
```

*Lembre-se que os microcontroladores da família 18F pode tratar as interrupções com dois níveis de prioridade: alta e baixa, nos endereços `0x08` e `0x18`, respectivamente.

Uma função de tratamento de interrupção (ou ISR, do inglês *Interrupt Service Routine*) apresenta algumas peculiaridades: (1) uma ISR não recebe argumentos de entrada ou fornece valores; (2) as ISRs são executadas a partir da ocorrência de eventos assíncronos capazes de gerar interrupções e nunca por outras funções e (3) variáveis globais que são acessadas tanto pela ISR quanto na rotina principal deve ser declarada com o modificador `volatile`.

Nos microcontroladores da família 18F, ao menos os registradores `WREG`, `BSR` e `STATUS` são automaticamente salvos no atendimento de uma ISR, diferentemente da família 16F em que o próprio programador é responsável por salvar e restaurar todo o contexto de memória de dados. É importante lembrar que o *flag* de interrupção deve ser "resetado" por software.

3 Funções relacionadas ao periférico Timer0

Ao invés de configurar o funcionamento de um periférico através da escrita nos correspondentes registradores, em linguagem C é possível usar funções do compilador[†] para tal propósito. Por exemplo: a função:

```
OpenTimer0(TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_1);
```

habilita o flag de interrupção do `timer0` e o configura para operar com 16 bits, para usar os ciclos de máquina com fonte de *clock* e o recurso de *prescale* em 1:1.

Para escrever valores nos registradores de contagem, há a função `WriteTimer0(•)`, por exemplo:

```
WriteTimer0(60536);
```

escreve `60536` nos registradores `TMR0H` e `TMR0L`. Também é possível ler o valor desses registradores através da função `ReadTimer0()` que retorna um inteiro não sinalizado.

⇒ As funções de relacionadas aos periféricos de temporização (*timers*) estão declaradas no arquivos `timers.h`.

4 Parte Prática

1. Com o intuito de utilizar as funções apresentadas anteriormente, elabore um software que faça o segmento `g` do segmento de algum dos displays piscar com uma frequência de 2Hz. A base de tempo deve, obrigatoriamente, ser gerada pelo `timer0`. Sugestão: utilizar o software `sw0.c` (listado em anexo) como ponto de partida.

[†]Para uma listagem completa das funções usadas na operação de periféricos, consulte o documento C18 *Compiler Library*, da Microchip.

```
1   #include<p18f452.h>
2
3   #pragma config WDT = OFF, LVP = OFF, OSC = XT, PWRT = ON, BOR = ON, BORV = 42
4
5   void ISR_High_Priority(void);   //Prototipagem da função de tratamento
6                                   //de interrupções de alta prioridade
7   unsigned char valor = 0;
8
9   #pragma code vec_int_high_priority = 0x08
10  void vec_int_high_priority(void)
11  { _asm GOTO ISR_High_Priority _endasm }
12
13  #pragma code
14  #pragma interrupt ISR_High_Priority
15  void ISR_High_Priority(void)
16  {
17      //if(INTCONbits.TMR0IF)
18      //{...}
19  }
20
21  void main(void)
22  {
23      PORTB = 0b10000010;
24      TRISB = 0b00001111;
25
26      PORTD = 0xFF;
27      TRISD = 0x00;
28
29
30      while(1)    //Loop principal
31      {
32          //..
33      }
34
35  }
```