DETUA
Departamento de Electrónica e Telecomunicações
Universidade de Aveiro



Microcontroladores PIC

Guia de Trabalhos Práticos do Aluno

José Miguel Oliveira Gaspar <a20042@alunos.det.ua.pt>
Olímpia Rodrigues <a17510@alunos.det.ua.pt>

Alunos de seminário da Licenciatura em Ensino de Electrónica e Informática

Última Revisão

4 de Maio de 2006

Guia realizado sob a orientação do

Prof. Doutor José Luís Azevedo <jla@det.ua.pt>
Prof. Auxiliar do Departamento de
Electrónica e Telecomunicações da Universidade de Aveiro

Resumo

A elaboração deste guia insere-se no âmbito do seminário da Licenciatura em Ensino de Electrónica e Informática. A motivação do trabalho vêm ao encontro da lacuna documental existente sobre microcontroladores no âmbito da disciplina de Sistemas Digitais do ensino secundário, por forma minimizar a falta de recursos foram desenvolvidos dois guias de trabalhos práticos sobre microcontroladores, um para cada, aluno e professor.
Para o efeito escolheu-se um microcontrolador da família 16F87X da Microchip, para a realização dos diversos trabalhos práticos

Agradecimentos

O nosso agradecimento vai para todos aqueles que connosco colaboraram para que este guia fosse uma realidade. Em especial aos colegas da sala 317 que em muito contribuíram durante todo o seminário.

Conteúdo

Resumo iii

Conteúdo viii

Lista de Figuras x

Lista de Tabelas xi

Introdução 1

1 Enquadramento geral 1

1.1 Trabalhos práticos a realizar 1

2 Algumas características do PIC16F876 3

3 Hardware - Placa PIC 3

TP0 - Introdução ao *Assembly* e MPLAB IDE 5

1 Resumo 5

2 Objectivos 5

3 Descrição 5

3.1 Ambiente de desenvolvimento **MPLAB IDE** 5

4 Trabalho a Realizar 8

4.1 Criação de um projecto no **MPLAB IDE** 8

4.2 Tradução do código fonte em código máquina 10

4.3 Programação do PIC16F876 através da linha série 11

4.4 Estrutura base de um programa em *Assembly* 12

4.5 Análise do programa exemplo 13

TP1 - Entrada/Saída **15**

1 Resumo 15

2 Objectivos 15

3 Descrição 15

3.1 Registos TRIS e PORT 16

3.2 Inicialização dos portos 17

4 Trabalho a realizar 18

TP2 - Descodificador Hexadecimal / 7Seg, 1 display **21**

1 Resumo 21

2 Objectivos 21

3 Descrição 21

3.1 Endereçamento Indirecto, Registo FSR e INDF 21

3.2 Selecção do Banco de Memória 23

4 Trabalho a realizar 23

TP3 - Descodificador Hexadecimal / 7Seg, 3 Displays **25**

1 Resumo 25

2 Objectivos 25

3 Descrição 25

4 Trabalho a realizar 26

4.1 Descrição do registo STATUS 27

TP4 – Contador de 60 segundos, versão I **29**

1 Resumo 29

2 Objectivos 29

3 Descrição 29

3.1 Rotina Delay 30

4 Trabalho a realizar 32

TP5 – Relógio de 60 segundos, versão II **35**

1 Resumo 35

2 Objectivos 35

3 Descrição 35

3.1 Timer 0 36

3.2 Registo OPTION_REG 38

3.3 Registo INTCON 39

4 Trabalho a realizar 40

TP6 – Semáforo rodoviário(automóveis/peões) **41**

1 Resumo 41

2 Objectivos 41

3 Descrição 41

3.1 Diagrama de estados - Semáforo rodoviário 42

4 Trabalho a realizar 43

TP7 - Relógio 60 segundos, versão III **45**

1 Resumo 45

2 Objectivos 45

3 Descrição 45

3.1 *Led* pisca-pisca com botão ON/OFF por interrupção 46

4 Trabalho a realizar 48

TP8 – USART Universal Synchronous Asynchronous Receiver Transmitter **51**

1 Resumo 51

2 Objectivos 51

3 Descrição 51

3.1 Configuração dos pinos de transmissão/recepção 52

3.2 Taxa de transmissão da USART 52
3.3 Operação de transmissão 53
3.4 Operação de recepção 54
3.5 Registo TXSTA 55
3.6 Registo RCSTA 56
4 Trabalho a realizar 57

TP9 – ADC - Conversão Analógica / Digital **59**
1 Resumo 59
2 Objectivos 59
3 Descrição 59
3.1 Entrada Analógica 59
3.2 Resultado da conversão A/D 60
3.3 Mecanismo de aquisição da ADC 61
3.4 Etapas na programação da ADC (*polling*) 62
3.5 Registo ADCON0 63
3.6 Registo ADCON1 64
4 Trabalho a realizar 65

Apendix A **67**

Apendix B **69**

Lista de Figuras

1 Disposição dos componentes na placa de CI DETUA 4

2 Esquema da placa PIC DETUA 4

3 Janela associada ao processo de simulação de um programa no MPLAB IDE 7

4 Projecto em MPLAB IDE 7

5 Visualização de janelas de informação memória do PIC 8

6 fase 1 - criação de um novo projecto 8

7 fase 2 - selecção do microcontrolador. 9

8 Adicionar um ficheiro fonte existente ou criar um novo. 9

9 Tradução do código fonte em código máquina. 10

10 WinPIC Loader 11

11 Configuração de uma linha em **E/S** 16

12 Diagrama do circuito, led on/off. 18

13 Diagrama temporal da saída temporizada 18

14 Diagrama do circuito para o contador up/down 19

15 Acesso memória 21

16 Endereçamento Indirecto código de demonstração 22

17 Selecção do banco de memória 23

18 Diagrama do circuito, 1 *display* de 7 segmentos 24

19 Processo de refrescamento dos *displays* 26

20 Selecção dos bancos de memória RAM. 27

21 Diagrama de fluxo da rotina Delay 30

22 Diagrama do circuito para para visualização do valor ‘0x60’ 32

23 Diagrama do circuito, exemplifica a passagem de ‘60’ ➠ ‘59’ 33
24 Diagrama do circuito, exemplifica o término da contagem 33
25 Diagrama do circuito com sw, antes de se pressionar no botão 34
26 Diagrama do circuito com sw, após pressionar no botão 34
27 registo contador de 8 bits 35
28 Diagrama de blocos do timer 0 36
29 Semáforo rodoviário (automóveis/peões) 41
30 Diagrama de estados do trabalho prático 42
31 Diagrama do circuito, semáforo rodoviário 43
32 Exemplo de uma interrupção 46
33 Diagrama do circuito, interrupção do RBO/INT 48
34 Atribuição do porto série 52
35 Operação de transmissão assíncrona 53
36 Operacão de recepcção assíncrona 54
37 Ligação série entre o PIC e o Computador 57
38 ADC, entradas analógicas 60
39 Formato do resultado de 10 bits 60
40 Mecanismo de aquisição da ADC 61
41 Diagrama do circuito do voltímetro digital, visualização com *displays* 7 segmentos . 65
42 Diagrama do circuito do voltímetro digital, visualização terminal série 65
43 Mapa de memória RAM, registos do PIC e registos de uso geral 67

Lista de Tabelas

1 Descrição dos temas a abordar. 1

2 HEX ➟ 7 segmentos 24

3 Descrição do estado dos *leds* 42

4 Tempo minímo entre estados 43

5 Taxa de transmissão em modo assíncrono (BRGH = 1) 52

Introdução

1 Enquadramento geral

Os trabalhos práticos apresentados neste guia foram organizados de modo a que o aluno adquira, numa perspectiva evolutiva, conceitos fundamentais associados à programação de microcontroladores.

Cada um dos trabalhos dá ênfase ao desenvolvimento de uma temática específica, podendo cada um deles ser realizado independentemente um dos outros. Contudo, para um aluno sem experiência prévia que utilize a programação de microcontroladores é aconselhável que os trabalhos sejam realizados pela ordem em que são apresentados, resultando esta recomendação em dois aspectos fundamentais: a) os trabalhos estão organizados com um grau de dificuldade crescente, sendo importante que os conceitos aí abordados, sejam compreendidos, antes de se passar aos seguintes; b) o código realizado num dado trabalho possa ser reaproveitado em trabalhos posteriores.

1.1 Trabalhos práticos a realizar

Trabalho	Duração	Descrição
0	1 aula	Familiarização c/ a linguagem *assembly* e ambiente de desenvolvimento MPLAB
1	3 aulas	Entrada/Saída - Configuração dos portos do PIC
2	1 aula	Implementação de uma *lookup table*
3	2 aulas	Implementação de um sistema de visualização por multiplexagem no tempo
4	2 aulas	Contagem de tempo por contagem de instruções
5	1 aula	Contagem de tempo com recurso a *timers*
6	2 aulas	Implementação de uma máquina de estados
7	3 aulas	Programação com interrupções (*Interrupt driven I/O*)
8	2 aulas	Comunicação série assíncrona (USART)
9	4 aulas	Conversão Analógia ➠ Digital (ADC)

Tabela 1: Descrição dos temas a abordar.

A tabela 1 fornece uma curta descrição dos temas a abordar em cada trabalho prático. O primeiro trabalho prático **TP0** tem como principal objectivo a familiarização com a linguagem assembly e o ambiente de desenvolvimento MPLAB IDE. Com este trabalho pretende-se que o aluno tome conhecimento das potencialidades que ambiente de desenvolvimento disponibiliza na realização dos trabalhos

futuros; o trabalho seguinte **TP1**, continua com a familiarização do aluno com novos conceitos, desta vez a familiarização do aluno com a linguagem assembly do PIC para programação dos portos de entrada e saída na leitura e escrita de dados.

No trabalho prático 2, **TP2**, pretende-se que o aluno adquira os conhecimentos necessários de modo a compreender o acesso a variáveis na memória RAM por endereçamento indirecto. Para tal, pretende-se implementar um descodificador hexadecimal / 7 segmentos através de um único *display*.

No seguimento deste trabalho, surge o trabalho 3 (TP3), em que se pretende uma ampliação do trabalho anterior a 3 *displays* de 7 segmentos, com o objectivo de construir um programa que permita a visualização de dados nos *displays*, recorrendo ao refrescamento destes por multiplexagem no tempo.

No trabalho prático seguinte (TP4), recorre-se a contagem de tempo por contagem de instruções, usando para tal uma rotina de contagem de tempo por contagem de instruções (rotina Delay).

O trabalho prático 5 (TP5), tem como um objectivo principal a introdução à programação do *timer 0* do PIC. Com este, vai-se poder alargar a gama de contagem de tempo, algo que com a utilização da rotina *Delay* não é possível.

Com o trabalho 6 (TP6), pretende-se construir uma máquina de estados que implementa um semáforo rodoviário para peões e automóveis, com o objectivo de aprofundar os conhecimentos acerca do *timer 0*.

Nos trabalhos práticos 4 e 5 implementou-se um relógio de 60seg recorrendo á rotina *Delay* e *timer 0*, respectivamente. Ambos implementados recorrendo à transferência de informação por *polling*. Durante o trabalho 7 (TP7) pretende-se implementar novamente um relógio, recorrendo agora à transferência de informação por interrupção. Para isso é necessária a compreensão do conceito de interrupção de um programa.

Com o grau de dificuldade a crescer, os dois últimos trabalhos reflectem essa complexidade nos conceitos a adquirir. Assim, o trabalho 8 (TP8) tem como objectivo, a comunicação série entre um computador (anfitrião) e o PIC, elaborando para tal um programa que implemente um terminal série, recorrendo a rotinas para envio e recepção de caracter ou caracteres.

Por fim, o trabalho 9 (TP9) tem como objectivo principal a introdução à conversão A/D, implementando para esse fim 2 trabalhos: um voltímetro digital básico, e um sistema de medição de distâncias recorrendo a um sensor infravermelhos.

2 Algumas características do PIC16F876

O PIC16F876, fabricado com a tecnologia CMOS dispõe de:

- Processador **RISC** (*Reduce Instrution Set Computer*)
  - 35 instruções de 14 bits;
  - Frequência máxima de funcionamento - 20Mhz (frequência do cristal);
  - Cada ciclo de relógio corresponde à frequência do cristal / 4 = 5Mhz, efectuando a cada segundo **5 MIPS** (milhões de instruções por segundo);
  - Tempo de execução das instruções normais: 1 ciclo de relógio;
  - Tempo de execução das instruções de salto condicional (*decfsz, incfsz, btfss, btfsc*), quando a executada a instrução de salto: 2 ciclos de relógio;
  - Tempo de execução de instruções de salto incondicional (*goto*): 2 ciclos de relógio.

- As seguintes características da memória;
  - Memória de programa (FLASH) de 8K (*words*) de 14 bits;
  - Cada instrução é codificada numa *word* de 14 bits;
  - Memória de dados RAM de 368 bytes;
  - Memória de dados EEPROM de 256 bytes;
  - Stack de 8 níveis.

- As seguintes características de periféricos:
  - 22 linhas de entrada/saída, agrupadas em 3 portos (PORTA ➠ 6 linhas, PORTB e PORTC ➠ 8 linhas);
  - 3 *timers*, 2 de 8 bits e 1 de 16 bits;
  - Conversor analógico ➠ digital de 10 bits, com um máximo de 5 canais de entrada analógica;
  - USART (*Universal Synchronous Asynchronous Receiver Transmitter*);
  - 13 tipos de interrupções, por exemplo externa RB0/INT, TMR0 *timer* overflow.

3 Hardware - Placa PIC

Não é propósito do guião descrever o funcionamento da placa PIC, este é feito no site da disciplina. Neste pode-se encontrar todas as informações relevantes sobre o *layout* e funcionamento da placa. Contudo para que o aluno/professor se se enquadrem com o hardware apresenta-se nas páginas seguintes o esquema eléctrico do circuito bem como a disposição dos componentes na placa de circuito impresso.

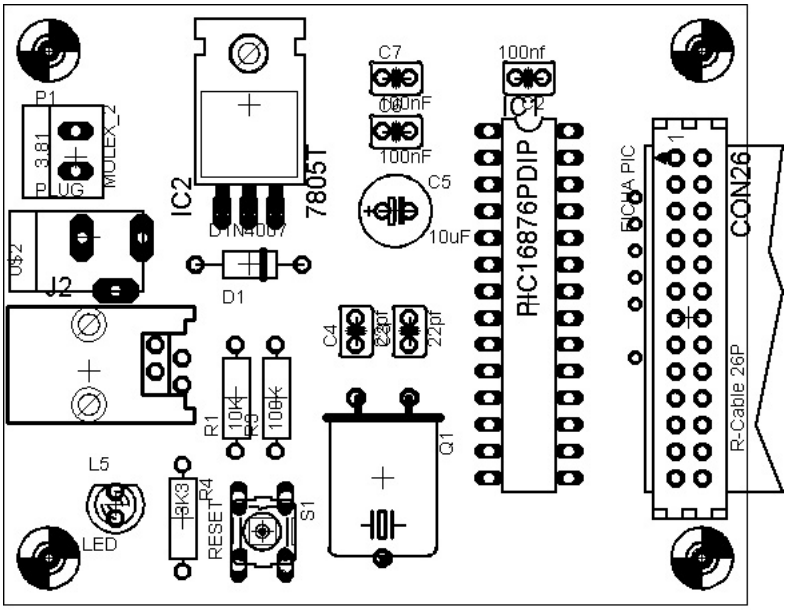


Figura 1: Disposição dos componentes na placa de CI DETUA

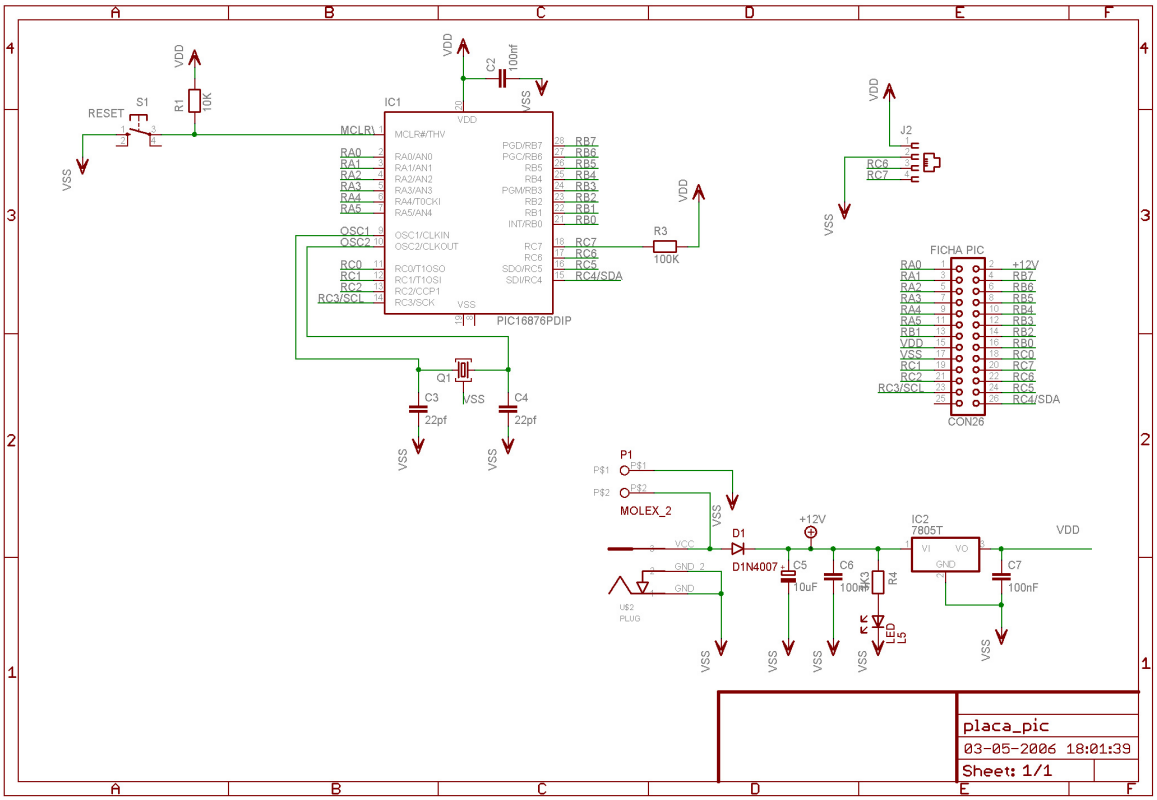


Figura 2: Esquema da placa PIC DETUA

TP0 - Introdução ao *Assembly* e MPLAB IDE



Tema a desenvolver	Duração
Familiarização com o Assemly e MPLAB IDE	1 aula

1 Resumo

Pretende-se mostrar com este trabalho os passos necessários à criação de um projecto no ambiente de desenvolvimento **MPLAB IDE** (programa de software destinado a desenvolver aplicações para micro-controladores da Microchip) e à tradução para código máquina do código fonte associado, recorrendo a um pequeno exemplo em linguagem *Assembly*. Mostra-se ainda, o processo de programação do PIC16F876 através da linha série.

2 Objectivos

- Criação de um projecto no ambiente de desenvolvimento **MPLAB IDE**.
- Conhecer o processo de tradução de código fonte para código máquina.
- Conhecer o processo de programação do microcontrolador através do **WinPIC Loader**.
- Introdução à programação em linguagem *Assembly*.

3 Descrição

3.1 Ambiente de desenvolvimento MPLAB IDE

O MPLAB IDE é um ambiente de desenvolvimento integrado que permite a edição, o *debugging* e a tradução para código máquina de programas em linguagem *Assembly*. Disponibiliza essencialmente, as seguintes ferramentas:

- Editor com reconhecimento das instruções do PIC e directivas do Assembler em *syntax high-light*.
- Visualização dos registos (memória RAM), da memória de programa e da EEPROM .
- **MPLAB SIM**, simulador de eventos, com as seguintes características:
  - ➪ Possibilidade de alteração do código fonte do programa, permitindo a sua re-execução imediata;

- Possibilidade de modificação dos valores dos registos e posições de memória do PIC em intervalos de tempo pré determinados.
- Possibilidade de modificação do valor lógico presente nas entradas do PIC.
- Simulação da evolução do programa através da criação de um cenário de estímulos externos.

No desenvolvimento inicial de um programa, é muito provável que este contenha erros de concepção que o impedem de realizar correctamente as tarefas para o qual foi projectado. Se porventura isso acontecer, é necessário voltar a analisar o código, de modo a encontrar a origem dos problemas. Em programas com alguma complexidade este processo é lento, e muitas vezes ineficiente, dado que é necessário carregar várias vezes o programa no PIC, de modo a testar o seu funcionamento. Este é um dos casos em que o simulador de software **MPLAB SIM** é útil uma vez que permite simular o programa no PC, como se este estivesse a ser executado no PIC. Outra das vantagens da utilização do simulador é a possibilidade da criação a priori de cenários de teste que servem de entrada para o simulador. A figura 3 mostra a janela associada ao processo de simulação do programa exemplo.

O ambiente de edição do projecto do programa exemplo é mostrado na figura 4. A janela localizada no canto superior esquerdo representa o *Project Manager*, que contém as referências aos ficheiros que compõem o projecto; logo abaixo desta, temos o *Memory Usage Gauge*, que mostra a informação da quantidade de memória do programa e dados usada; no lado direito temos o editor; finalmente abaixo encontra-se a janela de *output* (saída), que fornece informações sobre o estado do programa aquando da tradução de código *Assembly* para código máquina.

No MPLAB IDE, a visualização da memória (ver figura 5) é feita através das seguintes janelas:

- *Program Memory* mostra os endereços de memória o Opcode e a mnemónica correspondente do programa, alocados dentro da gama de memória disponível para o processador seleccionado. Se o PIC seleccionado suportar memória externa, e se esta estiver activa, a mesma também será visualizada.
- *File Register* mostra todos os registos do dispositivo seleccionado (corresponde á memória RAM do PIC).
- *EEPROM* mostra a memória de dados EEPROM para qualquer microcontrolador que disponha deste tipo de memória (por exemplo o PIC16F876).

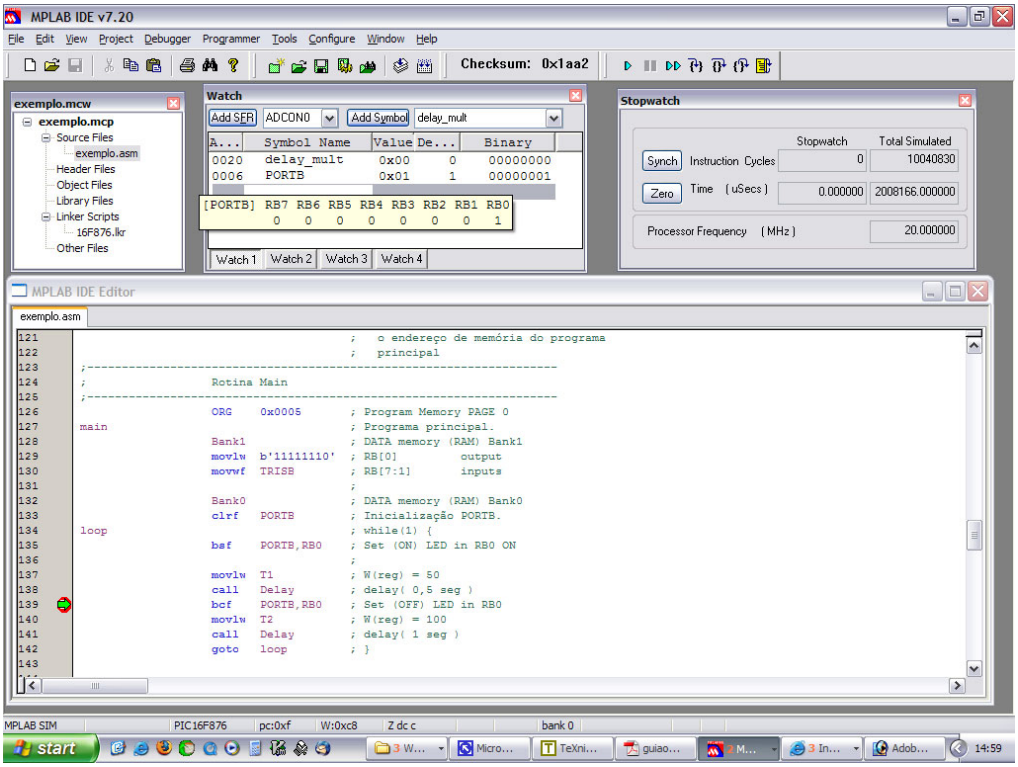


Figura 3: Janela associada ao processo de simulação de um programa no MPLAB IDE

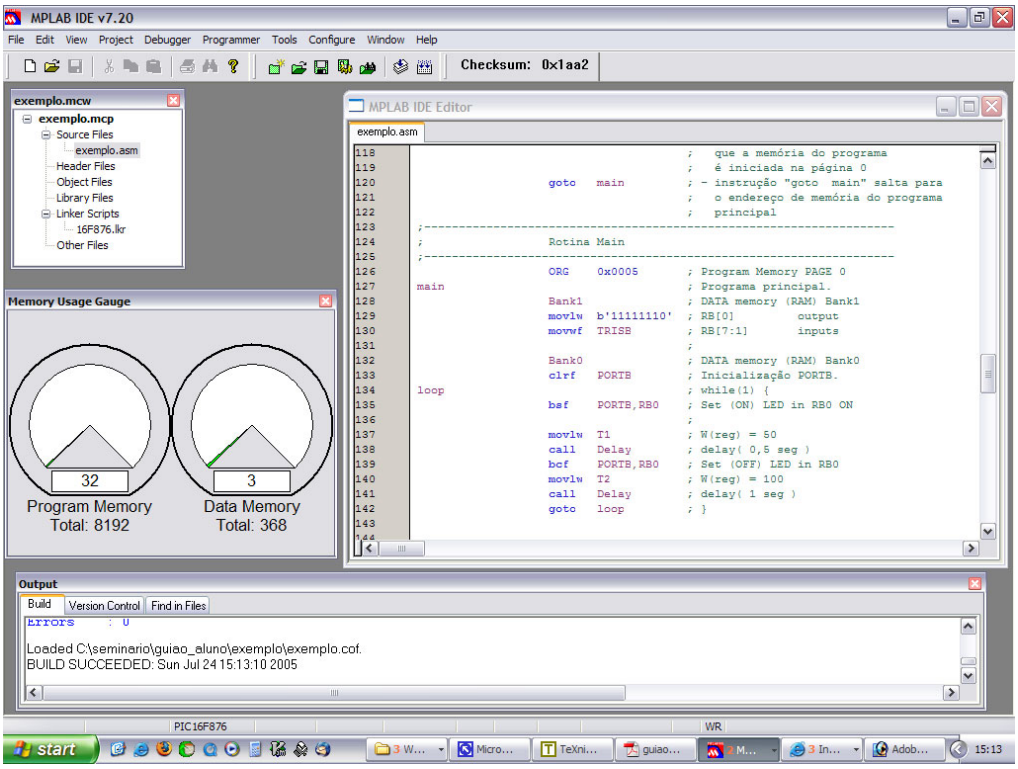


Figura 4: Projecto em MPLAB IDE

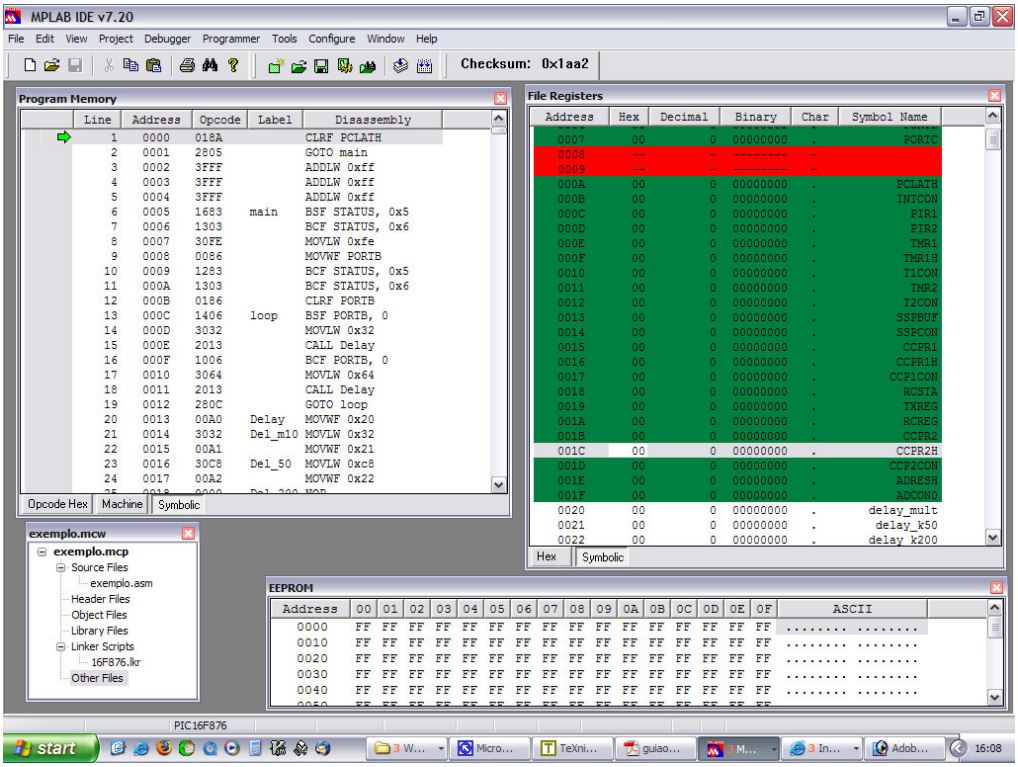


Figura 5: Visualização de janelas de informação memória do PIC

4 Trabalho a Realizar

4.1 Criação de um projecto no MPLAB IDE

1. Uma vez aberto o programa **MPLAB IDE**, na área de trabalho, selecciona-se no menu principal '*Project* → New'.

2. Quando a dialog box '*New Project*' surgir introduz-se no campo '*Project Name*' `exemplo`, e no campo '*Project Directory*' `c:\aulas\exemplo` (ver figura 6).

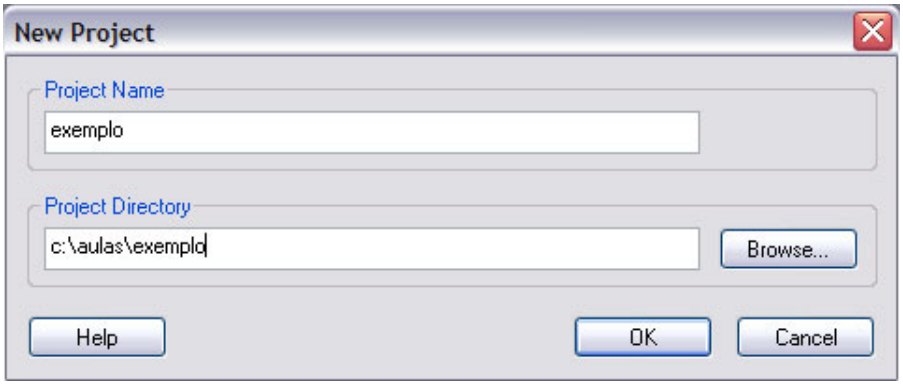


Figura 6: fase 1 - criação de um novo projecto .

3. Escolha do microcontrolador associado ao projecto: ir ao menu '*Configure → Select Device ...*' seleccionar o PIC16F876 e clicar **OK** (ver figura 7).

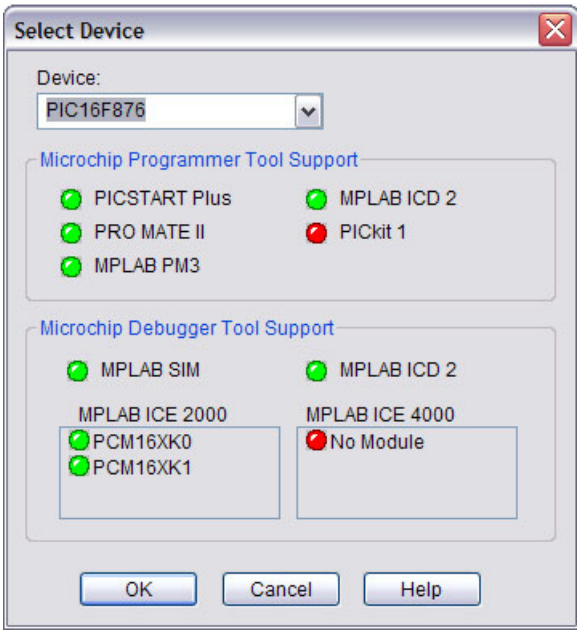
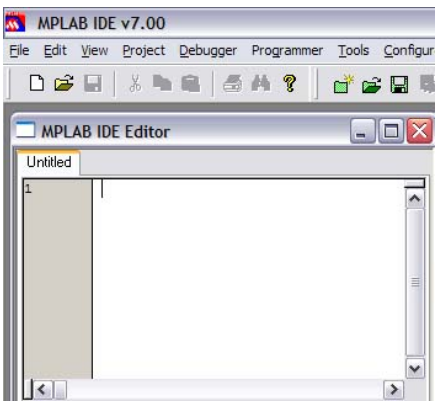
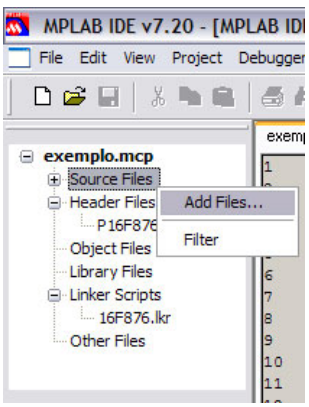


Figura 7: fase 2 - selecção do microcontrolador.

4. Ao projecto que acabou de se criar, associa-se agora o ficheiro com o código fonte do programa; se este não existir cria-se um novo (para criar um novo clicar em '*File → New*'). Antes de iniciar o processo de edição, deve-se guardar o ficheiro com a extensão `".asm"`, de modo a activar a funcionalidade *syntax highlight* (menu '*File → Save As...*' guardando o ficheiro no directório `c:\aulas\exemplo` criado anteriormente). Neste exemplo o código fonte já se encontra num ficheiro no directório do projecto. Este ficheiro pode ser adicionado ao projecto clicando com o botão direito do rato em cima de '*Source Files → Add Files...*' (ver figura 8(b)).



(a) Novo ficheiro.



(b) Adicionar ficheiro.

Figura 8: Adicionar um ficheiro fonte existente ou criar um novo.

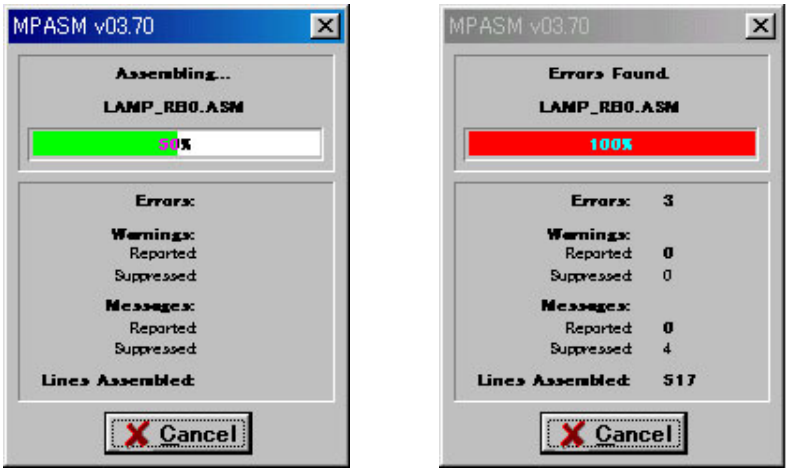
5. Dado que o projecto usa o**MPlink**, é necessário associar-se um *linker script*; este ficheiro define os comandos do *linker* para cada PIC, especificando o seguinte:

   - Regiões de memória de dados e programa do PIC.
   - Mapeamento de secções lógicas no código fonte, em regiões de programa e dados.

   Estão disponíveis ficheiros originais de *script* para todos os PIC, localizados por defeito no seguinte directório: `c:\Program Files\Microchip\MPASM Suite\LKR`. Para adicionar o ficheiro pretendido, clicar com o botão direito do rato em cima de '*Script Files* → *Add Files...*'.

4.2 Tradução do código fonte em código máquina

1. Após a edição do código fonte em *Assembly* do programa passa-se à fase de tradução do código fonte em código máquina. Para isso clica-se no ícone 'Build All'  disponível na *toolbar* do ambiente de desenvolvimento. No caso de o programa não apresentar erros de *syntax*, o gráfico de saída que mostra a evolução da compilação ficará completo a 100% e de cor verde; caso contrário ficará vermelho, o que indica a existência de um ou mais erros.
   Na ausência de erros de *syntax*, é criado o ficheiro de saída que tem como nome principal o nome do projecto com extensão `".hex"`.



(a) Evolução do *assembling*. (b) Ocorreu um erro.

Figura 9: Tradução do código fonte em código máquina.

4.3 Programação do PIC16F876 através da linha série

Após a tradução bem sucedida do código fonte em código máquina, passa-se à fase de programação do PIC. Para tal, é necessário que, previamente, tenha sido instalado no PIC um pequeno programa (o *boot-loader*) que gere a comunicação com o computador de desenvolvimento e que escreve o programa na memória *Flash*.

O que é o *boot-loader*?

O *boot-loader* é um programa que permite re-programar o PIC sem ter de o retirar do circuito onde está montado e sem ter que recorrer a um programador (i.e., a um aparelho para programar os PICs).

Como funciona?

O *boot-loader* funciona em conjunto com um programa que corre no PC, designado por **WinPIC Loader** (disponível para DOS / Windows / Linux) que controla o envio de um ficheiro em formato Intel Hex para o PIC.

Sempre que o PIC é ligado, inicia-se a execução do *boot-loader*. Este começa por inquirir o PC sobre o que fazer. Se não obtém resposta dentro de um tempo limite e se existir um programa válido na memória do PIC, então o *boot-loader* inicia a execução desse programa. No caso em que obtém uma resposta válida, inicia o processo de transferência e armazenamento de um novo programa.

Como re-programar?

1. Compilar o programa que se pretende transferir para o PIC usando o MPLAB. O compilador produz um ficheiro em formato Intel Hex (extensão `".hex"`).
2. Executar o programa **WinPIC Loader**, clicar em '*File* → *Open*' seleccionar o ficheiro `".hex"` respectivo, clicar OK.
3. Premir o botão de *reset* da placa PIC. A transferência e programação iniciam-se de imediato. A progressão é assinalada na janela WinPIC Loader no campo, *Progress:*.
4. Finalmente, para executar o programa, premir novamente o botão de *reset* da placa PIC.

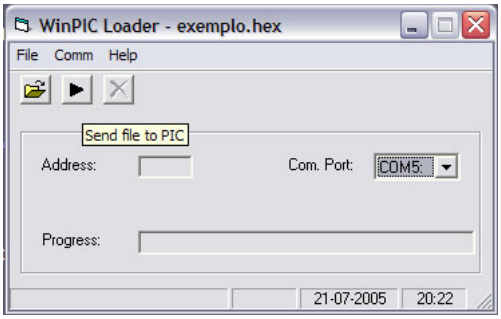


Figura 10: WinPIC Loader

4.4 Estrutura base de um programa em *Assembly*

```
        list    p=16f876            ; - directiva "list", define
                                    ;   o processador a usar
        radix   decimal             ; - directiva "radix" especifica
                                    ;   a definição das variáveis
        #include "p16f876.inc"       ; - directiva "include" inclui
                                    ;   ficheiro adicional

;------------ Definição de constantes -------------------------------


;------------ Definição de variáveis --------------------------------


;------------ Definição de macros -----------------------------------


;=================================================================
                                    ;
              ORG    0x0000         ; - directiva "ORG" origem do
                                    ;   programa "ORG 0x0000" vector
                                    ;   de reset
              clrf   PCLATH         ; - instrução "clrf PCLATH" apaga
                                    ;   todos os bits do registo garante
                                    ;   que a memória do programa
                                    ;   é iniciada na página 0
              goto   main           ; - instrução "goto  main" salta para
                                    ;   o endereço de memória do programa
                                    ;   principal
                                    ;
;------------ Rotina Main -------------------------------------------
                                    ;
              ORG    0x0005         ; - "0x0005" 1º endereço livre da
                                    ;   memória de programa
                                    ;
main                                ; - label programa principal
                                    ;
              goto   main           ;   Código assembly do
                                    ;   programa principal
                                    ;
;************ ROTINAS ****************************************************
                                    ;
                                    ;
                                    ;
                                    ;
;*****************************************************************
              END                   ; - directiva "END" fim do programa
```

4.5 Análise do programa exemplo

O objectivo do programa é manter o *led* (ligado ao bit 0 do PORTB, RB0), acesso durante um tempo **t1** e apagado durante um tempo **t2**.

Para efectuar a contagem de tempo utiliza-se a rotina *Delay* (ver código *Assembly* página, 14). Esta rotina permite a parametrização na gama [1..255] a que corresponde uma contagem de tempo na gama [0,1s..2,5s]. O registo **W** é utilizado para a passagem do parâmetro da entrada para a rotina.

O trecho de código abaixo indicado, representa o código da rotina *main* do programa. Inicialmente é feita a configuração do PORTB do PIC como saída, para que seja possível a activação/desactivação do *led* (RB0). De seguida o programa entra num ciclo infinito forçando a saída RB0 ao estado *ON* durante 0,5s e ao estado *OFF* durante 1s.

A primeira instrução (*bsf* PORTB,RB0) coloca na saída (RB0) o nível lógico '1' (5 volts). A seguinte instrução (*movlw* 50) coloca no registo **W** o valor do parâmetro de entrada da rotina *Delay*, invocada de seguida, o que vai fazer com que o *led* esteja acesso durante 0,5s. A instrução (*bcf* PORTB,RB0) coloca na saída (RB0) o nível lógico '0' (0 volts). A rotina *Delay* é novamente invocada tendo agora como valor de entrada 100, o que vai fazer com que o *led* esteja apagado durante um tempo **t2**. Este ciclo repete-se sempre.

```
main                Bank1                ; Memória RAM -> Bank 1
                    movlw  b'11111110'   ; RB[0]   output
                    movwf  TRISB         ; RB[7:1] inputs
                    Bank0                ; Memória RAM -> Bank 0
loop                                     ; while(1)
                                         ; {
                    bsf    PORTB,RB0     ;  Set LED in RB0 ON
                                         ;
                    movlw  50            ;    50 -> W(reg)
                    call   Delay         ;    delay( 0,5 seg )
                                         ;
                    bcf    PORTB,RB0     ;  Set LED in RB0 OFF
                                         ;
                    movlw  100           ;    100 -> W(reg)
                    call   Delay         ;    delay( 0,10 seg )
                                         ;
                    goto   loop          ; }
```

```
;=====================================================================;
; Delay                                                               ;
; Pode gerar delays entre 10 mS e 2,5 S                               ;
; O valor de entrada e' passado em W (1..255)                         ;
;                                                                     ;
; Notas: o loop base e' de 1 uS (T=200 nS <=> 4*1/20MHz)              ;
;                                                                     ;
;=====================================================================;
Delay
                  movwf  delay_mult   ; Carrega o valor múltiplo
Del_m10           movlw  50           ; de 10 mS
                  movwf  delay_k50    ; 50 => 10 mS
Del_50            movlw  200          ; 200 => 200 uS
                  movwf  delay_k200   ; Carrega o valor
Del_200           nop                 ; correspondente a 200 uS
                  nop
                  decfsz delay_k200,f ; Decrementa "delay_k200".
                                      ; O resultado é colocadao em
                                      ; "delay_k200"
                  goto   Del_200      ; while( delay_k200 > 0 )
                  decfsz delay_k50,f  ;
                  goto   Del_50       ; while( delay_k50 > 0 )
                  decfsz delay_mult,f ; Decrementa "delay_mult".
                                      ; O resultado é colocadao
                                      ; em "delay_mult"
                  goto   Del_m10      ; while( delay_mult > 0 )
                  return
```

- **Elementos de Apoio**

  - Manual teórico da disciplina.
  - DataSheet PIC16F876(DS30292C) Microchip, capítulo 3. - *I/O Ports*, página 29.
  - DataSheet PIC16F876(DS30292C) Microchip, capítulo 13 - *Instrution Set Summary*, página 135.

TP1 - Entrada/Saída



Tema a desenvolver	Duração
Entrada e Saída básica	2 aulas

1 Resumo

Programação e interface dos portos do PIC, através da implementação de programas em linguagem *Assembly* para leitura e escrita de dados nas linhas de entrada/saída (**E/S**) que compõem os portos.

2 Objectivos

- Familiarização do aluno com a linguagem *assembly* do PIC;
- Programação dos registos associados aos portos de **E/S**;
- Leitura e escrita de dados nos portos;
- No final do trabalho, deverá estar compreendido o mecanismo básico de **E/S** do PIC.

3 Descrição

O PIC16F876 dispõe de um total de 22 linhas de **E/S** de 1 bit. Estas estão organizados em 3 portos, denominados de porto A, porto B e porto C. Cada porto tem associado um par de registos: TRIS e PORT (ver descrição mais à frente). O porto A agrupa 5 linhas de 1 bit configuráveis, como entrada ou como saída, identificadas pelas siglas RA0, RA1, RA2, RA3 e RA4. Cada um dos portos B e C agrupa 8 linhas configuráveis como entrada ou como saída identificadas pelas siglas Rx0, Rx1, Rx2, Rx3, Rx4, Rx5, Rx6 e Rx7, em que 'x' pode tomar o valor B ou C.

Algumas destas 22 linhas têm atribuídas diversas funções (que não a de simples linha de entrada/saída), podendo o programador configurar por software a função efectivamente desempenhada. De entre as funções disponíveis destacam-se as seguintes:

- Entrada analógica, disponível, por exemplo na linha RA0/AN0 (2);
- Interrupção externa, disponível em RB0/INT (21);
- Entrada de *clock* externo do Timer 0, disponível em RA4/T0CKL (6);

- Transmissão de dados em comunicação série, RC6/TX (16);
- Recepção de dados em comunicação série, RC7/RX (18);

3.1 Registos TRIS e PORT

O registo PORT é um registo de dados, é aqui que a informação presente nas linhas de entrada ou saída é recolhida ou disponibilizada. Cada porto tem associado um registo de dados próprio, PORTA, PORTB e PORTC.

O registo TRIS é um registo programável de 1 byte (8 bits) disponível no banco 1 (memória RAM), que controla se uma linha em particular é uma entrada ou uma saída. Existe um registo TRIS para cada porto. O TRISA controla o estado de **E/S** das 6 linhas do porto A, enquanto que TRISB e TRISC controlam respectivamente o estado de **E/S** das 8 linhas do porto B e porto C. Uma vez configurada as direcções pretendidas das linhas do porto, por programação do registo TRIS, pode-se efectuar leituras ou escritas no porto usando o registo PORT.

Utilização dos registos TRIS e PORT

Todos os bits contidos nos registos mencionados, correspondem univocamente a uma linha de entrada/saída de um bit. Por exemplo, o bit 0 do registo PORTA e do registo TRISA, correspondem à linha RA0, o bit 1 à linha RA1 e assim por diante. Um '1' no bit 0 do registo TRIS configura a linha como entrada enquanto um '0' configura a linha como saída. Uma maneira intuitiva de relembrar o conceito, advém do facto de o '1' ser parecido com o 'I' que provém da inicial da palavra inglesa *Input* e o '0' ser parecido com 'O' que provém de *Output*.

A figura 11 mostra um esquema conceptual da configuração através do registo TRISB, de um porto. A linha RB0 está configurada como entrada ou como saída? Qual será o nível lógico presente no bit 0 do registo PORTB?

Observa-se na figura que o bit 0 do registo TRISB está definido a '1', logo a linha RB0 está configurada como entrada. Quanto ao valor do bit 0 no registo PORTB, temos de considerar dois momentos, uma vez que a entrada varia no tempo: ao primeiro momento corresponde o nível lógico 0 e ao segundo o nível lógico '1'.

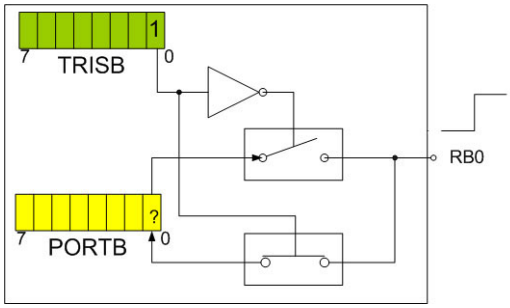


Figura 11: Configuração de uma linha em **E/S**

3.2 Inicialização dos portos

Quando é necessário usar uma ou mais linhas de **E/S** de um determinado porto, importante fazer-se a sua inicialização logo no inicio da rotina *main* do programa. A inicialização não é mais que a configuração da direcção dos dados presentes nas linhas de **E/S**, por configuração do registo TRIS.

É importante realçar que, se porventura a inicialização das linhas de **E/S** não for feita, estas por defeito (aquando do *reset* do PIC) encontram-se todas como entrada. Assim se uma das funcionalidades do programa a desenvolver for a recepção de informação digital em todas as linhas de um determinado porto (à excepção do **porto A**), este não necessita de ser inicializado.

RA4 é a única linha do porto A que pode ser configurada como entrada digital pelo registo TRISA, para as restantes é necessário também a configuração do registo ADCON1 (descrição feita no trabalho prático 9).

O trecho de código *Assembly* seguinte, exemplifica uma inicialização por programação de todos os portos do PIC. Este mostra, para cada porto, 3 formas de configuração do registo TRIS:

1. Transferência para o registo W do literal em binário, e consequente escrita em TRIS;
2. Colocação de todos os bits do registo TRIS a zero;
3. *Read modify write*, coloca apenas o bit correspondente a zero.

```
;---------------- I/O - Config -------------------------------------;

                    Bank1               ; DATA memory (RAM) Bank1
                                        ;
config 1            movlw  b'00010000'  ; RA[7:5 & 3:0]   saídas
                    movwf  TRISA        ; RA[4]           entradas
                                        ;
config 2            clrf   TRISB        ; RB[7:0]
                                        ;
config 3            bcf    TRISC,RC0    ; RC[7:1]         entradas
                                        ; RC[0]           saída
                                        ;
                    Bank0               ; DATA memory (RAM) Bank0
                    clrf   PORTA        ; Apaga todos os bits do PORTA
                    clrf   PORTB        ; Apaga todos os bits do PORTB
                    clrf   PORTC        ; Apaga todos os bits do PORTC
```

4 Trabalho a realizar

1. Alteração do programa exemplo *exemplo.asm* de modo a que, através do botão de pressão S2 da placa PIC ligado a RA4 (ver esquema da figura 12), seja possível controlar o estado do *led* ligado ao pino RB0; botão premido → *led* ligado, botão não premido → *led* desligado.

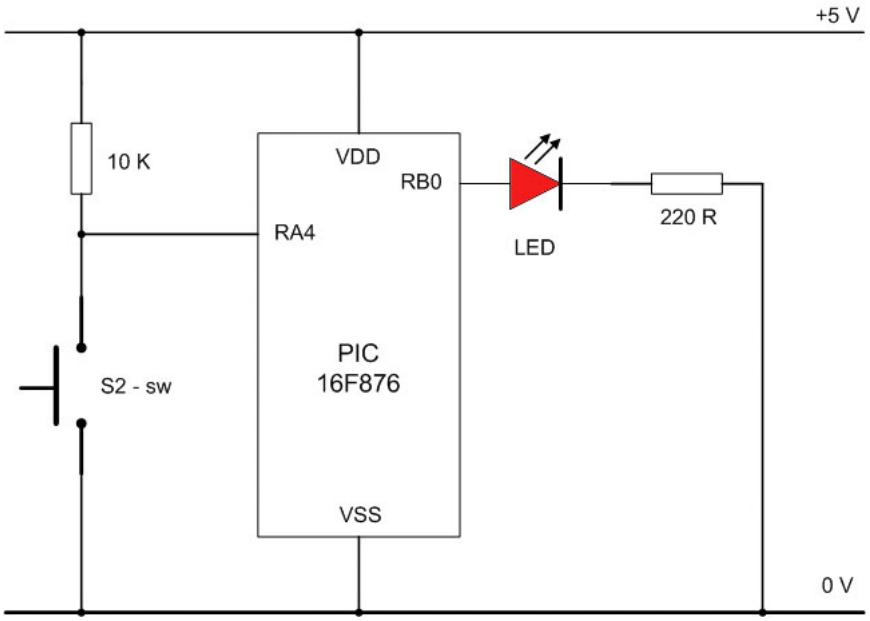


Figura 12: Diagrama do circuito, led on/off.

2. Mantendo o circuito, alterar o programa anterior por forma a permitir o controlo temporizado do *led* ligado a RB0.
   Inicialmente o *led* é activado por pressão no botão ligado a RA4. Após se ter deixado de premir o botão, manter durante um tempo **t** o *led* acesso. A figura 13, mostra o comportamento pretendido para a saída, com **t**=2,5s.

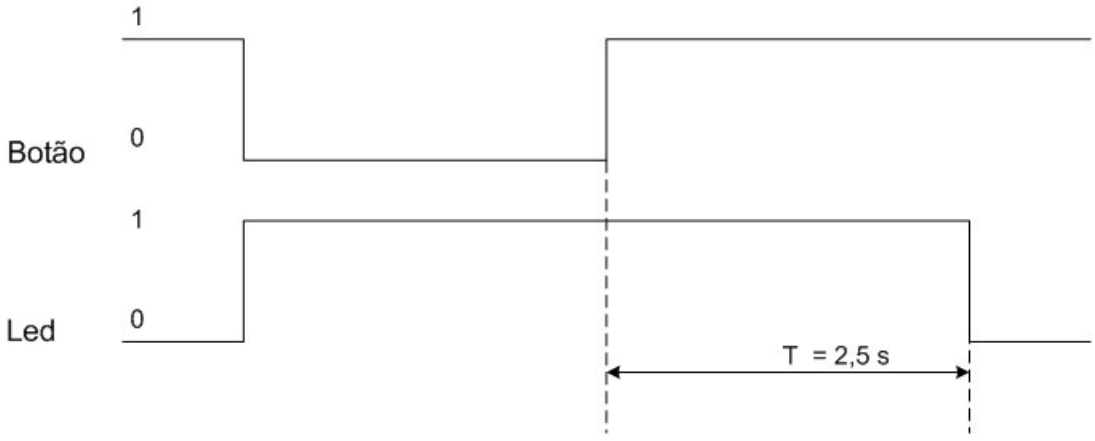


Figura 13: Diagrama temporal da saída temporizada

3. Construção de um programa, que implemente um contador binário *up/down*. O incremento ou decremento deve ser feito com um intervalo de tempo fixo (aproximadamente de 0,5 segundos), através de 4 *leds*, ligados aos portos RB3 a RB0 da placa PIC (ver esquema da figura 14). Inicialmente é feito o incremento de valores em binário no PORTB, contudo, se premir continuamente o botão de pressão ligado a RA4, dever-se-à, visualizar o seu decremento. A figura 14 mostra também a visualização do valor '5' correspondente ao valor binário 0101, sendo que o bit mais significativo corresponde ao *led3* ligado a RB3.

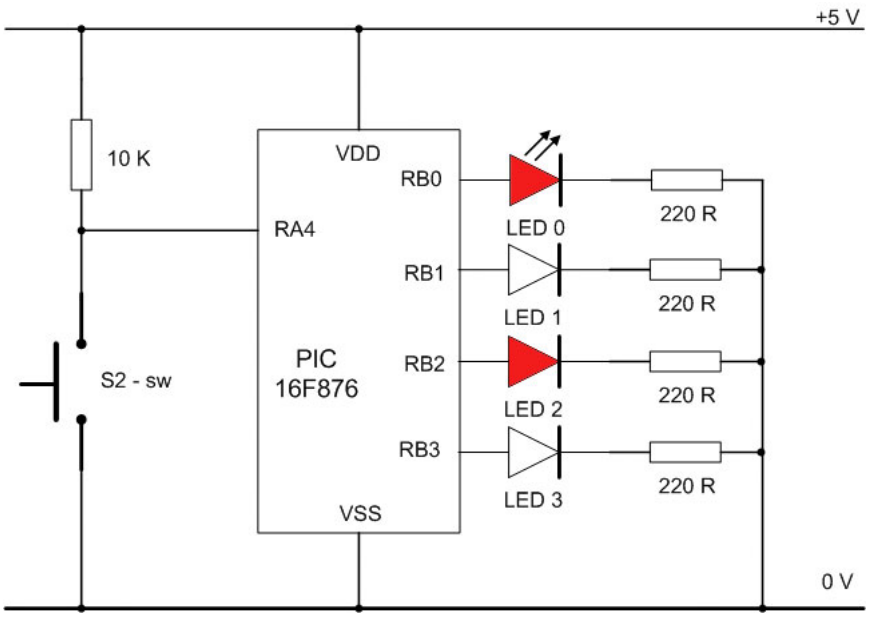


Figura 14: Diagrama do circuito para o contador up/down

• Elementos de Apoio

- Manual teórico da disciplina.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 3. - *I/O Ports*, página 29.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 11. - *Analog-to-Digital Converter (A/D) Module*, página 111.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 13 - *Instrution Set Summary*, página 135.

NOTAS:

TP2 - Descodificador Hexadecimal / 7Seg, 1 display

Tema a desenvolver	Duração
Implementação de uma *lookup table*	1 aula

1 Resumo

Com este trabalho, pretende-se consolidar os conhecimentos adquiridos nos trabalhos anteriores e estudar o modo de acesso a variáveis em memória RAM, por endereçamento indirecto.

2 Objectivos

- Implementação de um descodificador hexadecimal / 7Segmentos por software.
- Utilização de subrotinas na estrutura do programa.
- Compreensão dos conceitos associados ao acesso a variáveis na memória RAM por endereçamento indirecto.

3 Descrição

3.1 Endereçamento Indirecto, Registo FSR e INDF

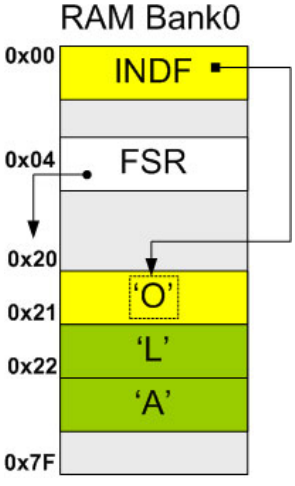


Figura 15: Acesso memória

O **FSR** (*File Select Register*). É usado no acesso em endereçamento indirecto ou indexado de outros registos, em particular dos registos de uso geral. Se um registo com endereçamento entre [0x20-0x7F] (Banco 0 de memória RAM) é carregado no **FSR**, o conteúdo desse registo pode ser lido ou escrito através do registo de endereçamento indirecto (**INDF**), localizado no topo de cada banco de memória (ver mapa de registos do PIC, Anexo A). Este método pode ser usado no acesso a um grupo de dados localizados em memória RAM, através da leitura ou escrita de dados em INDF, a selecção do próximo elemento do grupo é feito através do incremento de **FSR** (ver figura 16). O endereçamento indirecto ou indexado, é particularmente útil na salvaguarda em memória RAM, de um grupo de dados lidos de um dado porto E/S, e, por exemplo no acesso a *arrays* ou tabelas.

Endereçamento Indirecto - Exemplo

Uma demonstração de endereçamento indirecto, é o trecho de código *Assembly* seguinte, em que o objectivo é apagar posições consecutivas de memória RAM, localizadas entre 0x20h e 0x2F. Inicialmente é feita a inicialização de **FSR** com o primeiro valor do grupo de valores a ser apagado, ou seja **FSR** opera como ponteiro para o valor '0' do *array* (bloco de valores), sendo incrementado a cada operação de escrita. Em cada ciclo "NEXT", está-se na realidade a apagar (**clrf INDF**) os conteúdos das posições de memória apontadas por FSR, usando para isso o registo INDF, que não é um registo físico e que só mostra o conteúdo do endereço guardado em FSR.

```
                movlw   0x20        ; inicializa ponteiro
                movwf   FSR         ; copia para a RAM
                movlw   n_val       ; nº de posições mem
                movwf   i           ; contador
NEXT            clrf    INDF        ; apaga registo INDF
                incf    FSR,F       ; incrementa ponteiro
                decfsz  i,f         ; contador--
                goto    NEXT        ; apaga o próximo
CONTINUE
                :                   ; continua
```

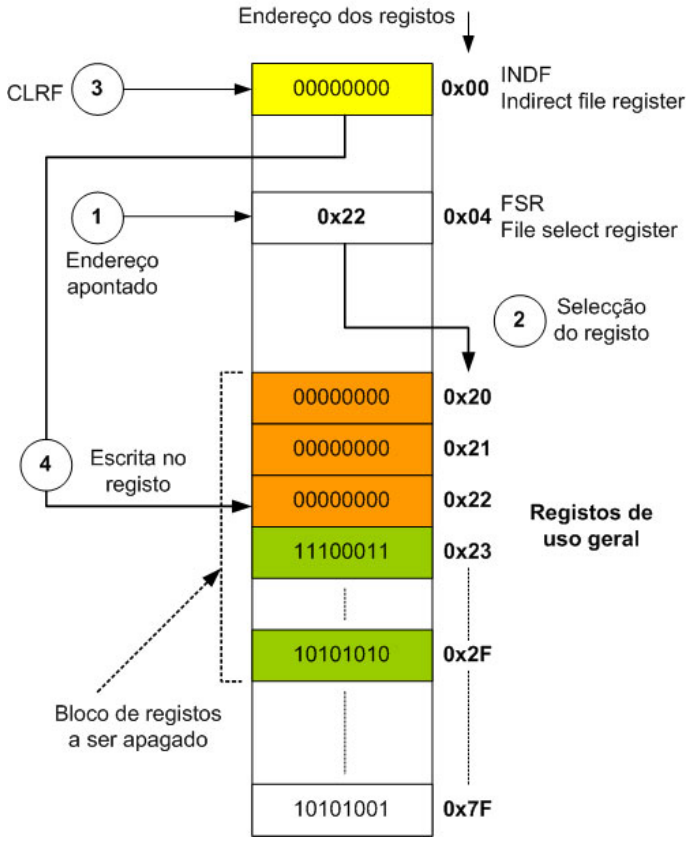


Figura 16: Endereçamento Indirecto código de demonstração

3.2 Selecção do Banco de Memória

A memória de dados (RAM) encontra-se organizada em bancos distintos (Banco 0, 1, 2 e 3) 4 no total, sendo que o acesso a cada um deles é feito através de selecção. A selecção pode ser feita através de endereçamento directo (ver registo STATUS página 27) ou indirecto. neste último o bit mais significativo do registo **FSR**, é usado como bit menos significativo na selecção do banco de memória a aceder; os restantes 7 bits são usados na especificação do registo dentro do banco de memória seleccionado. O bit mais significativo para selecção do banco de memória, é dado pelo bit IRP do registo de **STATUS**. Para mais detalhes ver descrição sobre o registo **STATUS**, página 27 do guia.

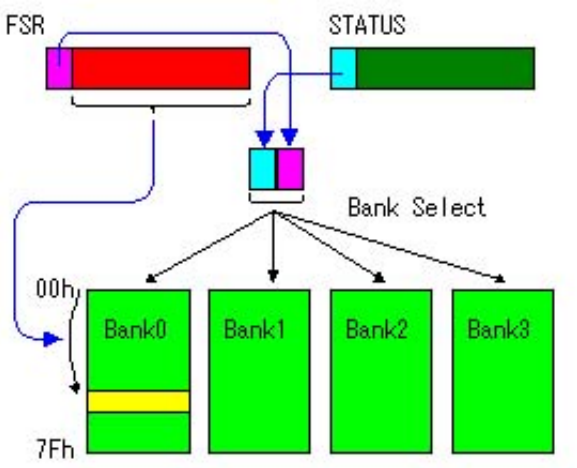


Figura 17: Selecção do banco de memória

4 Trabalho a realizar

1. Visualização, num *display* de 7 segmentos, de um digito codificado em hexadecimal (ver diagrama do circuito, figura 18).
   Para isso é necessária a construção de uma rotina que receba um digito em hexadecimal e devolva o correspondente valor para activação de cada um dos segmentos do *display*. Este valor deverá ser depois enviado para o PORTB.
   Para melhor compreender o conceito, a tabela 2 mostra a correspondência entre os dígitos hexadecimal de entrada e o valor a enviar para cada um dos 7 segmentos do *display*. Por exemplo, para visualizar o digito 7 é necessário que o segmento a,b e c estejam activos, isto é, mover para o **PORTB** o valor em binário b'00001110'.

2. Construir finalmente um contador [0..F] crescente/decrescente por pressão nos botões S1/S2 respectivamente (S1 encontra-se ligado a RB0, S2 encontra-se ligado a RA4).

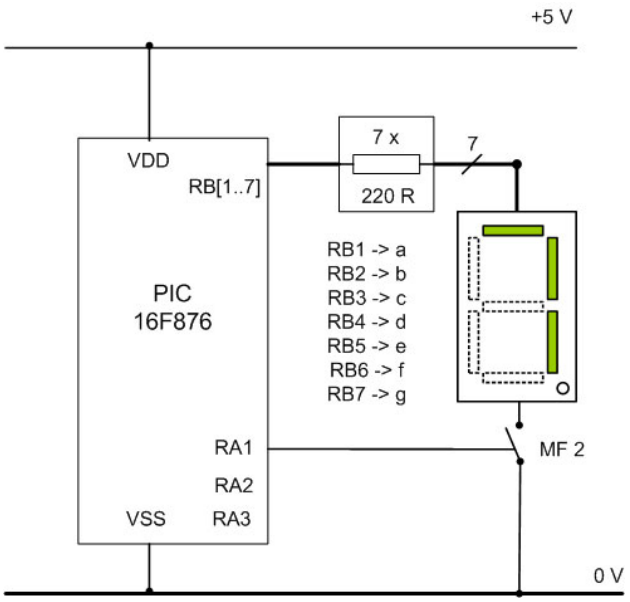


Figura 18: Diagrama do circuito, 1 *display* de 7 segmentos

Valor BCD	PORTB ➟ Segmentos do Display						
	RB7 ➟ g	RB6 ➟ f	RB5 ➟ e	RB4 ➟ d	RB3 ➟ c	RB2 ➟ b	RB1 ➟ a
0	0	1	1	1	1	1	1
1	0	0	0	0	1	1	0
2	1	0	1	1	0	1	1
3	1	0	0	1	1	1	1
4	1	1	0	0	1	1	0
5	1	1	0	1	1	0	1
6	1	1	1	1	1	0	0
7	0	0	0	0	1	1	1
8	1	1	1	1	1	1	1
9	1	1	0	0	1	1	1
A	1	1	1	0	1	1	1
B	1	1	1	1	1	0	0
C	0	1	1	1	0	0	1
D	1	0	1	1	1	1	0
E	1	1	1	1	0	0	1
F	1	1	1	0	0	0	1

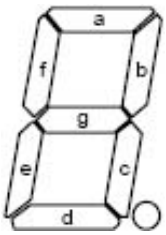


Tabela 2: Valores Hexadecimal ➟ 7 segmentos.

• Elementos de Apoio

- Manual teórico da disciplina.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 2. - *Memory Organization*, secção 2.5 - *Indirect Addressing, INDF and FSR Registers* página 27.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 3. - *I/O Ports*, página 29.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 13 - *Instrution Set Summary*, página 135.

TP3 - Descodificador Hexadecimal / 7Seg, 3 Displays



Tema a desenvolver	Duração
Sistema de visualização com refrescamento, por multiplexagem no tempo	2 aulas

1 Resumo

Ampliação do trabalho prático anterior a 3 displays de 7 segmentos.

2 Objectivos

- Implementação de um sistema de visualização hexadecimal com 3 *displays*, com a construção de uma rotina para refrescamento dos 3 *displays* por multiplexagem no tempo.

- Aprofundar os conhecimentos adquiridos durante o trabalho prático 3.

3 Descrição

O princípio de funcionamento deste circuito é em tudo igual ao circuito do trabalho anterior, o número dos *displays* é que vai ser alterado. Serão 3 os *displays* que terão de ser refrescados no tempo, para que os valores enviados possam ser visualizados em simultâneo.

Analisando os diagramas dos circuitos (ver página 19) entende-se que apenas um valor é enviado, de cada vez para um só *display*, (só um interruptor se encontra fechado). Note-se ainda que os todos os segmentos dos *displays* estão ligados á mesma saída (PORTB RB[7..1]) do PIC. Assim e tendo em conta que os dados na saída são diferentes, aquando do envio de informação para cada *display* ao mesmo tempo. Estes devem ser actualizados periodicamente com uma taxa fixa (taxa de refrescamento aprox. = 10ms). uma vez que o olho humano não tem a percepção do envio da informação enviada aos três *displays* ao mesmo tempo.

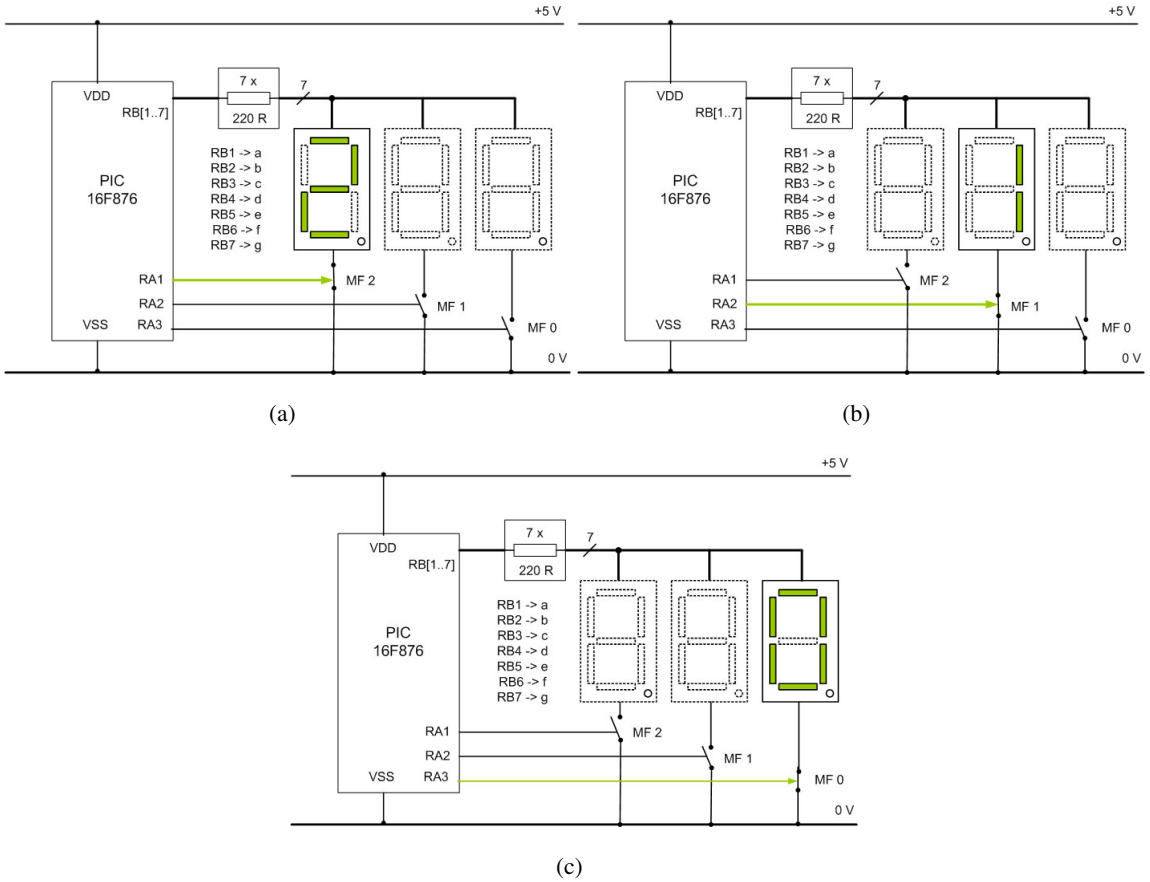


Figura 19: Processo de refrescamento dos *displays*

4 Trabalho a realizar

1. Construção de um sistema de visualização em 3 *displays* de 7 segmentos, de valores codificados em hexadecimal. Para isso, sugere-se o reaproveitamento da "bcd_7seg" construída no trabalho anterior. De modo a que a rotina "bcd_7seg" seja chamada por uma outra rotina, esta controla o envio da informação para cada *display* em momentos distintos, ao longo do tempo.

• Elementos de Apoio

- Manual teórico da disciplina.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 2. - *Memory Organization*, secção 2.5 - *Indirect Addressing, INDF and FSR Registers* página 27.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 3. - *I/O Ports*, página 29.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 13 - *Instrution Set Summary*, página 135.

4.1 Descrição do registo STATUS

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

O registo STATUS, fornece informações sobre o resultado de instruções aritméticas realizadas na ALU (Unidade Aritmética e Lógica do CPU), do banco de memória seleccionado e do estado de reset.

bit 7 **IRP**: Usado na selecção do banco de memória em endereçamento indirecto.

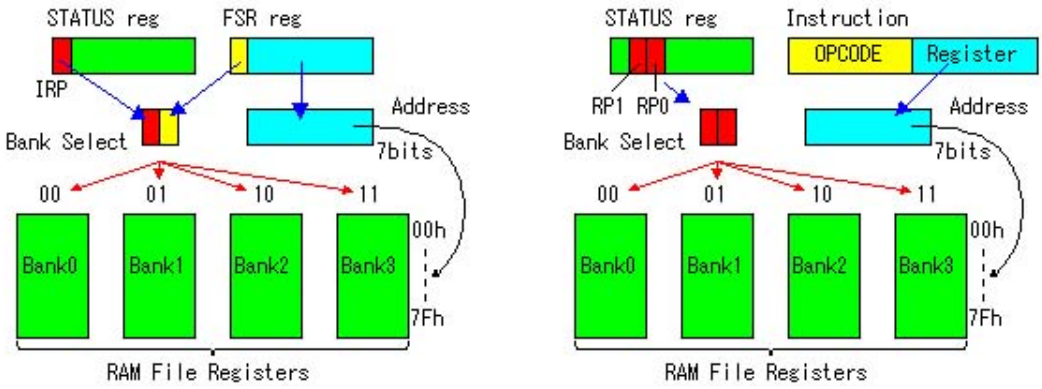
1 = Bank 2,3 (100h - 1FFh)

0 = Bank 0,1 (00h - FFh)

Como se observa na figura 20(a), o banco de memória é especificado com o bit IRP e o bit mais significativo do registo FSR. Mais detalhes sobre o registo **FSR**, ver página 21 do guia.

Um endereço dentro do banco seleccionado é especificado pelos 7 bits menos significativos do registo FSR.

Uma vez que o registos STATUS e FSR são comuns a todos os bancos de memória RAM, não existe obstáculo na operação, mesmo que o banco de memória RAM seja alterado.



(a) Selecção com, IRP e FSR bits

(b) Selecção com RP1:RP0 bits

Figura 20: Selecção dos bancos de memória RAM.

bit 6-5 **RP1:RP0**: Usados na selecção do banco de memória em endereçamento directo.

11 = Bank 3 (180h - 1FFh)

10 = Bank 2 (100h - 17Fh)

01 = Bank 1 (80h - FFh)

00 = Bank 0 (00h - 7Fh)

Na figura 20(b), o banco de memória pretendido é especificado com os bits RP1:RP0. Uma forma de optimizar este procedimento de alternância entre bancos é através do uso de *MACROS*. Para isso, basta definir inicialmente uma *macro* para cada banco de memória RAM. No exemplo seguinte temos

uma *macro* para o Bank1 de memória RAM, a generalização para outros bancos é possível desde de que, seja respeitada a configuração dos bits RP1:RP0.

```
Bank1               MACRO            ; Macro to select data RAM Bank 1
                    bsf    STATUS,RP0; Set '1' - bit RP0
                    bcf    STATUS,RP1; Set '0' - bit RP1
                    ENDM             ;
```

bit 4 $\overline{\textbf{TO}}$: Condição de *time-out* do ***watchdog timer***, é permitida apenas leitura.

1 = Após power-on, reset, instrução de CLRWDT, ou com instrução de SLEEP.
0 = Ocorrência de time-out do **watchdog timer**.

bit 3 $\overline{\textbf{PD}}$: Condição de power-down (poupança de energia), é permitida apenas leitura.

1 = Após power-up, ou após a execução da instrução de CLRWDT.
0 = Pela execução da instrução de SLEEP.

bit 2 **Z**: Flag de Z (zero flag).

1 = Quando o resultado de operação lógica ou aritmética é zero
0 = Quando o resultado de operação lógica ou aritmética não é zero.

NOTA: Provavelmente uma das flags mais utilizadas, usada na detecção de zero no resultado de uma operação lógica ou aritmética.

bit 1 **DC**: Flag de Digit Carry.

1 = Quando o ocorre carry no bit 3 do resultado.
0 = Quando não ocorre carry no bit 3 do resultado.

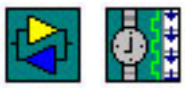
NOTA: A flag é activa, aquando da execução de uma operação, ADDWF, ADDLW, SUBLW ou SUBWF, causando carry no bit 3 do resultado (operações de 4 bits).

bit 0 **C**: Flag de carry

1 = Quando o ocorre carry no bit 8 do resultado.
0 = Quando não ocorre carry no bit 8 do resultado.

NOTA: Esta flag é activa, quando a operação sobre o registo causa um carry no bit 8 do resultado (operações de 8 bits).

TP4 - Contador de 60 segundos, versão I



Tema a desenvolver	Duração
Contagem de tempo, por contagem de instruções	2 aulas

1 Resumo

Implementação de um contador de 60 seg., usando contagem do tempo por execução de um programa.

2 Objectivos

- Manipulação da rotina *Delay* para contagem de tempo;
- Envio de informação para dois *displays* a partir de um registo de 8 bits;
- Construção de uma rotina de decremento de uma variável em BCD.

3 Descrição

O tempo de execução de uma instrução normal é de 1 ciclo de relógio (200 ns), com a excepção das instruções de salto condicional, **decfsz, incfsz, btfss, btfsc**, (apenas quando executada a instrução de salto) e as de salto incondicional, **goto**, que levam 2 ciclos de relógio (400 ns). Assim e de forma precisa, pode-se o contar tempo recorrendo à contagem de instruções de um determinado trecho de código, exemplo:

```
                                ; nº de ciclos de instruções
                                ;--------------------------
delay           movlw   100     ;                       1
                movwf   timer   ;                     + 1
loop            nop             ;             + (1 x 100)
                nop             ;             + (1 x 100)
                decfsz  timer,f ;             + (1 x 100)
                goto    loop    ;          + (2 x 99) + 1
                                ;--------------------------
                return          ; Total               501
```

O tempo que o processador leva a executar cada um dos primeiros 99 ciclos de *loop* é de 1μs, uma vez que, a instrução **nop** demora 1 ciclo de relógio, **decfsz** 1 ciclo de relógio e **goto** 2 ciclos de relógio. O último ciclo de *loop* demora menos 200ns, dado que a instrução **goto** não é executada. Logo o tempo total é de aproximadamente 100μs (mais exactamente 99,8μs).

3.1 Rotina Delay

Disponível deste o primeiro trabalho, a rotina *Delay* implementa a funcionalidade de contagem de tempo por contagem de instruções. Um exemplo disso, é o trabalho prático 0, onde o atraso efectuado pela rotina *Delay*, é útil no controle do tempo de **on/off** do *led*.

A figura 21, mostra o diagrama de fluxo da rotina *Delay*, mostrando as várias etapas na contagem do tempo.

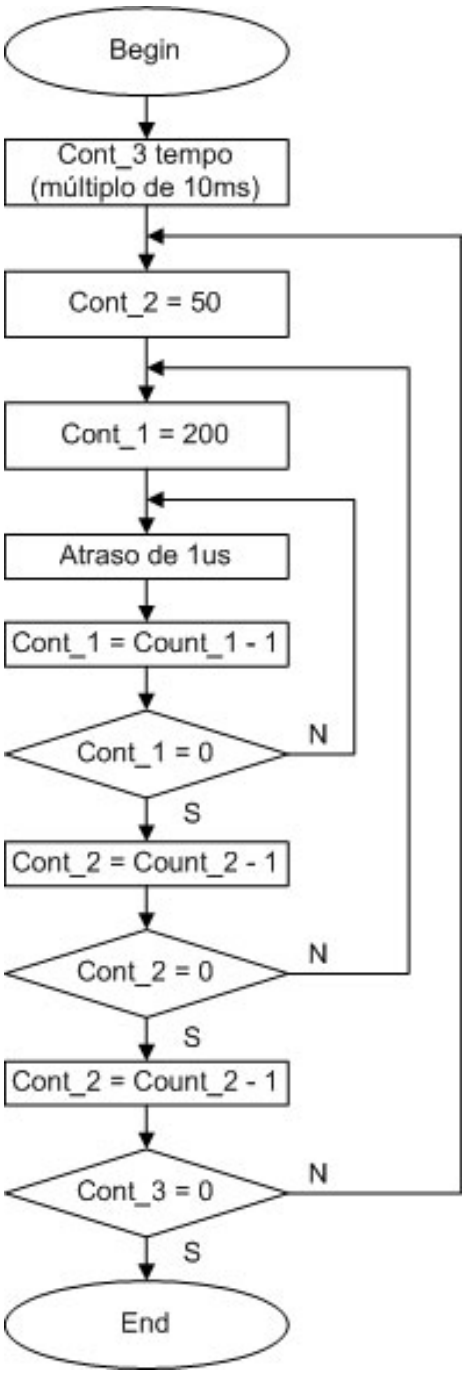


Figura 21: Diagrama de fluxo da rotina Delay

Abaixo mostra-se o código *Assembly* da rotina *Delay*, de notar que, o ciclo base de 1μs da rotina, foi feito um pouco à imagem do trecho de código exemplo descrito no início do trabalho, recorrendo às instruções de **nop** (no operation), *decfsz* (decrement skip if zero) e *goto*. No entanto, envoltos no ciclo base estão mais dois ciclos de *loop* que provocam um aumento do valor de instruções executadas o que implica necessariamente um aumento de tempo. Estes ciclos **Del_200** e **Del_50**, respectivamente de 1μs$\times$200 = 200μs e 1μs$\times$200$\times$50 = 10ms, permitem que para uma entrada compreendida entre [1..255] podemos ter atrasos de entre 10ms e 2,5s.

```
;=================================================================;
; Delay                                                           ;
; Pode gerar delays entre 10 mS e 2,5 S                           ;
; O valor de entrada e' passado em W (1..255)                     ;
;                                                                 ;
; Notas: o loop base e' de 1 uS (T=200 nS <=> 4*1/20MHz)          ;
;                                                                 ;
;=================================================================;
Delay
                movwf  delay_mult   ; Carrega o valor múltiplo
Del_m10         movlw  50           ; de 10 mS
                movwf  delay_k50    ; 50 => 10 mS
Del_50          movlw  200          ; 200 => 200 uS
                movwf  delay_k200   ; Carrega o valor
Del_200         nop                 ; correspondente a 200 uS
                nop
                decfsz delay_k200,f ; Decrementa "delay_k200".
                                    ; O resultado é colocadao em
                                    ; "delay_k200"
                goto   Del_200      ; while( delay_k200 > 0 )
                decfsz delay_k50,f  ;
                goto   Del_50       ; while( delay_k50 > 0 )
                decfsz delay_mult,f ; Decrementa "delay_mult".
                                    ; O resultado é colocadao
                                    ; em "delay_mult"
                goto   Del_m10      ; while( delay_mult > 0 )
                return
```

4 Trabalho a realizar

1. Implementação de um programa que permita a visualização (com refrescamento a cada 10ms) em dois *displays* de 7 segmentos, de um valor presente em uma variável (registo de uso geral), inicializada com o valor '0x60'. A visualização deve ser feita utilizando as rotinas "display" e "bcd_7seg" desenvolvidas nos trabalhos anteriores. A figura 22 mostra o esquema de ligação pretendido, bem como a estratégia a usar na codificação dos bits a enviar para cada *display* de 7 segmentos.

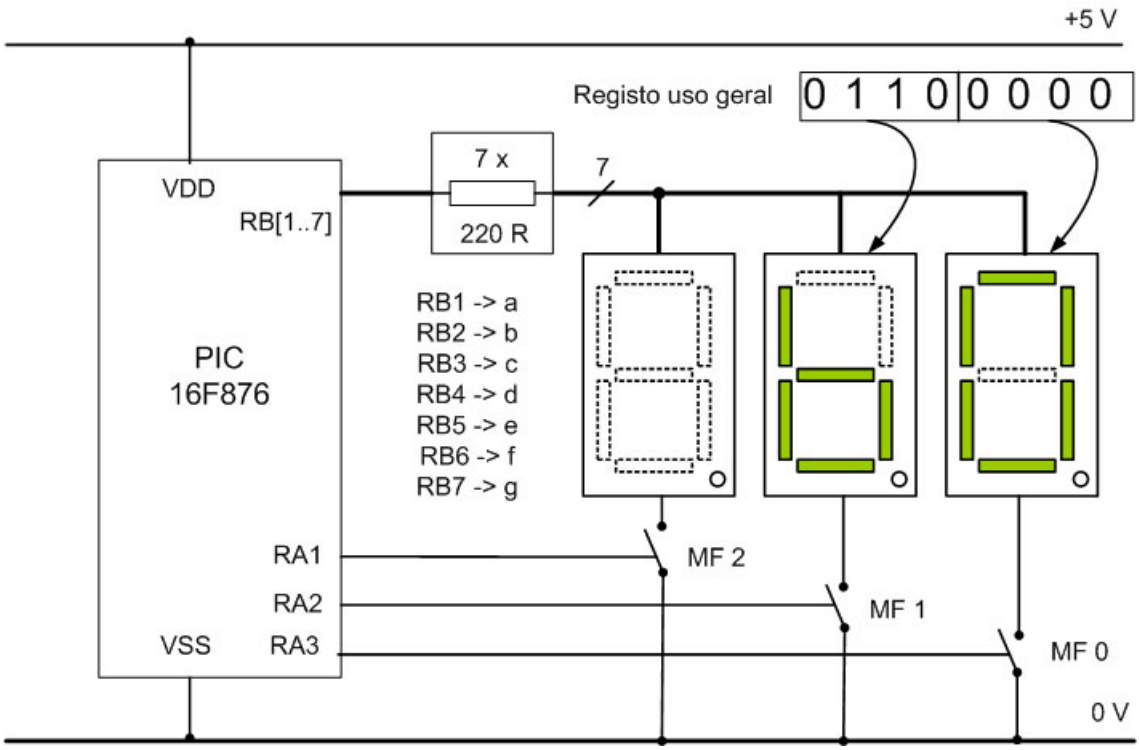


Figura 22: Diagrama do circuito para para visualização do valor '0x60'

2. Construir uma rotina que implemente a contagem decrescente do valor '0x60' presente na variável, inicializada no ponto anterior. O decremento deve ser feito em decimal, ou seja de '60s' ➠ '59s' até '0s' (ver figura 23).

   Esta rotina deve ser chamada, a cada segundo, no programa que implementa a visualização da variável descrita no ponto anterior. O programa principal deve, ao detectar a chegada da variável a '0', enviar para os *displays* a mensagem *End* (ver figura 24)

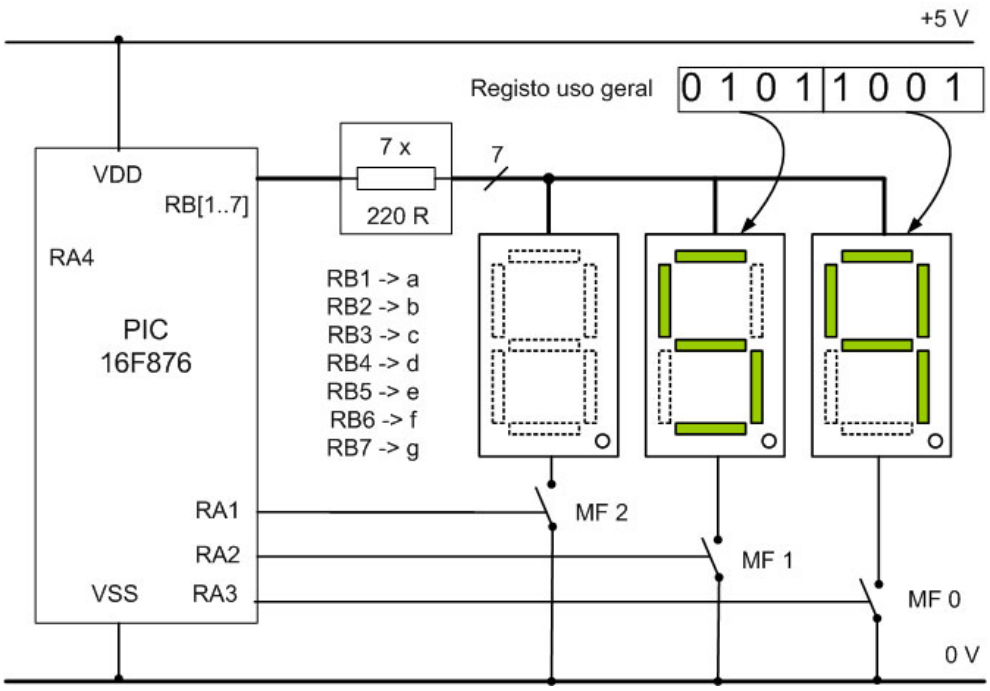


Figura 23: Diagrama do circuito, exemplifica a passagem de '60' ➠ '59'

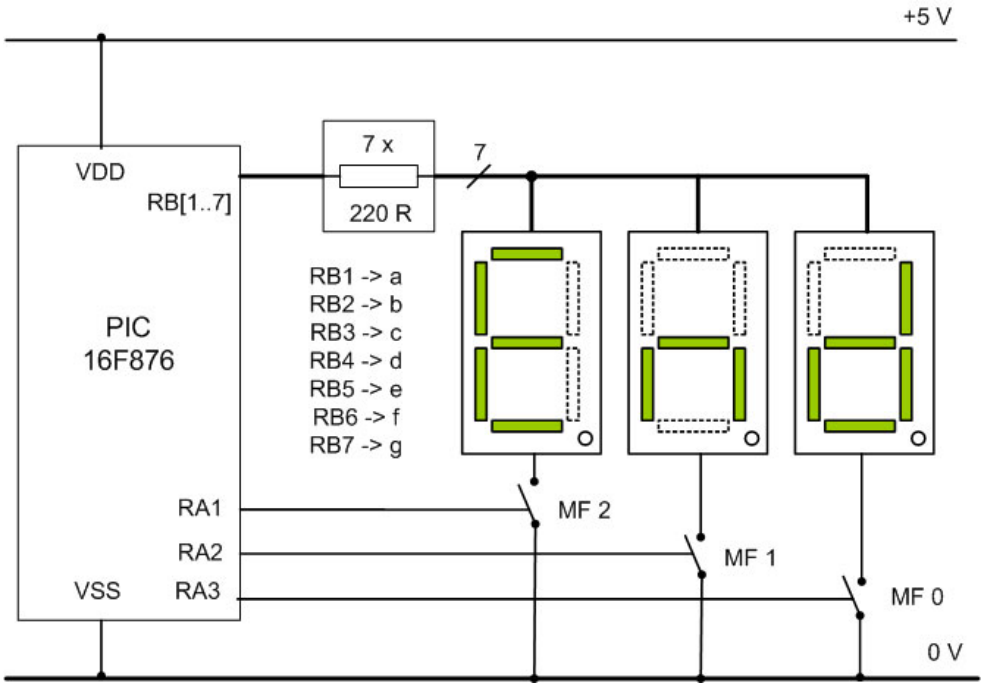


Figura 24: Diagrama do circuito, exemplifica o término da contagem

3. Depois do contador de 60s estar a funcionar correctamente, deve adicionar-se um botão de pressão que implemente apenas o START do início da contagem dos '60s' (ver figuras 25 e 26).

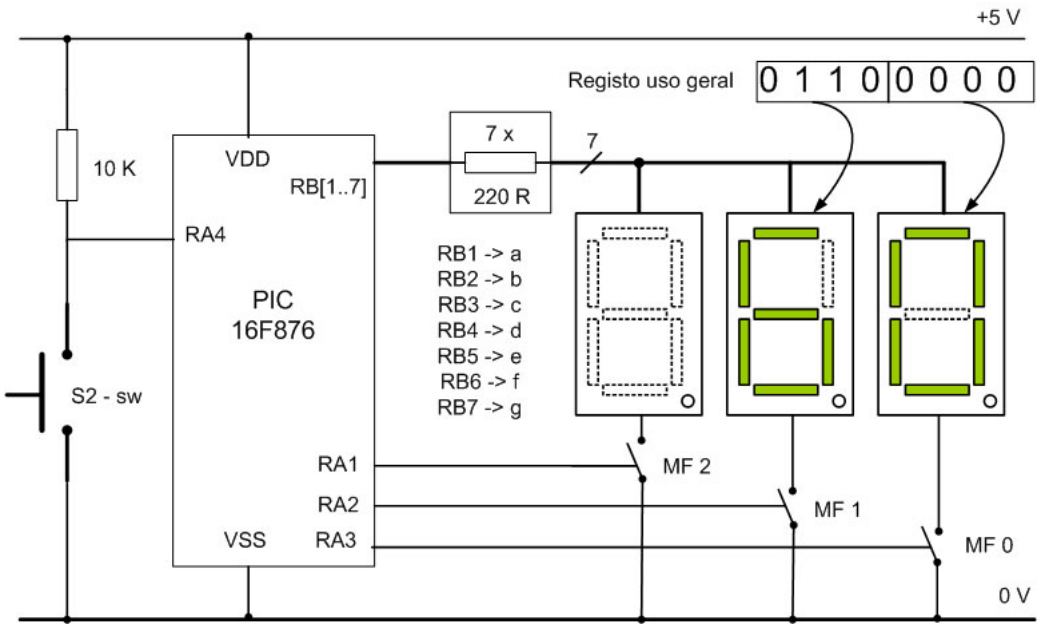


Figura 25: Diagrama do circuito com sw, antes de se pressionar no botão

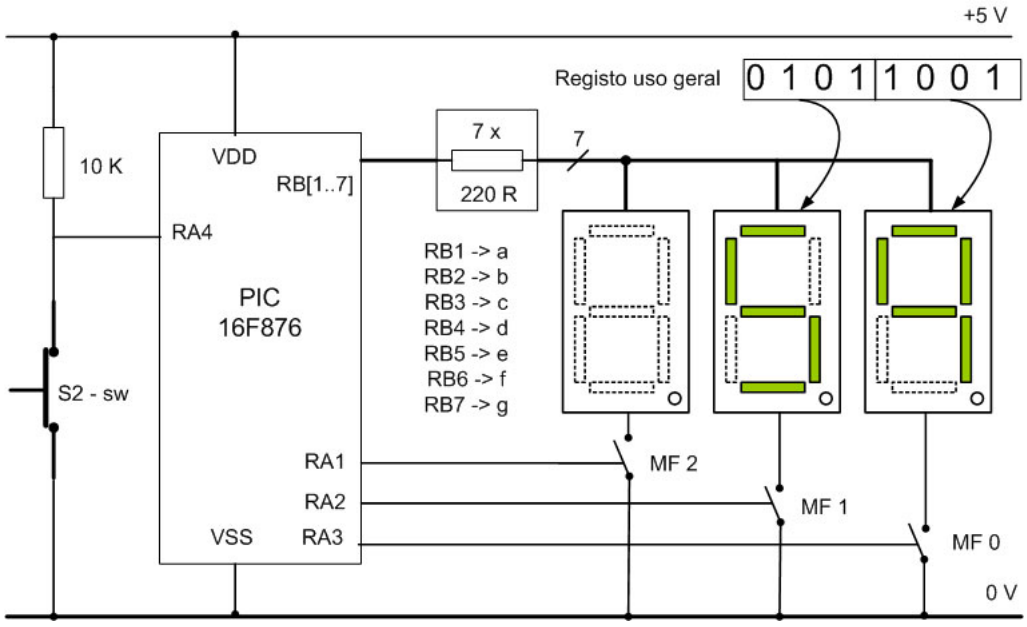


Figura 26: Diagrama do circuito com sw, após pressionar no botão

• Elementos de Apoio

- Manual teórico da disciplina.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 3. - *I/O Ports*, página 29.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 13 - *Instrution Set Summary*, página 135.

TP5 - Relógio de 60 segundos, versão II



Tema a desenvolver	Duração
Contagem de tempo com recurso a *timers*	1 aula

1 Resumo

Implementação de um relógio de 60s, usando um temporizador (*timer 0*) para contagem de tempo.

2 Objectivos

- Introdução à programação do *timer 0*, compreensão dos conceitos associados à configuração;
- Modificar o programa desenvolvido em TP4, implementado o refrescamento dos *displays* por *timers*;

3 Descrição

Em muitos programas escritos para microcontroladores, é muitas vezes necessário medir tempo de determinados eventos ou efectuar contagens. Por exemplo, um sensor ligado a uma roda de bicicleta pode dar-nos o nº de rotações em cada segundo/minuto, isto é conseguido com um registo contador que conta as voltas completas da roda, e um registo *timer* que conta os pulsos de relógio, sinalizando com um sinal interno a passagem de 1 segundo/minuto.

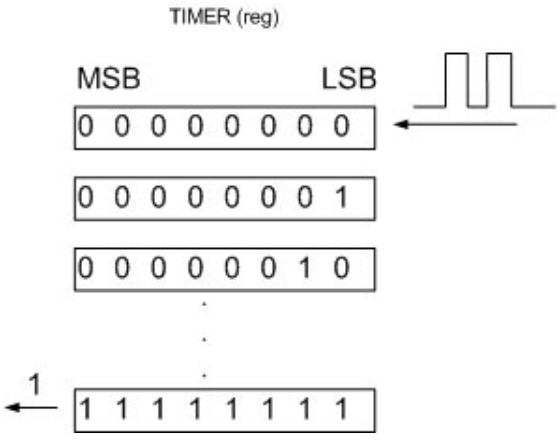


Figura 27: registo contador de 8 bits

Um registo contador/*timer* conta o número de pulsos digitais aplicados à sua entrada. Se o sinal de relógio usado é de uma frequência conhecida, estamos perante um registo *timer*, uma vez que o tempo de duração da contagem é igual ao valor apurado multiplicado pelo período de relógio.

A figura 27 mostra um registo contador/*timer* de 8 bits, com entrada no bit menos significativo (LSB) à direita. O valor binário guardado no registo incrementa a cada pulso aplicado. Três pulsos

relógio foram aplicados inicialmente, mostrando o contador o valor binário 3. Depois de 255 pulsos de relógio terem sido aplicados, o próximo pulso efectua a passagem, de '11111111' (0xFF) a '00000000' (0x00), sendo que a passagem é sinalizada através de um sinal de saída, a este sinal dá-se o nome de *timer overflow*. Pode ser gerado por interrupção (ver trabalho prático 7), ou por *polling*, a vantagem de usar uma interrupção para o *timer* é clara uma vez que o programa pode realizar outras operações, enquanto aguarda pela sinalização do *overflow*.

3.1 Timer 0

- Especificação do *timer 0*
  - O temporizador / contador de 8 bits, TIMER 0 pode contar até 0xFF (255);
  - Permite leitura e escrita;
  - Pré-divisor de 8 bits programável;
  - Selecção de clock interno (FOSC/4) ou externo;
  - Activação por *overflow* do bit 2 INTCON TOIF, na passagem de 0xFF para 0x00;
  - Selecção de flanco ascendente ou descendente do *clock* externo;
  - Pode gerar interrupções de fim de contagem, bit 2 INTCON TOIF, desde que o bit 7 INTCON GIE, esteja activo.

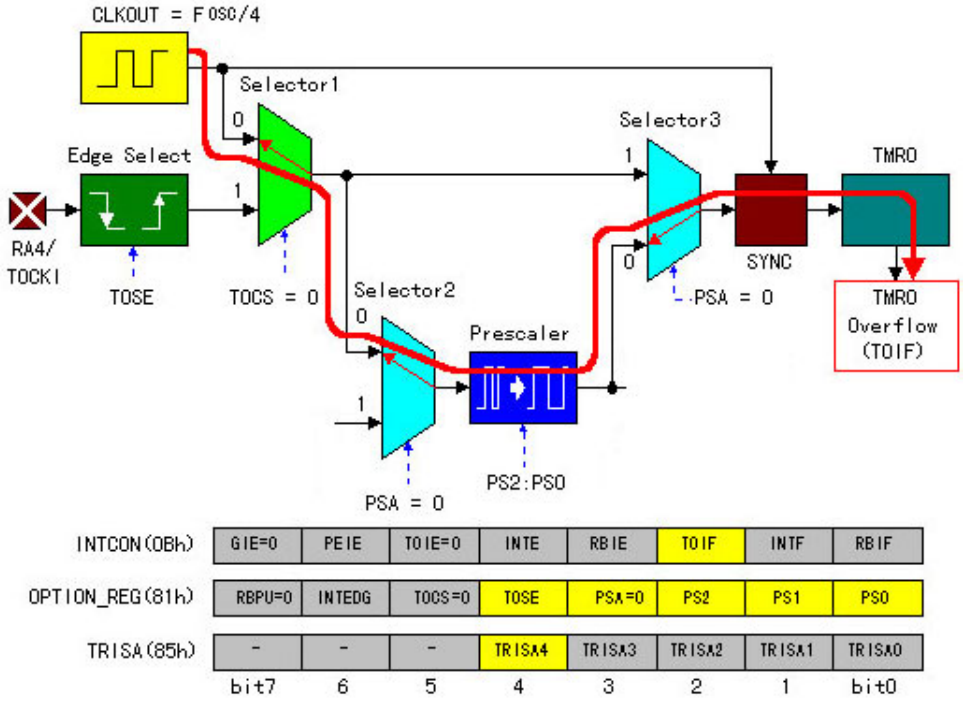


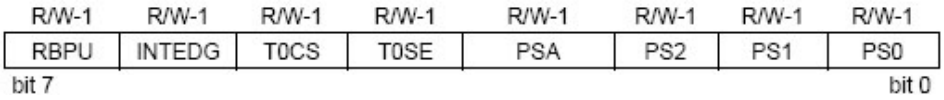
Figura 28: Diagrama de blocos do timer 0

A figura 28 mostra o diagrama de blocos para o *timer 0*. A linha a vermelho, descreve deste a entrada até ao *timer* o modo de selecção do sinal (relógio) de funcionamento. A selecção do sinal à entrada é feita pelo bit **TOCS** do registo OPTION_REG (ver página 38), sendo que este é o *clock* interno do PIC, 5Mhz, (1 ciclo de instrução de relógio do micro). A selecção efectuada usa ainda um pré-divisor ou contador programável (*prescaler*) de 8 bits, este permite dividir o sinal de entrada por um máximo de 256 (sinal de saída com pré divisão máxima = 19,5Khz). O número de contagens do pré-divisor é determinado pelos bits PS0, PS1, PS2 (bits 0, 1, & 2) do registo OPTION_REG. Finalmente o sinal de relógio chega ao módulo *timer 0*, este é um contador binário de 8 bits podendo contar até 256. Durante a contagem, quando é detectada a passagem de 255(0xFF) a 0(0x00), ou seja o *overflow*, é activado um sinal através de uma *flag*, (bit 2 do registo INTCON - **TOIF**, página 39). Esta *flag* deve ser reposta a zero por programação, de maneira a permitir a sinalização de novo *overflow*.

O código *Assembly* seguinte mostra uma possível programação do *timer 0* por *polling*, para uma contagem a cada 10 ms, na activação a '1' ou a '0' do registo *switch*. De notar que o valor carregado no registo **TMR0** é o valor de partida de contagem do módulo *timer 0*, ou seja uma contagem crescente até 256.

```
;---------------- TRM0 - InitCfg -----------------------------------;
                  Bank1                     ; DATA memory (RAM) Bank1
                  clrf   OPTION_REG         ; clear OPTION_REG
                  bsf    OPTION_REG,PS2     ;
                  bsf    OPTION_REG,PS1     ; TMR0 prescaler rate
                  bsf    OPTION_REG,PS0     ; 1:256
                                            ;
                  Bank0                     ; DATA memory (RAM) Bank0
                  movlw  60                 ;
                  movwf  TMR0               ; Set Hardware timer0 Value
                                            ;
                  clrf   switch             ; switch take val => 1/0
                                            ;
loop              btfss  INTCON,T0IF        ; while(TOIF == 0) {}
                  goto   $-1                ;
                  bcf    INTCON,T0IF        ; "reset flag" T0IF = 0;
                  movlw  60                 ;
                  movwf  TMR0               ; Set Hardware timer0 Value
                                            ;
                  incf   switch,w           ;
                  andlw  0x01               ; Toogle
                  movwf  switch             ;
                                            ;
                  goto   loop               ; repeat again
```

3.2 Registo OPTION_REG



bit 7 $\overline{RBPU}$: Activação dos *pull-ups* em PORTB .

1 = Desactiva os *pull-ups* em PORTB.
0 = Activa os *pull-ups* em PORTB (individualmente em cada latch).

bit 6 **INTEDG**: Selecção da transição de interrupção.

1 = Transição de '0' para '1' do pino RB0/INT.
0 = Transição de '1' para '0' do pino RB0/INT.

bit 5 **T0CS**: Bit de selecção da fonte de clock do TIMER 0.

1 = Através do pino RA4/T0CKL.
0 = Clock de ciclo das instruções do PIC (CLKO).

bit 4 **T0SE**: Selecção da transição de incremento do TMR0.

1 = Incremento na transição de '1' para '0' do pino RB4/T0CKL.
0 = Incremento na transição de '0' para '1' do pino RB4/T0CKL.

bit 3 **PSA**: Atribuição do *prescaler*.

1 = *Prescaler* atribuído ao *wachdogtimer* (WDT).
0 = *Prescaler* atribuído ao *timer 0*.

bit 2-0 **TMR0IF**: Bits de selecção do valor da taxa do *prescaler*.

Valor dos bits	Taxa do TMR0	Taxa do WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

3.3 Registo INTCON

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

O INTCON contém vários bits que inibem e desinibem as respectivas fontes de interrupção:

bit 7 **GIE**: Activação global das interrupções do PIC.

1 = Activa todas as interrupções.
0 = Desactiva todas as interrupções.

bit 6 **PEIE**: Activação das interrupções dos periféricos.

1 = Activa todas as interrupções dos periféricos.
0 = Desactiva todas as interrupções dos periféricos.

bit 5 **T0IE**: Activação da interrupção de *overflow* do *timer 0*.

1 = Activa a interrupção do *timer 0*.
0 = Desactiva a interrupção do *timer 0*.

bit 4 **INTE**: Activação da linha de interrupção *RB0/INT*.

1 = Activa *RB0/INT*.
0 = Desactiva *RB0/INT*.

bit 3 **RBIE**: Mudança de interrupção através do Porto RB.

1 = Activa a mudança de interrupção no Porto RB.
0 = Desactiva a mudança de interrupção no Porto RB.

bit 2 **T0IF**: Flag de *overflow* do *timer 0*.

1 = Indicação de *overflow* do *timer 0*, (reposição a 0 por programação).
0 = Ainda não acorreu *overflow* do *timer 0*.

bit 1 **INTF**: Flag de interrupção da linha *RB0/INT*.

1 = Ocorreu uma interrupção através da linha *RB0/INT* (reposição a 0 por programação).
0 = Não ocorreu qualquer interrupção na linha *RB0/INT*.

bit 0 **RBIF**: Flag de interrupção, alteração no porto RB.

1 = Quando pelo menos um dos portos RB7:RB4 mudam de estado
(reposição a 0 por programação).
0 = Nenhum dos pinos RB7:RB4 mudou de estado.

4 Trabalho a realizar

1. Voltar ao trabalho trabalho prático 4 (ponto 3, relógio de 60s com botão *START*), substituir a contagem de tempo efectuada pela rotina *Delay* no refrescamento dos *displays* de 7 segmentos, pelo *timer 0*.
   Perguntas:

   (a) Notou alguma diferença no funcionamento do programa, com a inclusão do *timer 0*?

   (b) Que vantagens existem na utilização de *timers* em virtude de rotinas *Delay*?

   (c) Dê exemplos do dia à dia, em que o uso de *timers* ou contadores é relevante?

• Elementos de Apoio

- Manual teórico da disciplina.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 3. - *I/O Ports*, página 29.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 5. - *Timer 0 Module*, página 47.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 13 - *Instrution Set Summary*, página 135.

TP6 - Semáforo rodoviário (automóveis/peões)

Tema a desenvolver	Duração
Implementação de uma máquina de estados	2 aulas

1 Resumo

Construção de uma máquina de estados que implemente um semáforo para automóveis e peões.

2 Objectivos

- Aprofundar os conhecimentos sobre timers.
- Implementação de uma máquina de estados por software.

3 Descrição

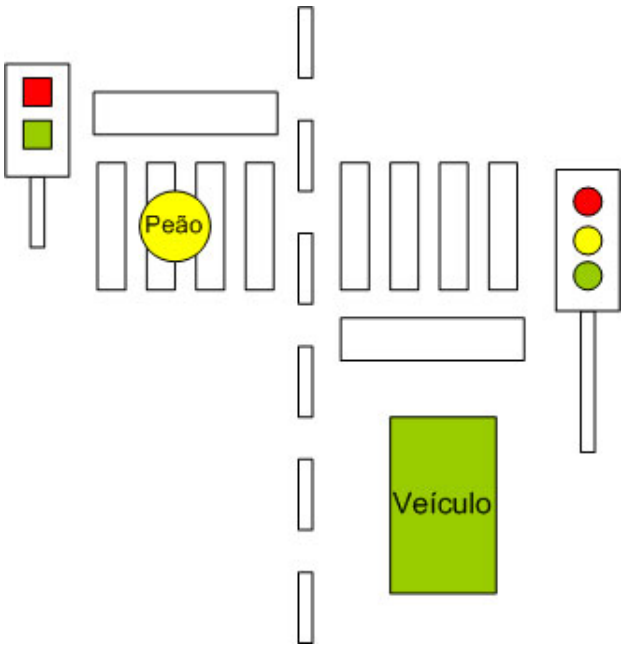


Figura 29: Semáforo rodoviário (automóveis/peões)

3.1 Diagrama de estados - Semáforo rodoviário

A figura 30 é um exemplo de um diagrama de estados, que mostra a interacção entre os vários estados propostos no trabalho prático, este exemplifica a maneira como são efectuadas as transições entre estados ao longo do tempo. De notar que os tempos **t1** a **t5** representam os tempos em que os estados estão em execução. A tabela 3 descreve a activação (ON) dos *leds* durante a execução dos estados.

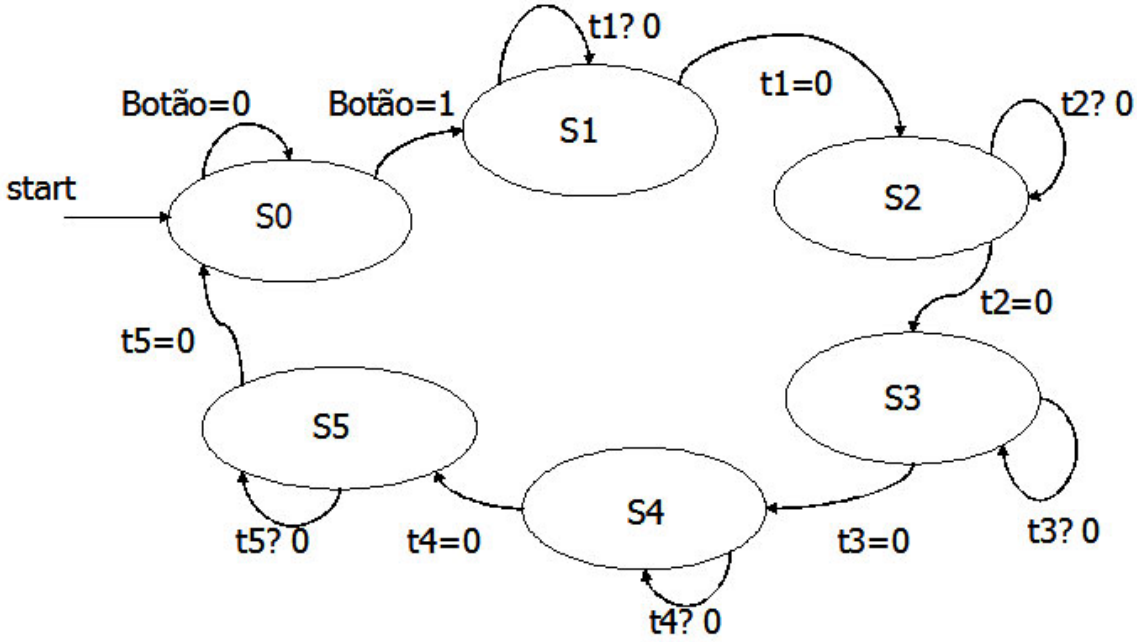


Figura 30: Diagrama de estados do trabalho prático

Estados	vermelho_auto	amarelo_auto	verde_auto	vermelho_peões	verde_peões
S0			on	on	
S1		on		on	
S2	on			on	
S3	on				on
S4	on				intermitente
S5	on			on	

Tabela 3: Descrição do estado dos *leds*

4 Trabalho a realizar

1. Dispondo de 5 *leds* (2 vermelhos, 2 verdes e 1 amarelo), implementar na placa branca dois semáforos, um para automóveis e outro para peões, ligados respectivamente em RB1, RB2, RB3 e RB4,RB5 (ver figura 31).

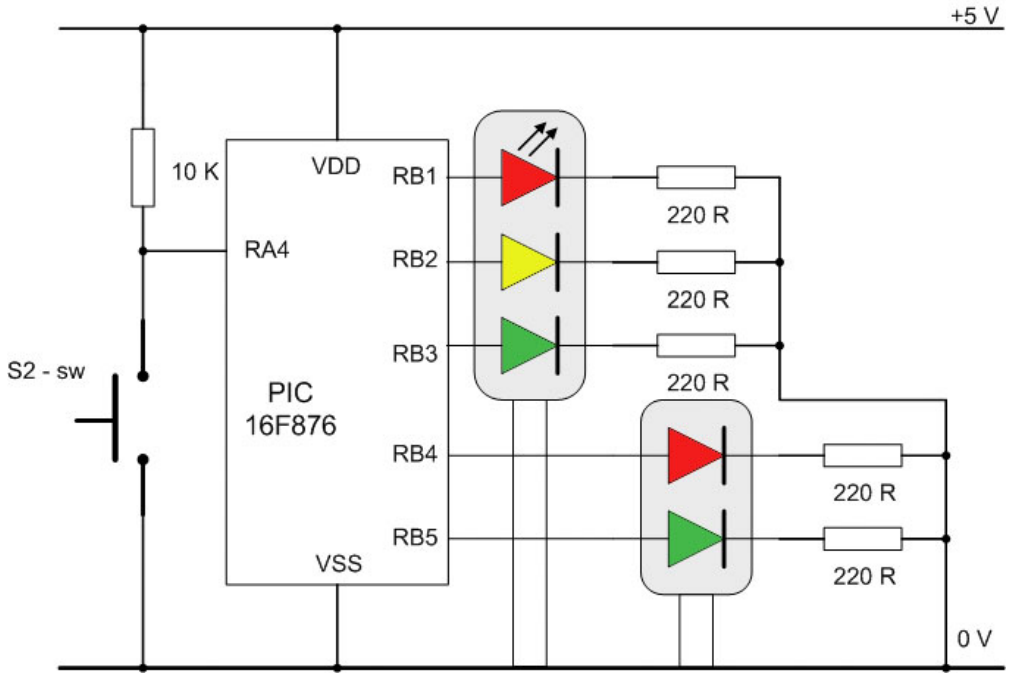


Figura 31: Diagrama do circuito, semáforo rodoviário

2. Desenvolvimento de um programa em assembly para controlo dos semáforos, de acordo com o diagrama de estados da figura 30, que efectue a contagem de tempo na transição dos estados, com auxílio do *timer 0*. A tabela 4, define o tempo mínimo que deve ser respeitado entre cada transição de estados.

estados	botão pressão	**t(segundos)**				
S0	activo ➠					
S1		1,5				
S2			1			
S3				4		
S4					3	
S5						1,5

Tabela 4: Tempo minímo entre estados

- **Elementos de Apoio**

  - Manual teórico da disciplina.
  - DataSheet PIC16F876(DS30292C) Microchip, capítulo 3. - *I/O Ports*, página 29.
  - DataSheet PIC16F876(DS30292C) Microchip, capítulo 5. - *Timer 0 Module*, página 47.
  - DataSheet PIC16F876(DS30292C) Microchip, capítulo 13 - *Instrution Set Summary*, página 135.

TP7 - Relógio 60 segundos, versão III



Tema a desenvolver	Duração
Programação com interrupções (Interrupt driven I/O)	3 aulas

1 Resumo

Pretende-se com este trabalho prático, introduzir o conceito da transferência de informação I/O por interrupção, ilustrado-o com uma solução do trabalho 1 (*led* controlado por botão de pressão) por interrupção.

No final do trabalho, utilizaremos o exemplo desenvolvido nos trabalhos 4 e 5 (relógio de 60s), para gerar interrupções através do *timer 0* e da linha INT0/RB0 do PIC.

2 Objectivos

- Compreensão da rotina de atendimento à interrupção **RSI** (**R**otina de **S**erviço de atendimento à **I**nterrupção) descrita em exemplo;
- Construção de rotinas de atendimento à interrupção.
- Geração de interrupções através da linha de interrupção RB0/INT do PIC.
- Geração de interrupções através da flag de *overflow* TOIF do *timer 0* do PIC, para refrescamento dos *displays* de 7 segmentos, e decremento da variável contadora do relógio.

3 Descrição

O PIC16F876 tem a possibilidade de contar com um sistema de interrupções. Este sistema, consiste num mecanismo através do qual um acontecimento interno ou externo, (assíncrono relativamente ao programa), pode interromper a execução, do programa em curso. É automaticamente produzido um salto para a rotina de atendimento (**RSI**), onde é executado o código que processa o evento gerador da interrupção, uma vez terminada a **RSI**, a execução do programa interrompido é retomada.

Existem 13 fontes de interrupção no PIC16F876, vejamos algumas:

- External RB0/INT pin, **INTF**.
- TMR0 timer overflow, **T0IF**.
- PORTB<7:4> interrupt-on-change, **RBIF**.

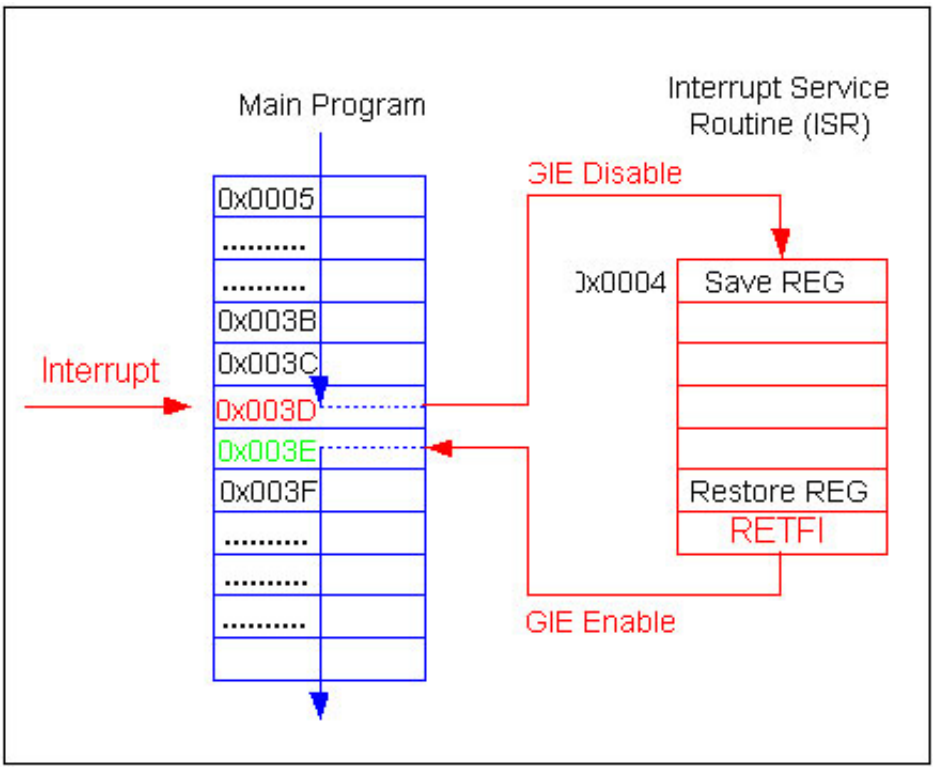


Figura 32: Exemplo de uma interrupção

A figura 32, representa o esquema simples de uma interrupção genérica. Antes da **RSI** propriamente dita, é feito automáticamente o *reset* do bit GIE, (bit 7 do registo INTCON), ou seja a desactivação de todas as interrupções durante o atendimento. O endereço de retorno é guardado na stack, sendo atribuído ao PC (program counter) o endereço 0x0004 (o que equivale a um salto para o endereço 4 da memória de programa).

É dever do programador, salvaguardar o contexto dos seguintes registos, W(reg), STATUS e PCLATH. Uma vez dentro da RSI, a fonte de interrupção pode ser determinada por *polling* aos bits de interrupção; das fontes de interrupção utilizadas. Os mesmos devem ser repostos a zero antes da terminação da **RSI**, evitando-se desse modo a geração (não desejada) de uma nova interrupção.

activar as interrupções globais, evitando assim interrupções recursivas.

Para melhor compreender o conceito, mostra-se em exemplo da rotina de atendimento à interrupção para o trabalho prático 1, ver página 48.

3.1 *Led* pisca-pisca com botão ON/OFF por interrupção

Nos trechos de código *assembly* seguintes, mostra-se em primeiro lugar o programa principal, inicialmente com a configuração dos registos TRISB e INTCON, cuja descrição pode ser encontrada nas páginas 16 e 39 respectivamente. De seguida o programa entra num ciclo infinito forçando a saída RB0 ao estado ON durante 0,5s e ao estado OFF durante 1s.

Em segundo lugar temos a rotina de atendimento à interrupção (RSI), activada sempre que a linha RB0/INT é activa a zero. A rotina de atendimento implementa o *toggle* (activação ou desactivação de um bit), de um registo denominado de start_stop. Este registo é testado continuamente no ciclo infinito descrito em cima.

```
                    ORG    0x0000          ; Processor reset vector
                    clrf   PCLATH          ; Ensure page bits are cleared
                    goto   main            ; Go to beginning of program
                    ORG    0x0004          ; Processor Interrupt Vector
                    goto   interrupt_start ; Go to ISR

;-----------------------------------------------------------------
;                   Rotina Main
;-----------------------------------------------------------------
                    ORG    0x0005
main                                      ;Aqui começa o programa principal

;----------------- I/O - InitCfg -----------------------------------;
                    Bank1                 ; DATA memory (RAM) Bank1
                    movlw  b'00000010'    ; RB1           output
                    movwf  TRISB          ; RB[7:2 & 0]   inputs

;----------------- Interrupts enable bits --------------------------;
                    Bank0                 ; DATA memory (RAM) Bank0
                    bsf    INTCON,GIE     ; Global Int. Enable bit
                    bsf    INTCON,INTE    ; External Int. enable bit

;-----------------------------------------------------------------;
                    clrf   PORTB          ; Inicialização PORTB.
                    clrf   start_stop     ;
while_1                                   ; while(1) {
                    btfsc  start_stop,0   ; while (start_stop == 1) {}
                    goto   $-1            ;

                    bsf    PORTB,RB1      ; Set RB1 ON
                    movlw  25             ;
                    call   Delay          ; delay( 0,25 seg )
                    bcf    PORTB,RB1      ; Set RB1 OFF
                    movlw  25             ;
                    call   Delay          ; delay( 0,25 seg )

                    goto   while_1        ; }
```

```
;************************* ISR ***********************************
;*****************************************************************
;*****************************************************************
interrupt_start
                    movwf   w_save        ; salvaguarda do W reg.
                    swapf   STATUS,W      ;
                    clrf    STATUS        ; DATA memory (RAM) Bank0
                    movwf   status_save   ; salvaguarda do STATUS reg.
                    movf    PCLATH,W      ;
                    movwf   pclath_save   ; salvaguarda do PCLATH reg.
                    clrf    PCLATH        ; página 0, de memória de
                                          ; programa
                    bcf     INTCON,INTF   ; reset, do bit INTF
                    incf    start_stop,w  ;
                    andlw   0x01          ;
                    movwf   start_stop    ; toggle do start_stop (0/1)
                                          ;
interrupt_end       movf    pclath_save,w; restauro do PCLATH reg.
                    movwf   PCLATH        ;
                    swapf   status_save,w; restauro da DATA memory (RAM)
                                          ; Bank? inicial
                    movwf   STATUS        ;
                    swapf   w_save,f      ; restauro do W reg.
                    swapf   w_save,w      ;

                    retfie                ; retorno ao prog. principal
```

4 Trabalho a realizar

1. Modificar o programa (contador de 60s, versão II) implementado em TP5, este deve agora implementar o START / STOP do relógio de 60 segundos, em qualquer altura da contagem decrescente. O diagrama do circuito pretendido, é mostrado na figura 33.

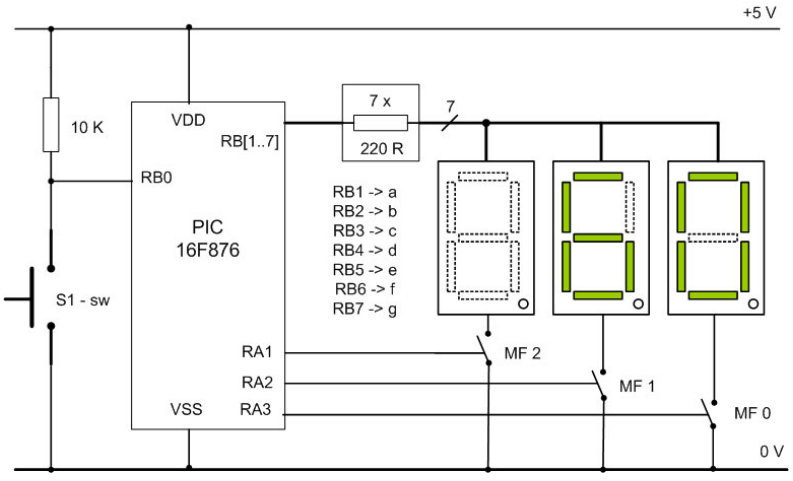


Figura 33: Diagrama do circuito, interrupção do RBO/INT

2. Configurar o *timer 0* de modo a que este gere interrupções por *overflow*, construir a rotina **RSI** que ao ser activada pelo **TOIF** (bit 2 do registo INTCON), sinalizando o *overflow* do *timer 0* do PIC. Esta deve implementar o refrescamento dos *displays* de 7 segmentos a cada 10ms e decremento da variável contadora a cada segundo.

3. Finalmente, acrescentar mais uma interrupção à rotina anterior, desta vez gerada pela linha INT0/RB0 do PIC, de modo a implementar o START/STOP do relógio de de 60 segundos.
   Nota: A **RSI** anterior, deve ser capaz de distinguir qual o interrupção gerada, *timer 0* ou RB0/INT, e agir em conformidade como a interrupção gerada.

• Elementos de Apoio

- Manual teórico da disciplina.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 3. - *I/O Ports*, página 29.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 5. - *Timer 0 Module*, página 47.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 12. - *Special Features Of Tje Cpu*, secção 12.10 - *Interrupts*, página 129.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 13 - *Instrution Set Summary*, página 135.

NOTAS:

TP8 - USART do PIC



Tema a desenvolver	Duração
Comunicação série assíncrona (USART)	2 aulas

1 Resumo

Neste trabalho prático, será explorada a USART (Universal Synchronous Asynchronous Receiver Transmitter) do PIC16F876, no modo assíncrono *full duplex*, na comunicação série entre o PIC e o computador do laboratório, através de um programa que implementa um terminal série.

2 Objectivos

- Compreensão dos passos de programação da USART, para recepção e transmissão de dados.
- Elaboração de rotinas para transmissão e recepção de um carácter.
- Elaboração de uma rotina para transmissão de um *array* de caracteres (strings).

3 Descrição

A USART, é uma das duas portas de comunicação integradas no PIC16F876. Este circuito permite a configuração para comunicação síncrono ou assíncrono, embora neste trabalho apenas estejamos interessados no modo de comunicação assíncrona.

Na comunicação com a USART cada bloco de dados é composto por 8 bits (contudo pode ser utilizado um bit adicional para implementar um esquema de detecção de erros, de comunicação por verificação de paridade).

O funcionamento da USART, é definido através da programação de dois registos de controlo, TXSTA (registo de controlo/estado de transmissão) e RCSTA (registo de controlo/estado de recepção), mais detalhes ver páginas 55 e 56 respectivamente.

3.1 Configuração dos pinos de transmissão/recepção

No modo assíncrono, usa-se o pino (17) RC6 / TX / CK para transmissão de dados, sendo o pino (18) RC7 / RX / DT usado para recepção de dados. Para a USART funcionar correctamente, a entrada (linha RX), e a saída (linha TX) têm de ser definidos no registo TRISC, (ver configuração de um porto página 16). Assim, o bit 6 do porto C, é definido em modo saída valor '0' e o bit 7 é definido em modo entrada valor '1'. Usando, RC6 e RC7 como porto para a USART, o bit SPEN do registo RCSTA, deve ser activado a '1'.

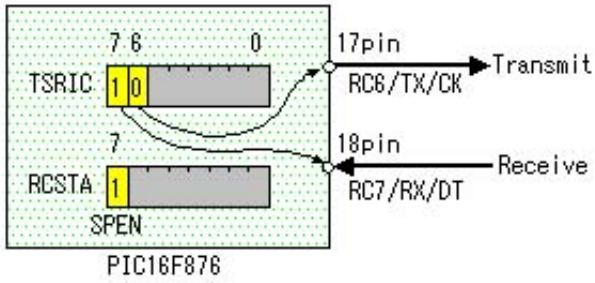


Figura 34: Atribuição do porto série

3.2 Taxa de transmissão da USART

A taxa de transmissão da USART, é controlada pelo BRG (Baud Rate Generator). Esta é definida pelo valor a colocar no registo SPBRG, este registo controla o período do sinal de saída de um *free run timer* de 8 bits. Em modo assíncrono, o bit BRGH no registo TXSTA, é usado para controlar a taxa de transmissão entre *low speed* e *high speed*, usar *high speed* na maioria dos casos é vantajoso, uma vez que reduz a taxa de erro.

As taxas de transmissão disponíveis, para o PIC com uma FOSC de 20MHz estão descritas na tabela 5.

Taxa de transmissão (BAUD RATE)	Valor (decimal) a colocar em SPBRG
9600	129
19200	64
28800	42
33600	36
57600	20
115200	10

Tabela 5: Taxa de transmissão em modo assíncrono (BRGH = 1)

3.3 Operação de transmissão

É no registo TXREG que são carregados os dados a transmitir, estes são depois enviados de forma automática para o registo interno TSR e posteriormente transmitidos pelo porto TX. A transmissão de dados é activada pelo bit TXEN do registo TXSTA. O bit menos significativo (LSB, Least Significant Bit), é transferido em primeiro lugar.

Uma vez transmitido o conteúdo do registo TXREG para TSR, o **TXIF** (bit indicador do estado do buffer em PIR1) sinaliza que o registo de transmissão da USART está vazio, tomando o valor '1'. Os dados podem ser continuamente transmitidos, se primeiro fôr detectada a sinalização de **TXIF**, e de seguida fôr efectuada uma escrita no registo TXREG. O **TXIF** não pode ser apagado por programação: é automáticamente forçado a '0' quando os dados são movidos para TXREG;

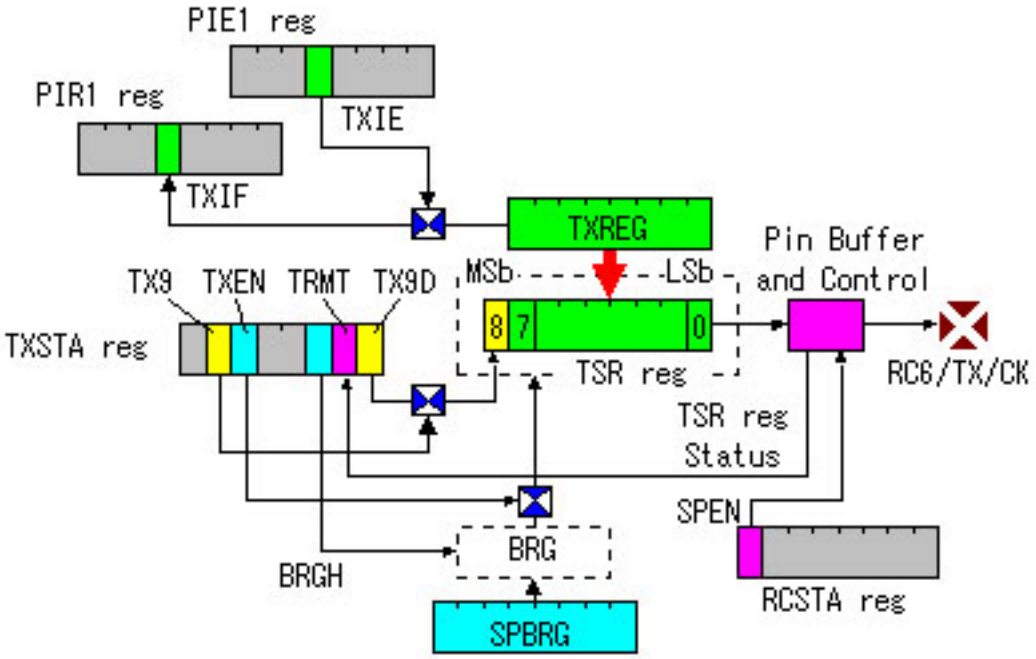


Figura 35: Operação de transmissão assíncrona

Passos a seguir na configuração de uma Transmissão Assíncrona.

1. Inicializar o registo SPBRG com a taxa de transferência apropriada (ver tabela 5);
2. Activação da porta série em modo assíncrono, pondo a '0' o bit SYNC do registo TXSTA e a '1' o bit SPEN do registo RCSTA;
3. Por forma activar a transmissão o bit TXEN (registo TXSTA) é activo a '1'. Nessa altura o registo TXREG está em condições de ser escrito e o bit TXIF do registo PIR1 vem a '1';
4. Enviar os dados para o TXREG e começar a transmissão.

3.4 Operação de recepção

Os dados recebidos pela porta RX são recolhidos no registo RSR, através do circuito de recolha de dados (Data Recovery). Os dados são recolhidos pelo registo RSR, de acordo com a taxa de transmissão especificada pelo registo SPBRG e pelo bit BRGH do registo TXSTA (ver página 55).

Aquando da detecção do stop bit, o conteúdo do registo RSR é transferido para o registo RCREG. O bit RCIF do registo PIR1, toma o valor '1' quando os dados são escritos no registo RCREG, (sinalizando a recepção de nova informação). Os dados podem ser continuamente recebidos, se primeiro fôr detectada a sinalização e de seguida efectuar-se uma leitura do registo RCREG. O RCIF não pode ser apagado por programação, este é '0' quando os dados são lidos de RCREG.

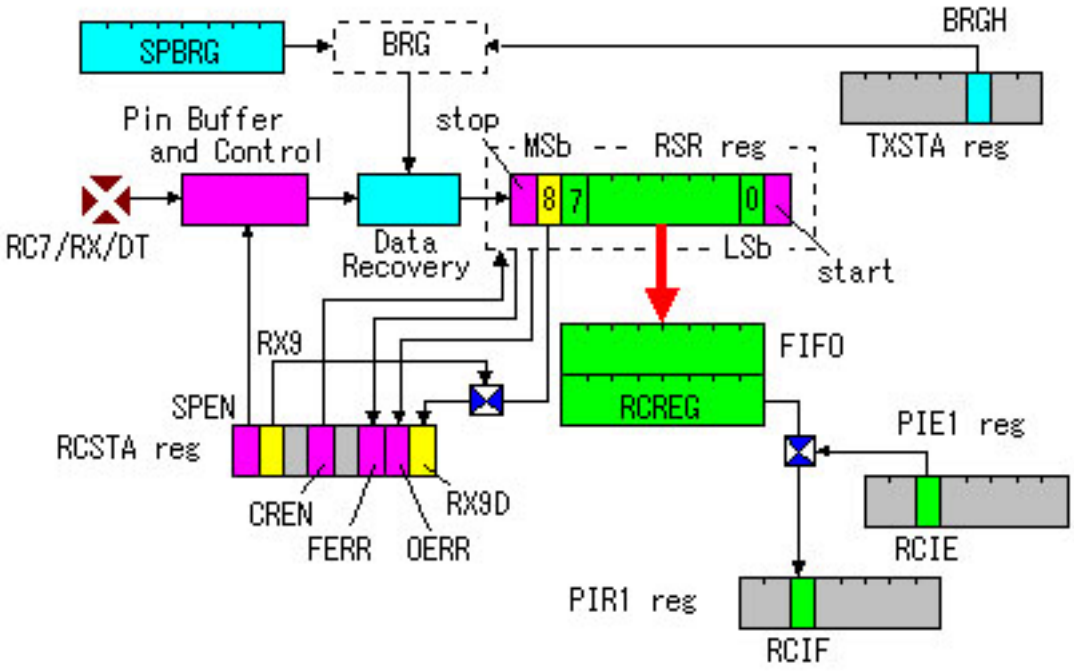


Figura 36: Operacão de recepcção assíncrona

Passos a seguir na configuração na Recepção Assíncrona.

1. Inicializar o registo SPBRG com a taxa de transmissão apropriada;
2. Activação da porta série em modo assíncrono, pondo a '0' o bit SYNC do registo TXSTA e a '1' o bit SPEN do registo RCSTA;
3. Permitir a recepção, activando o bit CREN do registo RCSTA;
4. Quando a recepção estiver terminada, a flag RCIF do registo PIR1 é activado;
5. Para ler os oito bits recebidos, deverá ser lido o registo RCREG.

3.5 Registo TXSTA

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC**: Selecção da origem do relógio.

Modo Assíncrono:
Dont´t care.
Modo Síncrono:
1 = Modo master (relógio gerado internamente por BRG).
0 = Modo slave (relógio gerado externamente).

bit 6 **TX9**: Transmissão com 9 bits.

1 = Activa a transmissão a 9 bits.
0 = Desactiva a transmissão a 9 bits.

bit 5 **TXEN**: Activa transmissão.

1 = Activa transmissão.
0 = Desactiva transmissão.

bit 4 **CREN**: Modo de comunicação da USART.

1 = Modo Assíncrono.
0 = Modo Síncrono.

bit 3 **Não Implementado**: leitura a ’0’.

bit 2 **BRGH**: Bit de selecção de HIGH Baud Rate.

Modo Assíncrono:
1 = HIGH SPEED.
0 = LOW SPEED.
Modo Síncrono:
Não usado neste modo

bit 1 **TRMT**: Transmit Shift Register, bit de estado.

1 = TSR vazio.
0 = TSR cheio.

bit 0 **TX9D**: Nono bit, na transmissão de dados, (usado para paridade).

3.6 Registo RCSTA

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN**: Activação da porta série do PIC.

1 = Porta série activada (configura RC7/RX/DT e RC6/TX/CK pinos como portos série) .
0 = Porta série desactivada.

bit 6 **RX9**: Recepção com 9 bits.

1 = Activa a recepção a 9 bits.
0 = Desactiva a recepção a 9 bits.

bit 5 **SREN**: Activa recepção única.

Modo Assíncrono:
Dont´t care.
Modo Síncrono - master:
1 = Activa recepção única.
0 = Desactiva recepção única. NOTA: Este bit é apagado, após uma recepção completa.
Modo Síncrono - slave:
Dont´t care.

bit 4 **CREN**: Activa a recepção contínua.

Modo Assíncrono:
1 = Activa recepção contínua.
0 = Desactiva recepção contínua.
Modo Síncrono:
1 = Activa recepção contínua, até bit CREN activo ser desactivado (CREN supra sucede SREN).
0 = Desactiva recepção contínua.

bit 3 **ADDEN**: Activa a detecção de endereço.

Modo Assíncrono 9-bit (RX9 = 1):
1 = Activa detecção de endereço, activa interrupção e carga do buffer de recepção, quando RSR<8> activo.
0 = Desactiva detecção de endereço, todos os bytes são recebidos, e o nono bit pode ser usado como bit de paridade.

bit 2 **FERR**: Indicação de erro de **frame** (leitura apenas).

1 = Indicação de **framing error**.
NOTA:
Pode ser actualizado lendo o registo RCREG e efectuando a recepção do próximo byte válido).
0 = Não ocorreu **framing error**.

bit 1 **OERR**: Indicação de erro do bit **overrun** (leitura apenas).

1 = Ocorreu um erro de **overrun** (reposição a 0 por reposição a zero de CREN).
0 = Não ocorreu erro de **overrun**.

bit 0 **RX9D**: Nono bit, na recepção de dados (usado para paridade).

4 Trabalho a realizar

1. Implementar um programa em *assembly* que permita a transmissão de 1 carácter, para o programa terminal do computador anfitrião (*host computer terminal*).
2. Implementar um programa em *assembly* que permita a recepção de 1 carácter, e posterior transmissão (através da rotina criada anteriormente) para o *host computer terminal*.
3. Implementar um programa em *assembly* que permita o envio de um *array* de caracteres, para o *host computer terminal*. Inicialmente o programa deve enviar para o terminal a string "Digite -> (p,i,e,x)?", e consoante o carácter seleccionado (premido), será enviada para o terminal a string correspondente:
   (a) Quando recebido o carácter 'p' é enviada a mensagem "PIC - Ola, Mundo";
   (b) Se recebido o carácter 'i' ou 'e' serão enviadas as strings respectivas ás mensagens ora em Inglês ora em Espanhol.
   (c) Quando recebido o carácter 'x', será enviada a mensagem "PIC - BYE BYE user". No entanto se um outro carácter for recebido, deverá ser enviada para o terminal a mensagem seguinte "Caracter Invalido".

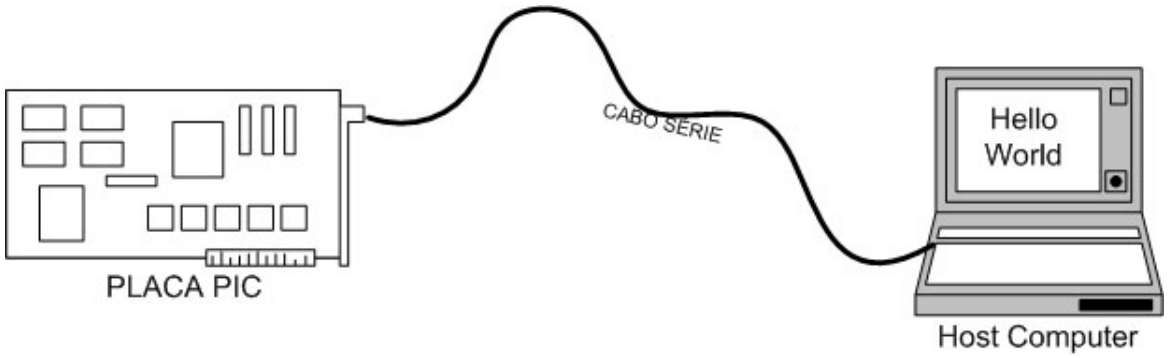


Figura 37: Ligação série entre o PIC e o Computador

• Elementos de Apoio

- Manual teórico da disciplina.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 10. - *Addressable Universal synchronous Asyncronous Receiver Transmiter (USART)*, página 95.
- DataSheet PIC16F876(DS30292C) Microchip, capítulo 13 - *Instrution Set Summary*, página 135.

NOTAS:

TP9 - Conversor A/D do PIC



Tema a desenvolver	Duração
Conversão Analógia -> Digital (ADC)	4 aulas

1 Resumo

Conversão A/D de valores de tensão (analógicos) provenientes de diferentes dispositivos. Interface através de sensor infravermelhos e potenciómetro com visualização de resultados em *displays* de 7 segmentos ou através do PC (terminal série).

2 Objectivos

- Introdução ao estudo do conversor A/D (ADC) do PIC.
- Implementação de um voltímetro digital básico.
- Medição de distâncias através de um sensor de infravermelhos.

3 Descrição

No processamento de sinais-analógicos há, muitas vezes, vantagem em os converter para sinais digitais, de modo a que possam se processados por um microprocessador. Os elemento que efectua a conversão de um sinal analógico num digital (um valor numa determinada voltagem) é designado por "conversor analógico digital"' ADC.

O PIC16F876 dispõe de uma ADC de 10 bits, o que significa que o resultado pode ser representado com uma resolução de 1/1024 gama máxima. É possível distinguir valores com cerca de 5mV de resolução aproximadamente (5/1024 = 0.004883), supondo que a gama máxima de valores a medir é de 5V, ou seja toda a gama entre 0V..5V.

3.1 Entrada Analógica

No PIC16F876, podemos seleccionar como entrada analógica, por programação 1 dos 5 canais disponíveis (entradas RA0 - RA5, excepto RA4). O registo ADCON1 configura as linhas RA0 - RA5 (execpção RA4) do PIC como entradas analógicas/digitais ou referência; a entrada seleccionada deve

ser programada como entrada afectando o bit correspondente no registo TRISA. Mais detalhes ver configuração de um porto com entradas analógicas, página 16.

À ligação da entrada analógica com o conversor A/D, dá-se o nome de canal *(channel)*, a selecção do canal a ser activado é feita pelo registo ADCON0 (bits CHS2-CHS0), ver página 63.

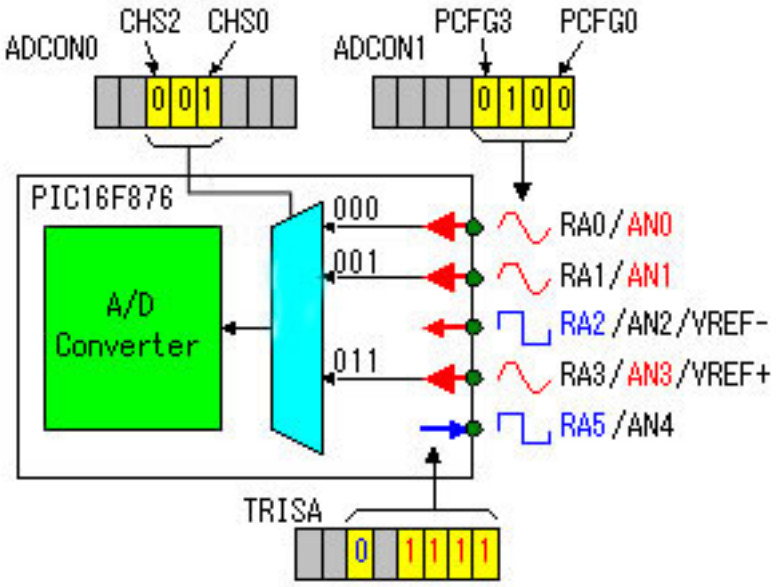


Figura 38: ADC, entradas analógicas

3.2 Resultado da conversão A/D

O par de registos ADRESH e ADRESL de 8 bits cada, forma o local onde são colocados os 10 bits do resultado da conversão, estes registos formam assim em conjunto um registo de 16 bits.

A ADC, implementa duas formas no modo como formata o resultado de 10 bits nos registos ADRESH e ADRESL, justificado este ora à esquerda ou à direita (restantes bits extra são carregados a '0'), ver figura 39. A selecção do formato é feita pelo bit ADFM do registro ADCON1, os bits , .

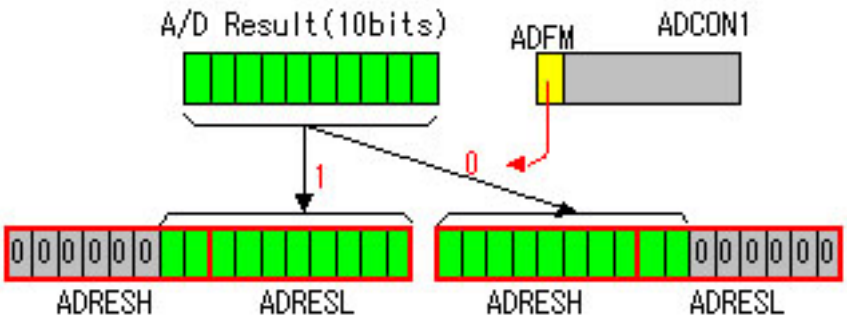


Figura 39: Formato do resultado de 10 bits

3.3 Mecanismo de aquisição da ADC

O condensador de amostragem (CHOLD) do módulo conversor A/D, é carregado pela tensão aplicada à entrada analógica. A tensão de CHOLD é convertida para valores digitais através do conversor A/D. Uma vez que o condensador não carrega instantaneamente, é necessário aguardar um certo tempo, designado por tempo de HOLD, após a selecção da entrada analógica pretendida nos bits CHS0-3 do ADCON0 (ver página 63).

Uma vez que este tempo não é implementado por hardware, é da responsabilidade do programador implementar no programa um tempo de espera de aproximadamente 20μs, entre a selecção da entrada analógica e o inicio da conversão A/D. Durante este tempo, o CHOLD está desligado da entrada analógica.

A conversão A/D é iniciada quando o bit GO do registo ADCON0 é activado a '1'. Quando a conversão acaba o bit GO é reposto a '0' por hardware. Existem duas maneiras de detectar o fim de conversão da ADC, a primeira é através de *polling* ao bit GO até este ser '0', a segunda mais elegante, é através de uma interrupção global. No caso de usar interrupções, o bit ADIE do registo PIE, assim como o bit do GIE do registo INTCON, devem estar activos a '1'. Quando o fim de conversão for detectado, o bit ADIF do registo PIR1 é activo '1' e a interrupção global ocorre.

A figura 61 mostra o circuito no qual o condensador CHOLD é carregado por VA.

CHOLD é carregado quando o interruptor (SS) é fechado. O díodo foi colocado por protecção no sentido de evitar situações de tensão superiores a VDD ou inferiores a VSS na entrada analógica.

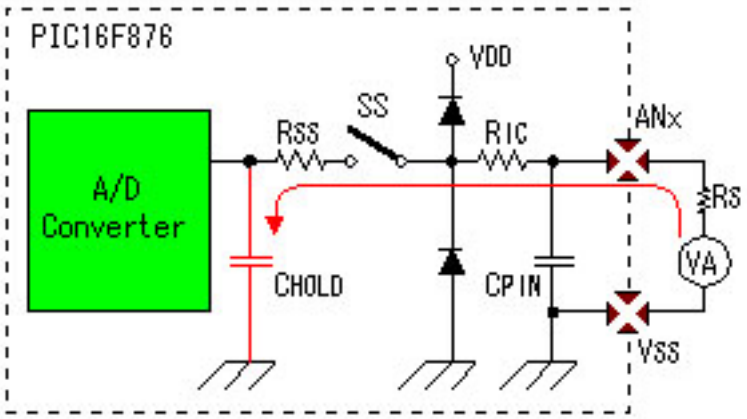


Figura 40: Mecanismo de aquisição da ADC

- Legenda:
  - CHOLD: Condensador de amostragem / retenção (120pF)
  - RSS: Resistência de amostragem (cerca de 7K-ohm a VDD=5V)
  - SS: Interruptor
  - RIC: Resistência de ligação (=<1K-ohm)

- CPIN: Condensador de entrada (5pF)
- ANx: Entrada Analógica
- RS: Impedância da entrada analógica (=<10K-ohm)
- VA: Fonte de tensão analógica

3.4 Etapas na programação da ADC (*polling*)

1. Configuração do conversor A/D (Processo inicial)
   - Configuração das entradas analógicas / tensão de referência / I/0 digital ADCON1/PCFG0-3.
   - Selecção do clock de conversão da ADC, ADCON0/ADCS0-1
   - Ligar o conversor A/D, ADCON0/ADON
2. Configuração do canal de entrada
   - Selecção do canal de entrada, ADCON0/CHS0-2
3. Tempo guarda de carga do condensador CHOLD
   - Aguardar até o CHOLD esteja completamente carregado, aprox. 20μs
4. Início da conversão
   - Activar o bit GO, ADCON0/GO = '1'
5. Aguardar o fim de conversão da ADC
   - Fazer *polling* ao bit GO até que este seja '0'
6. Leitura do resultado da conversão A/D
   - Ler o resultado no par de registos, ADRESH,ADRESL
7. Para a próxima conversão, ir para item 1 ou 2 como desejado

3.5 Registo ADCON0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1,ADCS0**: Bits de selecção do clock do conversor A/D.

00 = FOSC/2.
01 = FOSC/8.
10 = FOSC/32.
11 = FRC (clock derivado do oscilador interno da ADC).

bit 5-3 **CH2**: Bits de selecção do canal analógico .

000 = canal 0, (RA0/AN0).
001 = canal 1, (RA1/AN1).
010 = canal 2, (RA2/AN2).
011 = canal 3, (RA4/AN3).
100 = canal 4, (RA5/AN4).

bit 2 **GO/**$\overline{DONE}$: Bit de estado da conversão A/D.

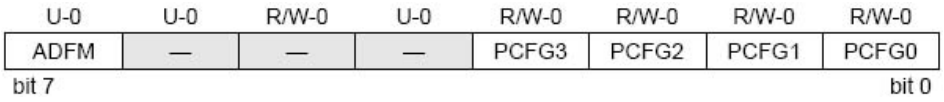
1 = Conversão A/D a decorrer (activar a '1' inicia a conversão A/D).
0 = conversão A/D a parada (este bit é automaticamente *reset* pelo PIC quando a conversão A/D
é completada).

bit 1 **Não Implementado**: leitura a '0'.

bit 0 **ADON**: Conversor A/D ligado.

1 = Conversor A/D ligado e em actividade.
0 = Conversor A/D desligado, não consome corrente.

3.6 Registo ADCON1



bit 7 **ADFM**: Selecção do formato da conversão A/D.

0 = Justificado à direita. 6 bits mais significativos do ADRESH colocados a '0'.
1 = Justificado à esquerda. 6 bits menos significativos do ADRESL colocados a '0'.

bit 6 **ADCS2**: Bit de selecção do *clock* de conversão.

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock de conversão
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	Frc (clock derivado do oscilador RC, interno da ADC)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	Frc (clock derivado do oscilador RC, interno da ADC)

bit 5-4 **Não Implementado**: leitura a '0'.

bit 3-0 **PCFG3:PCFG0**: Bits de controlo das entradas analógicas do PIC.

PCFG <3:0>	RA5	RA3	RA2	RA1	RA0
0000	A	A	A	A	A
0001	A	Vref+	A	A	A
0010	D	A	A	A	A
0011	D	Vref+	A	A	A
0100	D	A	D	A	A
0101	D	Vref+	A	A	A
011x	D	D	D	D	D
1000	A	Vref+	Vref-	A	A
1001	A	A	A	A	A
1010	A	Vref+	A	A	A
1011	A	Vref+	Vref-	A	A
1100	D	Vref+	Vref-	A	A
1101	D	Vref+	Vref-	A	A
1110	D	D	D	D	A
1111	D	Vref+	Vref-	D	A

A = entrada analógica, D = entrada ou saída, digital, Vref$\pm$ = tensão de referência

4 Trabalho a realizar

1. Construção de um voltímetro digital simples. Este voltímetro faz a leitura de uma tensão eléctrica entre 0V e 5V apresentando o resultado da conversão A/D, nos *displays* de 7 segmentos. O valor convertido é afixado com um casa decimal (0.0 a 5.0V).
   De modo a implementar esta funcionalidade, deve-se programar a ADC com os seguintes parâmetros: entrada analógica RA0, selecção do clock do conversor A/D = FOSC/32, resolução de 8 bits na saída. NOTA: Uma vez que a escala de valores retornados da ADC é [0..255], deve implementar uma mudança de escala para valores entre [0..50].
   Implementação de uma rotina de hexadecimal ☞ decimal, de modo a poder afixar nos *displays* de 7 segmentos o valor convertido.

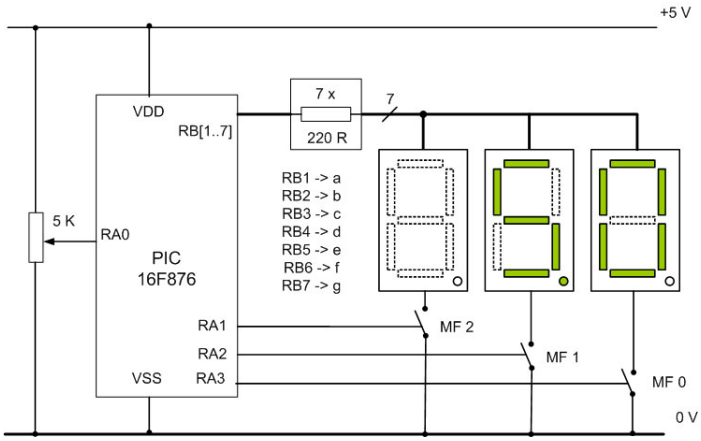


Figura 41: Diagrama do circuito do voltímetro digital, visualização com *displays* 7 segmentos

2. Finda a parte 1 do trabalho prático, o passo seguinte é alterar o programa anterior, de modo a que seja possível o envio dos valores de tensão através da porta série, para o PC.

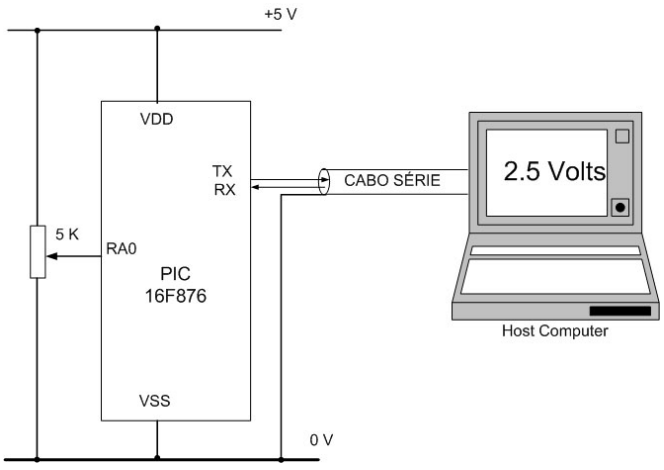


Figura 42: Diagrama do circuito do voltímetro digital, visualização terminal série

3. Neste ponto do trabalho, pretende-se medir distâncias com o auxílio a um sensor de infravermelhos, com visualização da distância nos *displays* de 7 segmentos. O sistema de medição deve efectuar medições entre [10..40cm] com uma resolução de 1 cm. Fora desta gama deve indicar, através das mensagens, **OFF** e **End** que se encontra entre 0..10cm e entre 40..∞, respectivamente.

   Sugestão: Implementar uma rotina de cálculo da distancia, tendo em conta os valores de calibração do sensor fornecidos.

   NOTA: Os valores de calibração do sensor estão disponíveis para consulta no *site* da disciplina.

Apendix A

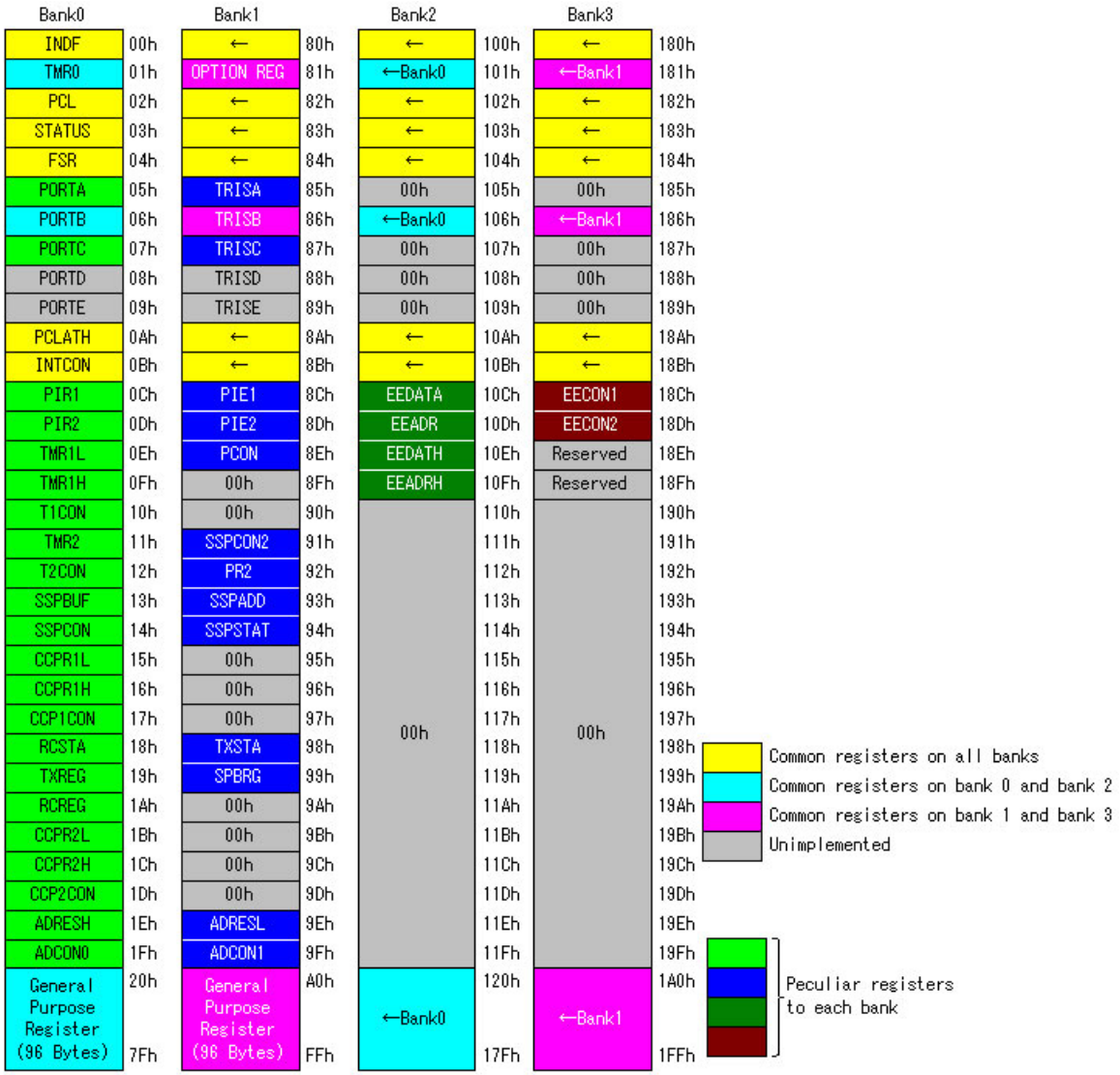


Figura 43: Mapa de memória RAM, registos do PIC e registos de uso geral

Apendix B

```
;    Título :
;    MPLAB IDE programa de demonstração
;
;    Resumo:
;    Led pisca-pisca (DUTTY CYCLE variável)
;
;    ____________
;   |            |
;   | PIC16F876  |
;   |            |
;   |            |                /-----\        RES
;   |            |->-- RB0 ------| LED |------\/\/\/---- 0V
;   |            |                \-----/
;   |____________|
;
;                           figura 1. esquema de ligação PIC.
;
;                    ON (T1)|   OFF (T2)   | ON (T1)
;                    _______                _______
;                   |       |              |       |
;    led    ________|       |______________|       |_____________________
;
;
;                     figura 2. diagrama temporal da saída.
;
;*********************************************************************
;                                                                    *
;    Nome ficheiro: exemplo.asm                                      *
;    Última alteração: 24/Junho/2005                                 *
;                                                                    *
;    Autor:          Prof. José Luís Azevedo                         *
;                                                                    *
;    Modificado por: José Miguel Oliveira Gaspar                     *
;                    Olímpia Rodrigues                               *
;                                                                    *
;    Universidade de Aveiro                                          *
;    Seminário EEI 2004/2005                                         *
;                                                                    *
;*********************************************************************
;                                                                    *
```

```
;    Ficheiro necessário:              16f876.lkr                  *
;                                                                   *
;*******************************************************************

        list    p=16f876                ; - directiva 'list', define
                                        ;   o processor a usar
        radix   decimal                 ; - directiva 'radix' especifica
                                        ;   definição das variáveis
        #include "p16f876.inc"          ; - directiva 'include' inclui
                                        ;   ficheiro adicional

        errorlevel -302                 ; Turn off banking message
                                        ; known tested (good) code
;-------------------------------------------------------------------
;                   Constantes
;-------------------------------------------------------------------
RA0                 EQU    0
RA1                 EQU    1
RA2                 EQU    2
RA3                 EQU    3
RA4                 EQU    4
RA5                 EQU    5
RB0                 EQU    0
RB1                 EQU    1
RB2                 EQU    2
RB3                 EQU    3
RB4                 EQU    4
RB5                 EQU    5
RB6                 EQU    6
RB7                 EQU    7
RC0                 EQU    0
RC1                 EQU    1
RC2                 EQU    2
RC3                 EQU    3
RC4                 EQU    4
RC5                 EQU    5
RC6                 EQU    6
RC7                 EQU    7
T1                  EQU    50           ; Gama [1..255] = [10ms..2,5s]
T2                  EQU    100          ;
;-------------------------------------------------------------------
;     Variáveis (REGISTOS DE USO GERAL) Bank0
;-------------------------------------------------------------------
group1              iDATA               ; dados inicializados
delay_mult          RES    1             ; Variaveis de controlo
delay_k50           RES    1             ; dos loops da rotina Delay
delay_k200          RES    1             ;

;-------------------------------------------------------------------
```

```
;                   Macros to select the register Banks
;-----------------------------------------------------------------------
Bank0               MACRO             ; Macro to select data RAM Bank 0
                    bcf  STATUS,RP0
                    bcf  STATUS,RP1
                    ENDM

Bank1               MACRO             ; Macro to select data RAM Bank 1
                    bsf  STATUS,RP0
                    bcf  STATUS,RP1
                    ENDM

Bank2               MACRO             ; Macro to select data RAM Bank 2
                    bcf    STATUS,RP0
                    bsf    STATUS,RP1
                    ENDM

Bank3               MACRO             ; Macro to select data RAM Bank 3
                    bsf    STATUS,RP0
                    bsf    STATUS,RP1
                    ENDM

;**********************************************************************
Reset_vector        ORG    0x0000       ; - directiva "ORG" origem do
                                        ;   programa "ORG 0x0000" vector
                                        ;   de reset
                    clrf   PCLATH       ; - instrução "clrf PCLATH" apaga
                                        ;   todos os bits do registo garante
                                        ;   que a memória do programa
                                        ;   é iniciada na página 0
                    goto   main         ; - instrução "goto  main" salta para
                                        ;   o endereço de memória do programa
                                        ;   principal
;-----------------------------------------------------------------------
;                   Rotina Main
;-----------------------------------------------------------------------
                    ORG    0x0005      ; Program Memory PAGE 0
main                                    ; Programa principal.
                    Bank1               ; DATA memory (RAM) Bank1
                    movlw  b'11111110'  ; RB[0]         output
                    movwf  TRISB        ; RB[7:1]       inputs
                                        ;
                    Bank0               ; DATA memory (RAM) Bank0
                    clrf   PORTB        ; Inicialização PORTB.
loop                                    ; while(1) {
                    bsf    PORTB,RB0    ; Set (ON) LED in RB0 ON

                    movlw  T1           ; W(reg) = 50
                    call   Delay        ; delay( 0,5 seg )
```

```
                    bcf    PORTB,RB0    ; Set (OFF) LED in RB0
                    movlw  T2           ; W(reg) = 100
                    call   Delay        ; delay( 1 seg )
                    goto   loop         ; }

;*****************************************************************


;************************* ROTINAS *********************************
;*****************************************************************
;*****************************************************************
;=================================================================;
; Delay                                                           ;
; Pode gerar delays entre 10 mS e 2,5 S                           ;
; O valor de entrada e' passado em W (1..255)                     ;
; Retorna em Bank0                                                ;
;                                                                 ;
; Notas: o loop base e' de 1 uS (T=200 nS <=> 4*1/20MHz)          ;
;                                                                 ;
;=================================================================;
Delay
                    movwf  delay_mult   ; Carrega o valor multiplo
Del_m10             movlw  50           ; de 10 mS
                    movwf  delay_k50    ; 50 => 10 mS
Del_50              movlw  200          ; 200 => 200 uS
                    movwf  delay_k200   ; Carrega o valor
Del_200             nop                 ; correspondente a 200 uS
                    nop
                    decfsz delay_k200,f ; Decrementa "delay_k200".
                                        ; O resultado é colocadao em
                                        ; "delay_k200"
                    goto   Del_200      ; while( delay_k200 > 0 )
                    decfsz delay_k50,f  ;
                    goto   Del_50       ; while( delay_k50 > 0 )
                    decfsz delay_mult,f ; Decrementa "delay_mult".
                                        ; O resultado é colocadao
                                        ; em "delay_mult"
                    goto   Del_m10      ; while( delay_mult > 0 )

                    return              ;

;*****************************************************************

                    END                 ; directive 'end of program'
```

```
; Trabalho nº 1
; Implementação de um led pisca-pisca, em RB1
; com botão on/off por INTERRUPÇÂO
;
; Objectivos:
; Introdução às Interrupções EXEMPLO
;
; Observações:
;
;*****************************************************************
;                                                                *
;   Nome ficheiro: trab1Int.asm                                  *
;   Data: 1/Junho/2005                                           *
;   Versão: 1.0                                                  *
;                                                                *
;   Autores:    José Miguel Gaspar                               *
;               Olímpia Rodrigues                                *
;   Orientador: Prof. José Luís Azevedo                          *
;                                                                *
;   Universidade de Aveiro                                       *
;   Seminário EEI 2004/2005                                      *
;                                                                *
;*****************************************************************
;                                                                *
;   Ficheiro necessário:     P16F876.LKR                         *
;                                                                *
;*****************************************************************

        list    p=16f876                ; list directive to define
        radix   decimal                 ; processor

        #include "p16f876.inc"          ; processor specific variable
                                        ; definitions
        errorlevel -302                 ; Turn off banking message
                                        ; known tested (good) code

;-----------------------------------------------------------------------------
;    Constantes
;-----------------------------------------------------------------------------

RA0                 EQU    0
RA1                 EQU    1
RA2                 EQU    2
RA3                 EQU    3
RA4                 EQU    4
RA5                 EQU    5

RB0                 EQU    0
RB1                 EQU    1
```

```
RB2                EQU    2
RB3                EQU    3
RB4                EQU    4
RB5                EQU    5
RB6                EQU    6
RB7                EQU    7

RC0                EQU    0
RC1                EQU    1
RC2                EQU    2
RC3                EQU    3
RC4                EQU    4
RC5                EQU    5
RC6                EQU    6
RC7                EQU    7

;-----------------------------------------------------------------------
;     Variáveis (REGISTOS DE USO GERAL)
;-----------------------------------------------------------------------
variaveis          UDATA  0x20
delay_mult         RES    1     ; Variaveis de controlo dos loops
delay_k50          RES    1     ; da rotina Delay
delay_k200         RES    1     ;

start_stop         RES    1

w_save             RES    1     ; SAVE W
status_save        RES    1     ; SAVE STATUS REG
pclath_save        RES    1     ; SAVE PCLATH REG

;-----------------------------------------------------------------------
;                  Macros to select the register Banks
;-----------------------------------------------------------------------

Bank0              MACRO            ; Macro to select data RAM Bank 0
                   bcf    STATUS,RP0
                   bcf    STATUS,RP1
                   ENDM

Bank1              MACRO            ; Macro to select data RAM Bank 1
                   bsf    STATUS,RP0
                   bcf    STATUS,RP1
                   ENDM

Bank2              MACRO            ; Macro to select data RAM Bank 2
                   bcf    STATUS,RP0
                   bsf    STATUS,RP1
                   ENDM
```

```
Bank3               MACRO            ; Macro to select data RAM Bank 3
                    bsf    STATUS,RP0
                    bsf    STATUS,RP1
                    ENDM

;*****************************************************************

                    ORG    0x0000          ; Processor reset vector
                    clrf   PCLATH          ; Ensure page bits are cleared
                    goto   main            ; Go to beginning of program
                    ORG    0x0004          ; Processor Interrupt Vector
                    goto   interrupt_start ; Go to interrupt service
                                           ; rotine
;-----------------------------------------------------------------
;                   Rotina Main
;-----------------------------------------------------------------
                    ORG    0x0005
main                                       ;Aqui começa o programa principal
                    call   InitCfg
while_1                                    ; while(1) {
                    btfsc  start_stop,0    ; while (start_stop == 1) {}
                    goto   $-1             ;

                    bsf    PORTB,RB1       ; Set RB1 ON
                    movlw  25              ;
                    call   Delay           ; delay( 0,25 seg )

                    bcf    PORTB,RB1       ; Set RB1 OFF
                    movlw  25              ;
                    call   Delay           ; delay( 0,25 seg )

                    goto   while_1         ; }

;*****************************************************************


;************************** ROTINAS *********************************
;*****************************************************************
;*****************************************************************
;=================================================================;
; InitCfg                                                         ;
; Configuração inicial dos registros associados                   ;
; ao programa principal, PORTOS I/O, Timer0, INTERRUPTS.          ;
;                                                                 ;
; Retorna em Bank0                                                ;
;                                                                 ;
; Notas:                                                          ;
;=================================================================;
InitCfg
```

```
;----------------- I/O - InitCfg ----------------------------------;
                  Bank1               ; DATA memory (RAM) Bank1
                  movlw  0x01         ; RB[1]         output
                  movwf  TRISB        ; RB[7:2 & 0]   inputs

;----------------- Interrupts enable bits -------------------------;
                  Bank0               ; DATA memory (RAM) Bank0
                  bsf    INTCON,GIE   ; Global Int. Enable bit
                  bsf    INTCON,INTE  ; External Int. enable bit

;------------------------------------------------------------------;
                  clrf   PORTB        ; Inicialização PORTB.
                  clrf   start_stop   ;

                  return

;==================================================================;
; Delay                                                            ;
; Pode gerar delays entre 10 mS e 2,5 S                            ;
; O valor de entrada e' passado em W (1..255)                      ;
;                                                                  ;
; Notas: o loop base e' de 1 uS (T=200 nS <=> 4*1/20MHz)           ;
;                                                                  ;
;==================================================================;
Delay
                  movwf  delay_mult   ; Carrega o valor multiplo
Del_m10           movlw  50           ; de 10 mS
                  movwf  delay_k50    ; 50 => 10 mS
Del_50            movlw  200          ; 200 => 200 uS
                  movwf  delay_k200   ; Carrega o valor
Del_200           nop                 ; correspondente a 200 uS
                  nop
                  decfsz delay_k200,f ; Decrementa "delay_k200".
                                      ; O resultado é colocadao em
                                      ; "delay_k200"
                  goto   Del_200      ; while( delay_k200 > 0 )
                  decfsz delay_k50,f  ;
                  goto   Del_50       ; while( delay_k50 > 0 )
                  decfsz delay_mult,f ; Decrementa "delay_mult".
                                      ; O resultado é colocadao
                                      ; em "delay_mult"
                  goto   Del_m10      ; while( delay_mult > 0 )

                  return

;******************************************************************


;************************* ISR ************************************
```

```
;*****************************************************************
;*****************************************************************
interrupt_start
                  movwf  w_save       ; salvaguarda do W reg.
                  swapf  STATUS,W     ;
                  clrf   STATUS       ; DATA memory (RAM) Bank0
                  movwf  status_save  ; salvaguarda do STATUS reg.
                  movf   PCLATH,W     ;
                  movwf  pclath_save  ; salvaguarda do PCLATH reg.
                  clrf   PCLATH       ; página 0, de memória de
                                      ; programa
                  bcf    INTCON,INTF  ; reset, do bit INTF
                  incf   start_stop,w ;
                  andlw  0x01         ;
                  movwf  start_stop   ; toggle do start_stop (0/1)
                                      ;
interrupt_end     movf   pclath_save,w; restauro do PCLATH reg.
                  movwf  PCLATH       ;
                  swapf  status_save,w; restauro da DATA memory (RAM)
                                      ; Bank? inicial
                  movwf  STATUS       ;
                  swapf  w_save,f     ; restauro do W reg.
                  swapf  w_save,w     ;

                  retfie

                  END
```