Microcontroladores

Programação em Assembly para PIC usando MPLab

Gustavo Medeiros

Universidade Federal de Pernambuco – UFPE
Dep. de Engenharia Elétrica – DEE

Outubro de 2014

Linguagem de Programação Assembly

- **Linguagem de programação** é um método padronizado para comunicar instruções para um μP de forma a executar um algoritmo;
- Principal objetivo da linguagem de programação → Aumentar a produtividade do programador → Pois, permite expressar o algoritmo de forma mais fácil do que em **código de máquina** (linguagem que um μP entende nativamente);

Código-fonte

```
1 #INCLUDE <P16F628A.INC>
2
3     ORG     0X00
4 MAIN_LOOP
5     MOVF    PORTA,W
6     MOVWF   PORTB
7     GOTO    MAIN_LOOP
8     END
```

Código em Intel Hex Format (INHX8M)

```
:020000040000FA
:0600000005088600002 83F
:04000600003400348E
:00000001FF
```

Código de máquina

Program

	Line	Address	Opcode	Label	DisAssy
➪	1	000	0805	MAIN_LOOP	MOVF PORTA, W
	2	001	0086		MOVWF PORTB
	3	002	2800		GOTO 0x0
	4	003	3400		RETLW 0x0
	5	004	3400		RETLW 0x0
	6	005	3FFF		ADDLW 0xFF
	7	006	3FFF		ADDLW 0xFF

Linguagem de Programação Assembly

- ***Assembly*** ou **linguagem de montagem** é uma notação legível por humanos para o código de máquina que é usado por uma determinada arquitetura de computador;
- A conversão da **linguagem de montagem** para o **código de máquina** é feita pelo **montador** ou ***assembler***, que é basicamente um tradutor de comandos, sendo mais simples que um **compilador**;

Código-fonte (asm) em linguagem *Assembly*

```
1  #INCLUDE <P16F628A.INC>
2
3      ORG      0X00
4  MAIN_LOOP
5      MOVF     PORTA,W
6      MOVWF    PORTB
7      GOTO     MAIN_LOOP
8      END
```

Assembler ou montador

Disassembler ou desmontador

Código de máquina em Intel Hex Format (INHX8M)

```
:020000040000FA
:0600000005088600002 83F
:04000600003400348E
:00000001FF
```

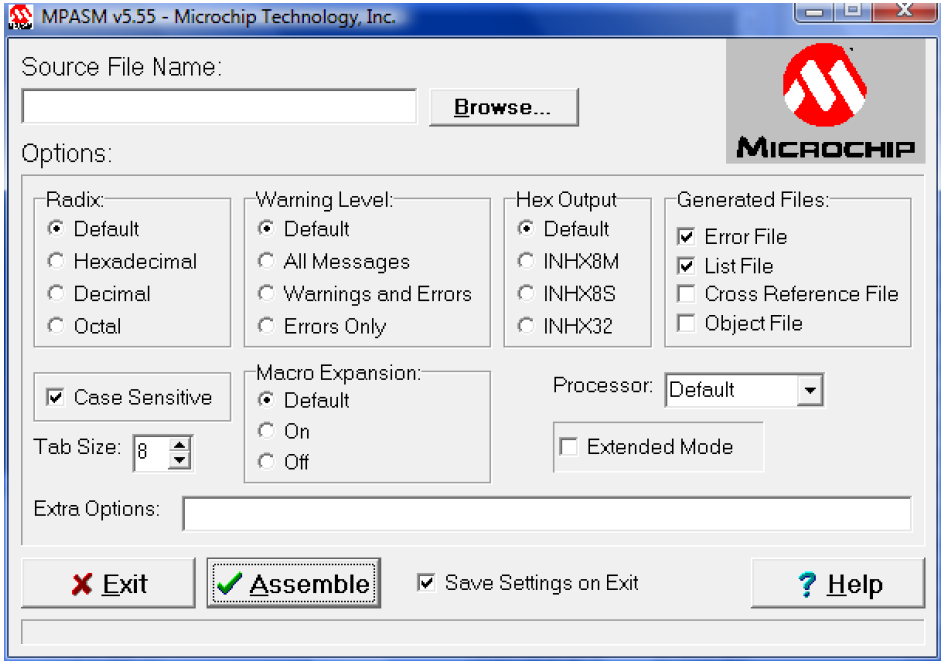
Linguagem de Programação Assembly

- ***Assembly*** é considerada uma **linguagem de baixo nível**, pois é muito próxima do **código de máquina** (há uma correspondência unívoca entre eles);
- Portanto, a linguagem *Assembly* varia de acordo com a máquina (μP). Por exemplo, o código-fonte em Assembly de um PIC é diferente do código-fonte Assembly de um MSP430 da Texas Instruments;
- **Linguagens de alto nível**, como a **linguagem C** por exemplo, precisam de um **compilador** para converter o código-fonte para *Assembly* ou diretamente para código de máquina. Neste caso, o mesmo código-fonte em C pode ser usado para diferentes máquinas (o compilador é que muda em função da máquina que receberá o código de máquina).

Programação Assembly para PIC

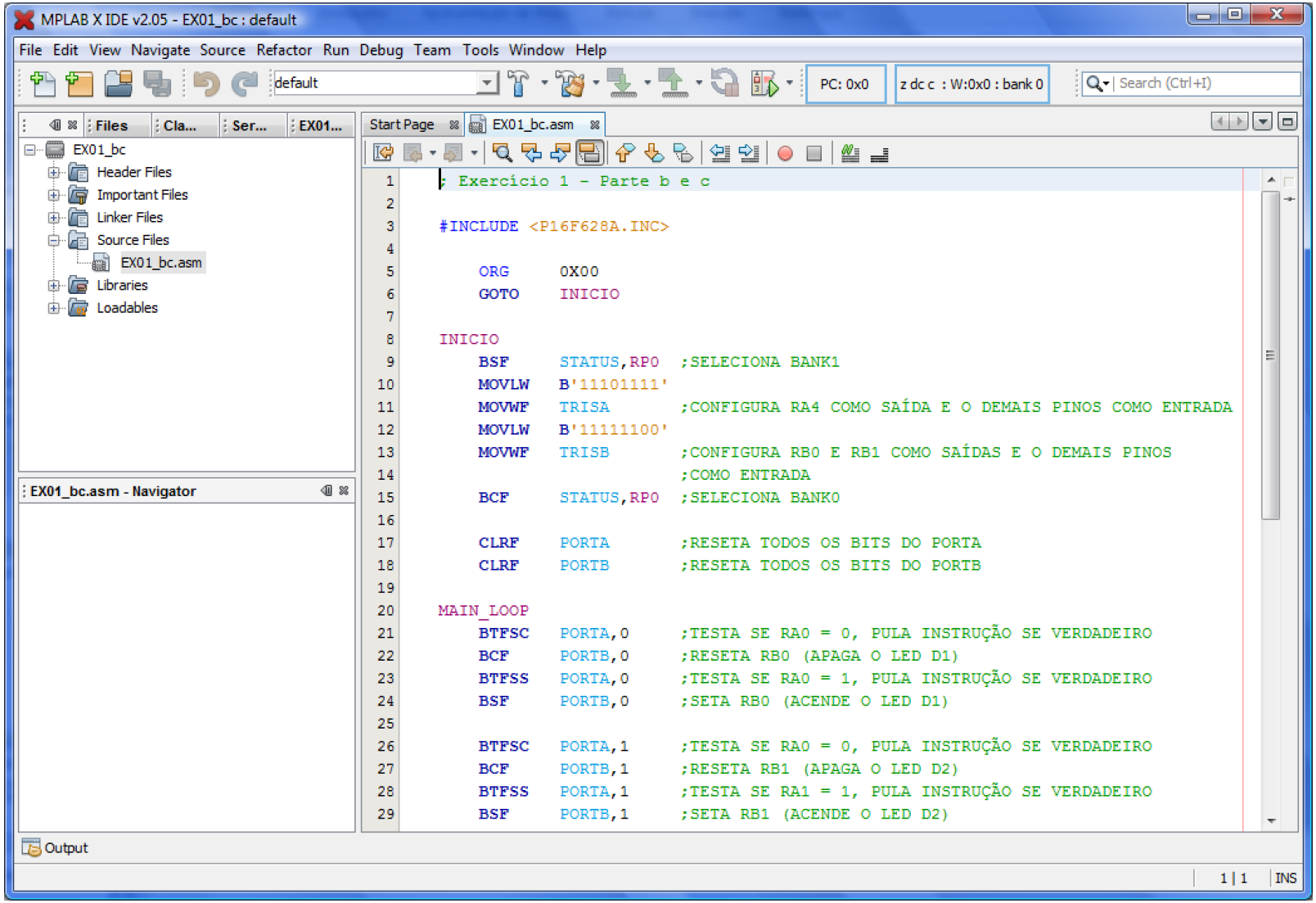
- A Microchip disponibiliza gratuitamente o programa *Assembler* (montador) MPASM;
- O código *Assembly* pode ser escrito em qualquer editor de texto simples (ex.: notepad);
- Porém, a Microchip também disponibiliza o MPLab, uma ferramentas para gerenciar os projetos desenvolvidos para os dispositivos da família PIC, que possui um editor e integra o MPASM.

Programa *Assembler* (montador) da Microchip



Programação Assembly para PIC

- Programa MPLab IDE (*Integrated Development Environment*):



Estrutura do Código-fonte

- Para que o programa funcione corretamente é necessário escrever as **instruções certas** na **ordem correta**;
- O código-fonte deve ser devidamente **estruturado** e **padronizado** para facilitar futuras alterações e o entendimento de outros programadores;
- Para isso, "***comentar o código***" também é muito importante.

Exemplo da estrutura do Código-fonte

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                              NOME DO PROJETO                                  *
;*   AUTOR: GUSTAVO AZEVEDO                                                      *
;*   VERSÃO: 1.0                                          DATA: 01/05/14         *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                            DESCRIÇÃO DO ARQUIVO                               *
;*--------------------------------------------------------------------------------*
;*   MODELO PARA PIC 16F628A                                                     *
;*                                                                               *
;*                                                                               *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                             ARQUIVOS DE DEFINIÇÕES                            *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
#INCLUDE <P16F628A.INC> ;ARQUIVO PADRÃO MICROCHIP PARA 16F628A

    __CONFIG _BOREN_ON & _CP_OFF & _PWRTE_ON & _WDT_OFF & _LVP_OFF & _MCLRE_ON & _XT_OSC

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                              PAGINAÇÃO DE MEMÓRIA                             *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;DEFINIÇÃO DE COMANDOS DE USUÁRIO PARA ALTERAÇÃO DA PÁGINA DE MEMÓRIA
#DEFINE BANK0   BCF STATUS,RP0  ;SETA BANK 0 DE MEMÓRIA
#DEFINE BANK1   BSF STATUS,RP0  ;SETA BANK 1 DE MAMÓRIA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                                  VARIÁVEIS                                    *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS
; PELO SISTEMA
    CBLOCK  0x20    ;ENDEREÇO INICIAL DA MEMÓRIA DE
                    ;USUÁRIO
        W_TEMP      ;REGISTRADORES TEMPORÁRIOS PARA USO
        STATUS_TEMP ;JUNTO ÀS INTERRUPÇÕES

        ;NOVAS VARIÁVEIS

    ENDC            ;FIM DO BLOCO DE MEMÓRIA
```

Exemplo da estrutura do Código-fonte

```
40  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
41  ;*                                FLAGS INTERNOS                                 *
42  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
43  ; DEFINIÇÃO DE TODOS OS FLAGS UTILIZADOS PELO SISTEMA
44
45  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
46  ;*                                 CONSTANTES                                    *
47  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
48  ; DEFINIÇÃO DE TODAS AS CONSTANTES UTILIZADAS PELO SISTEMA
49
50  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
51  ;*                                  ENTRADAS                                     *
52  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
53  ; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO ENTRADA
54  ; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)
55
56  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
57  ;*                                   SAÍDAS                                      *
58  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
59  ; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO SAÍDA
60  ; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)
61
62  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
63  ;*                                VETOR DE RESET                                 *
64  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
65
66      ORG 0x00            ;ENDEREÇO INICIAL DE PROCESSAMENTO
67      GOTO    INICIO
68
69  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
70  ;*                            INÍCIO DA INTERRUPÇÃO                              *
71  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
72  ; ENDEREÇO DE DESVIO DAS INTERRUPÇÕES. A PRIMEIRA TAREFA É SALVAR OS
73  ; VALORES DE "W" E "STATUS" PARA RECUPERAÇÃO FUTURA
74
75      ORG 0x04            ;ENDEREÇO INICIAL DA INTERRUPÇÃO
76      MOVWF   W_TEMP      ;COPIA W PARA W_TEMP
77      SWAPF   STATUS,W
78      MOVWF   STATUS_TEMP ;COPIA STATUS PARA STATUS_TEMP
```

Exemplo da estrutura do Código-fonte

```
80    ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
81    ;*                              ROTINA DE INTERRUPÇÃO                            *
82    ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
83    ; AQUI SERÁ ESCRITA AS ROTINAS DE RECONHECIMENTO E TRATAMENTO DAS
84    ; INTERRUPÇÕES
85
86    ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
87    ;*                         ROTINA DE SAÍDA DA INTERRUPÇÃO                        *
88    ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
89    ; OS VALORES DE "W" E "STATUS" DEVEM SER RECUPERADOS ANTES DE
90    ; RETORNAR DA INTERRUPÇÃO
91
92    SAI_INT
93        SWAPF   STATUS_TEMP,W
94        MOVWF   STATUS      ;MOVE STATUS_TEMP PARA STATUS
95        SWAPF   W_TEMP,F
96        SWAPF   W_TEMP,W    ;MOVE W_TEMP PARA W
97        RETFIE
98
99    ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
100   ;*                              ROTINAS E SUBROTINAS                             *
101   ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
102   ; CADA ROTINA OU SUBROTINA DEVE POSSUIR A DESCRIÇÃO DE FUNCIONAMENTO
103   ; E UM NOME COERENTE ÀS SUAS FUNÇÕES.
104
105   SUBROTINA1
106
107       ;CORPO DA ROTINA
108
109       RETURN
110
111   ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
112   ;*                               INICIO DO PROGRAMA                              *
113   ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
114
115   INICIO
116       BANK1               ;ALTERA PARA O BANCO 1
117       MOVLW   B'00000000'
118       MOVWF   TRISA       ;DEFINE ENTRADAS E SAÍDAS DO PORTA
```

Exemplo da estrutura do Código-fonte

```
110
111     ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
112     ;*                          INICIO DO PROGRAMA                                   *
113     ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
114
115     INICIO
116         BANK1               ;ALTERA PARA O BANCO 1
117         MOVLW   B'00000000'
118         MOVWF   TRISA       ;DEFINE ENTRADAS E SAÍDAS DO PORTA
119         MOVLW   B'00000000'
120         MOVWF   TRISB       ;DEFINE ENTRADAS E SAÍDAS DO PORTB
121         MOVLW   B'10000100'
122         MOVWF   OPTION_REG  ;DEFINE OPÇÕES DE OPERAÇÃO
123         MOVLW   B'00000000'
124         MOVWF   INTCON      ;DEFINE OPÇÕES DE INTERRUPÇÕES
125         BANK0               ;RETORNA PARA O BANCO
126         MOVLW   B'00000111'
127         MOVWF   CMCON       ;DEFINE O MODO DE OPERAÇÃO DO COMPARADOR ANALÓGICO
128
129     ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
130     ;*                       INICIALIZAÇÃO DAS VARIÁVEIS                             *
131     ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
132
133     ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
134     ;*                          ROTINA PRINCIPAL                                     *
135     ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
136     MAIN
137
138         ;CORPO DA ROTINA PRINCIPAL
139
140         GOTO MAIN
141
142     ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
143     ;*                          FIM DO PROGRAMA                                      *
144     ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
145
146         END
147
```

Código-fonte Assembly

- O código-fonte em linguagem Assembly é composto de:
  - **Instruções** → Nome mnemônico dado a uma operação que o microcontrolador pode realizar. Ex.: No PIC16F628A tem-se **`MOVWF`** = 0000001xxxxxxx;
  - **Parâmetros** → Informações manipuladas por uma instrução. Ex.: **`BCF`** `PORTA,2`;
  - **Rótulos** → Nomes dados as linhas do programa. Servem para que em uma instrução de desvio possa se determinar o ponto para onde se deseja ir no programa. Os rótulos sempre são alinhados na coluna 0 (sem espaços antes do mesmo), enquanto que as instruções devem ser escritas após uma margem (obrigatoriamente após a coluna 0);

Código-fonte Assembly

- **Diretivas** → São linhas que determinam como o programa montador irá operar. Não geram efeito direto no código binário gerado;
- **Comentários** → São trechos de texto escritos após um sinal de ponto e vírgula (;). São úteis para que possamos adicionar pequenos lembretes no programa, facilitando a manutenção futura. Não interferem no tamanho do programa binário gerado.

Diretivas Assembler

- Diferentemente das instruções (MOV, ADD, CALL...), que são comandos direcionados à CPU, as diretivas são comandos direcionados ao *Assembler* (montador);
- As **diretivas** são executadas em tempo de "compilação" (montagem). Já as **instruções** são executadas em tempo de execução;
- Diretivas podem ser comandos para reservar áreas de memória, definir procedimentos, definir constantes, entre outros.

Diretivas Assembler

#INCLUDE

Sintaxe:

```
#INCLUDE <[nome do arquivo de inclusão]>
#INCLUDE “[nome do arquivo de inclusão]”
```

Exemplos:

```
#INCLUDE <P16F628A.INC>
#INCLUDE "C:\PROJETOS\MACRO.INC"
```

- O arquivo especificado é lido como um código-fonte e incluído como texto completo a partir da posição onde a diretiva estiver escrita;

Diretivas Assembler

`#INCLUDE`

- Esta diretiva é usada para incluir no código-fonte o arquivo padrão do μC escolhido. Neste arquivo estão definidos todos os **nomes dos registradores** e os **bits** do dispositivo selecionado, não sendo necessário defini-los dentro do seu código-fonte;
- Também pode ser usada para incluir arquivos contendo sequências funcionais já desenvolvidas (Macros);
- Os arquivos a serem incluídos devem possuir a extensão *.INC*

Diretivas Assembler

`#DEFINE`

Sintaxe:

```
#DEFINE [nome]   [texto]
#DEFINE [nome]   [instrução]
```

Exemplos:

```
4       #DEFINE    BANK0    BCF STATUS,RP0
5       #DEFINE    BANK1    BSF STATUS,RP0
6
7       #DEFINE    BOTAO    PORTA,0
```

- Define a substituição do `[nome]` no código-fonte pelo `[texto]` ou `[instrução]`;
- Só é possível a substituição de uma única linha de instrução. Para a substituição de duas ou mais linhas de instrução (sequência) deve-se criar uma Macro.

Diretivas Assembler

EQU

Sintaxe:

```
[nome]   EQU   [valor]
[nome]   EQU   [expressão]
```

Exemplos:

```
4    PORTA       EQU     H'0005'
5
6    MYCONST1    EQU     0XA2
7    MYCONST2    EQU     B'00011001'
8    MYCONST3    EQU     (.123 - .23)
```

- Relaciona [nome] a um valor numérico que pode ser representado na forma binária, decimal ou hexadecimal;

Diretivas Assembler

`EQU`

- Formas de representação:
  - Valor decimal: `D'dd'` ou `.dd`
  - Valor hexa: `H'dd'` ou `0Xdd`
  - Valor binário: `B'dddddddd'`
  - Valor ASCII: `A'l'`

- No lugar de um valor numérico pode-se utilizar uma expressão aritmética onde os valores numéricos dentro da expressão podem ser representados na forma binária, decimal ou hexa;
- O *Assembler* somente interpreta valores numéricos inteiros e positivos dentro do intervalo de 0 a 255;
- O resultado de uma expressão deve ser um valor numérico inteiro e positivo dentro do intervalo de 0 a 255;

Diretivas Assembler

CONSTANT

Sintaxe:

```
CONSTANT [nome]=[valor]
CONSTANT [nome]=[expressão]
```

Exemplos:

```
4
5       CONSTANT    MYCONST1 = 0XA2
6       CONSTANT    MYCONST2 = B'00011001'
7       CONSTANT    MYCONST3 = (.23 - .123)
8
```

- A diretiva CONSTANT tem o mesmo efeito da diretiva EQU;

Diretivas Assembler

CBLOCK e ENDC

Sintaxe:

```
CBLOCK [endereço memória]
ENDC
```

Exemplos:

```
20          CBLOCK  0x20
21              VAR1
22              VAR2
23          ENDC
```

- Define uma lista de **Nomes de Variáveis** que serão alocadas na memória RAM a partir do endereço especificado. A primeira variável da lista é alocada no endereço de memória especificado, enquanto as demais variáveis da lista são alocadas sequencialmente nos endereços seguintes;
- A lista de Nomes termina quando a diretiva ENDC é encontrada.

Diretivas Assembler

ORG

Sintaxe:

```
ORG [endereço memória]
```

Exemplos:

```
23    ;VETOR RESET
24      ORG       0X00
25      GOTO      INICIO
26
27    ;INTERRUPÇÃO
28      ORG 0X04
29      RETFIE
```

- Define o endereço da memória de programa em que parte do código-fonte deve ser colocado;
- Para a colocação de diretivas ORG sucessivas, o número de endereços disponíveis após cada diretiva ORG deve ser suficiente para conter todas as instruções previstas para este bloco;
- Se não houver nenhuma outra diretiva ORG em uma sequência de instruções, o compilador grava todas as instruções sequencialmente na memória.

Diretivas Assembler

END

Sintaxe:

```
END
```

- Indica o fim de um código-fonte. É necessária no final do código-fonte para indicar ao compilar o término da sequência de instruções do programa.

Diretivas Assembler

__CONFIG

Sintaxe:

```
__CONFIG  [dado de configuração]
```

Exemplo:

```
3     __CONFIG _BOREN_ON & _CP_OFF & _PWRTE_ON & _WDT_OFF & _LVP_OFF & _MCLRE_OFF
4             & _XT_OSC
```

- Define qual será o dado escrito na memória de configuração do PIC (endereço 0x2007 no PIC16F628A). Cada bit (dos 14 bits) seleciona uma configuração de um dos dispositivo do PIC;
- Os bits de configuração pertencem ao **espaço de memória especial de configuração** (0x2000 à 0x3FFF), o qual só pode ser acessado durante a gravação do PIC.

Diretivas Assembler

__CONFIG

- Registro de configuração (@ 0x2007):

REGISTER 14-1: CONFIG – CONFIGURATION WORD REGISTER

$\overline{CP}$	—	—	—	—	$\overline{CPD}$	LVP	BOREN	MCLRE	FOSC2	$\overline{PWRTE}$	WDTE	F0SC1	F0SC0
bit 13													bit 0

bit 13 **CP:** Flash Program Memory Code Protection bit
1 = Code protection off
0 = 0000h to 07FFh code-protected

bit 8 **CPD:** Data Code Protection bit
1 = Data memory code protection off
0 = Data memory code-protected

bit 7 **LVP:** Low-Voltage Programming Enable bit
1 = RB4/PGM pin has PGM function, low-voltage programming enabled
0 = RB4/PGM is digital I/O, HV on MCLR must be used for programming

bit 6 **BOREN:** Brown-out Reset Enable bit
1 = BOR Reset enabled
0 = BOR Reset disabled

bit 5 **MCLRE:** RA5/MCLR/VPP Pin Function Select bit
1 = RA5/MCLR/VPP pin function is MCLR
0 = RA5/MCLR/VPP pin function is digital Input

bit 3 **PWRTE:** Power-up Timer Enable bit
1 = PWRT disabled
0 = PWRT enabled

bit 2 **WDTE:** Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 4, 1-0 FOSC<2:0>: Oscillator Selection bits
111 = RC oscillator: CLKOUT function on RA6 pin, Resistor and Capacitor on RA7
110 = RC oscillator: I/O function on RA6 pin, Resistor and Capacitor on RA7
101 = INTOSC oscillator: CLKOUT function on RA6 pin, I/O function on RA7
100 = INTOSC oscillator: I/O function on RA6 pin, I/O function on RA7
011 = EC: I/O function on RA6 pin, CLKIN on RA7
010 = HS oscillator: High-speed crystal/resonator on RA6 and RA7
001 = XT oscillator: Crystal/resonator on RA6 and RA7
000 = LP oscillator: Low-power crystal on RA6 and RA7

Diretivas Assembler

__CONFIG

- Valores definidos no arquivo padrão do PIC (P16F628A.INC):

```
;----- CONFIG Options -----------------------------------------------
_LP_OSC              EQU  H'3FEC'     ; LP oscillator: Low-power crystal on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN
_XT_OSC              EQU  H'3FED'     ; XT oscillator: Crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN
_HS_OSC              EQU  H'3FEE'     ; HS oscillator: High-speed crystal/resonator on RA6/OSC2/CLKOUT and RA7/OSC1/CLKIN
_EXTCLK_OSC          EQU  H'3FEF'     ; EC: I/O function on RA6/OSC2/CLKOUT pin, CLKIN on RA7/OSC1/CLKIN
_INTRC_OSC_NOCLKOUT  EQU  H'3FFC'     ; INTOSC oscillator: I/O function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN
_INTRC_OSC_CLKOUT    EQU  H'3FFD'     ; INTOSC oscillator: CLKOUT function on RA6/OSC2/CLKOUT pin, I/O function on RA7/OSC1/CLKIN
_RC_OSC_NOCLKOUT     EQU  H'3FFE'     ; RC oscillator: I/O function on RA6/OSC2/CLKOUT pin, Resistor and Capacitor on RA7/OSC1/CLKIN
_RC_OSC_CLKOUT       EQU  H'3FFF'     ; RC oscillator: CLKOUT function on RA6/OSC2/CLKOUT pin, Resistor and Capacitor on RA7/OSC1/CLKIN

_WDT_OFF             EQU  H'3FFB'    ; WDT disabled
_WDT_ON              EQU  H'3FFF'    ; WDT enabled

_PWRTE_ON            EQU  H'3FF7'    ; PWRT enabled
_PWRTE_OFF           EQU  H'3FFF'    ; PWRT disabled

_MCLRE_OFF           EQU  H'3FDF'     ; RA5/MCLR/VPP pin function is digital input, MCLR internally tied to VDD
_MCLRE_ON            EQU  H'3FFF'    ; RA5/MCLR/VPP pin function is MCLR

_BOREN_OFF           EQU  H'3FBF'    ; BOD disabled
_BOREN_ON            EQU  H'3FFF'    ; BOD enabled

_LVP_OFF             EQU  H'3F7F'     ; RB4/PGM pin has digital I/O function, HV on MCLR must be used for programming
_LVP_ON              EQU  H'3FFF'     ; RB4/PGM pin has PGM function, low-voltage programming enabled

_CPD_ON              EQU  H'3EFF'    ; Data memory code-protected
_CPD_OFF             EQU  H'3FFF'    ; Data memory code protection off

_CP_ON               EQU  H'1FFF'    ; 0000h to 07FFh code-protected
_CP_OFF              EQU  H'3FFF'    ; Code protection off
```

Vetor de Reset e a Inicialização do Sistema

- Vetor de reset é o endereço para o qual o programa é desviado toda vez que ocorre um reset, seja ele pela energização do PIC, pelo *Master Clear* (MCLR) externo ou pelo estouro do *Watchdog Timer* (WDT);
- No endereço apontado pelo vetor de reset deve conter a **inicialização do sistema**;
- Essa inicialização deve preocupar-se primeiro com a configuração do μC e depois com as variáveis do programa;
- Os registradores que dever ser configurados são: TRISA, TRISB, OPTION_REG e INTCON.

Vetor de Reset e a Inicialização do Sistema

Exemplo:

```
62  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
63  ;*                                VETOR DE RESET                               *
64  ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
65
66      ORG 0x00            ;ENDEREÇO INICIAL DE PROCESSAMENTO
67      GOTO    INICIO
```

```
111 ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
112 ;*                              INICIO DO PROGRAMA                             *
113 ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
114
115 INICIO
116     BANK1               ;ALTERA PARA O BANCO 1
117     MOVLW   B'00000000'
118     MOVWF   TRISA       ;DEFINE ENTRADAS E SAÍDAS DO PORTA
119     MOVLW   B'00000000'
120     MOVWF   TRISB       ;DEFINE ENTRADAS E SAÍDAS DO PORTB
121     MOVLW   B'10000100'
122     MOVWF   OPTION_REG  ;DEFINE OPÇÕES DE OPERAÇÃO
123     MOVLW   B'00000000'
124     MOVWF   INTCON      ;DEFINE OPÇÕES DE INTERRUPÇÕES
125     BANK0               ;RETORNA PARA O BANCO
126     MOVLW   B'00000111'
127     MOVWF   CMCON       ;DEFINE O MODO DE OPERAÇÃO DO COMPARADOR ANALÓGICO
```

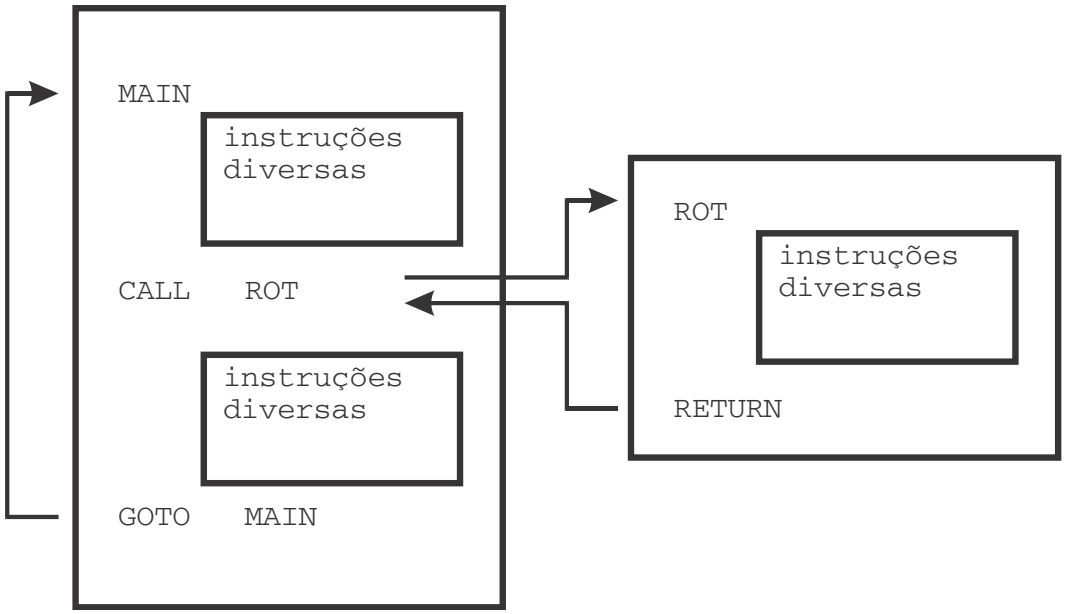
Inicialização das Variáveis

- É importante inicializar as variáveis na inicialização do sistema para garantir que o seus valores correspondam ao esperado;
- As variáveis devem ser inicializadas mesmo que seus valores devam ser zero.

```
116    ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
117    ;*                    INICIALIZAÇÃO DAS VARIÁVEIS                    *
118    ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
119
120        CLRF    PORTA       ;ZERA TODAS AS SAÍDAS DO PORTA
121        CLRF    PORTB       ;ZERA TODAS AS SAÍDAS DO PORTB
122        MOVLW   .10
123        MOVWF   CONTADOR    ;INICIA CONTADOR = 10
```

Rotinas

- Há dois tipos:
  - Rotinas de desvio → usa o instrução `GOTO`;
  - Rotinas de chamada → usa o instrução `CALL`;



Rotina de Desvio

- As rotinas de desvio produzem saltos na ordem de execução do programa usando a instrução `GOTO`;

Sintaxe:

```
GOTO [nome]
```

- `[nome]` é um rótulo que está relacionado a uma linha do programa;
- O rótulo não pode conter espaços. É aconselhável usar sublinhado (_) quando se deseje separar palavras;
- Para saltos curtos pode-se usar o operador $, que indica o valor do registrador PC.

Exemplo:

```
176         BTFSS   BOTAO       ;O BOTÃO CONTINUA PRESSIONADO?
177         GOTO    $-1         ;SIM, ENTÃO ESPERA LIBERAÇÃO
178         GOTO    MAIN        ;NÃO, VOLTA AO LOOP PRINCIPAL
179
```

Rotina de Chamada

- As rotinas de chamada são utilizadas quando uma tarefa deve ser repetida várias vezes e não deseja-se reescrevê-la para economizar espaço de memória de programa;
- Desta forma, a rotina pode ser usada como uma função, que é chamada de diversos pontos do programa;
- Esta rotina é chamada através da instrução `CALL`;

Sintaxe:

```
CALL [nome]
```

- As mesmas observações feitas para `[nome]` na instrução `GOTO` são válidas aqui;
- Quando se usa a instrução `CALL`, o endereço de programa seguinte ($+1) é armazenado na pilha (*Stack*). Isso permite retornar ao mesmo ponto quando a rotina for concluída.

Rotina de Chamada

- Outras rotinas podem ser chamadas dentro da rotina atual. Isso gera outros níveis de pilha com os respectivos endereços de retorno;
- Deve-se ter cuidado para não ultrapassar o limite da pilha. No PIC16F628A a pilha tem oito níveis;
- Para retornar de uma rotina, devem ser utilizadas as instruções `RETURN` ou `RETLW`;

Sintaxe:

```
RETURN
RETLW   k
```

Rotina de Chamada

Exemplo:

```
113     ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
114     ;*                          ROTINA DE DELAY                              *
115     ;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
116     ; ESTA ROTINA AGUARDA TANTOS MILISEGUNDOS QUANTO O VALOR PASSADO
117     ; POR W. POR EXEMPLO, SE W = .200, ELA AGUARDARÁ 200 MILISEGUNDOS.
118     ;
119     ; O DELAY PRINCIPAL DURA 1ms, POIS POSSUI 5 INSTRUÇÕES (5us) E É
120     ; RODADO 200 VEZES (TEMPO1). PORTANTO 200 * 5us = 1ms.
121     ; O DELAY PRINCIPAL É RODADO TANTAS VEZES QUANTO FOR O VALOR DE
122     ; TEMPO2, O QUAL É INICIADO COM O VALOR PASSADO EM W.
123
124
125     DELAY
126         MOVWF   TEMPO2      ;INICIA TEMPO 2 COM O VALOR
127                             ;PASSADO EM W
128     DL1
129         MOVLW   .200
130         MOVWF   TEMPO1
131
132     DL2                     ;ESTE DELAY DURA 1ms (5*200)
133         NOP
134         NOP
135         DECFSZ  TEMPO1,F    ;DECREMENTA TEMPO1. ACABOU?
136         GOTO    DL2         ;NÃO, CONTINUA AGUARDANDO
137                             ;SIM
138
139         DECFSZ  TEMPO2,F    ;DECREMENTA TEMPO2. ACABOU?
140         GOTO    DL1         ;NÃO, CONTINUA AGUARDANDO
141                             ;SIM
142         RETURN
143
```

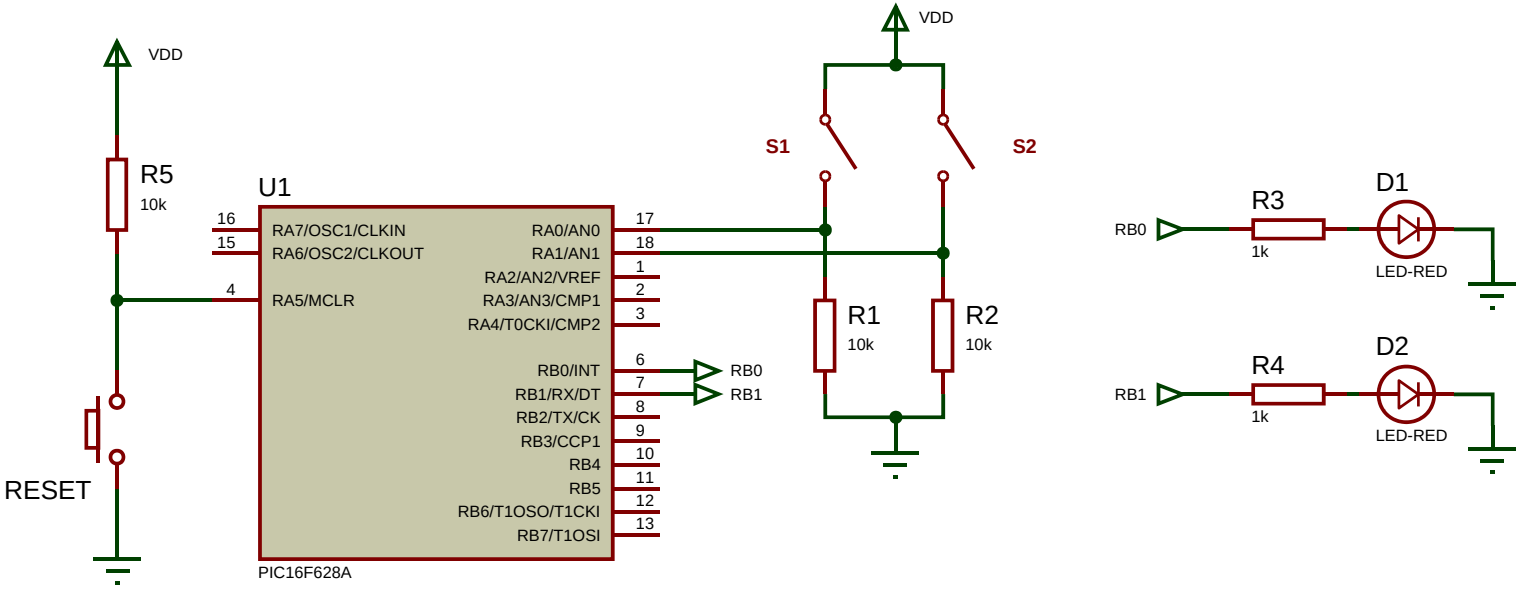
Exemplo 1

1. Criar um circuito com o PIC16F628A que possua as seguintes características:
   - Um botão *push-pull* para *resetar* o microcontrolador;
   - Um LED (D1) em RB0 que mude de estado cada vez que iniciar o loop principal do programa;
   - Duas chaves, S1 e S2, tipo *on-off* nas portas RA0 e RA1;
   - Um LED (D2) em RB1 que oscile em uma frequência definida pelo estados das chaves S1 e S2 conforme a tabela:

S2	**S1**	**D1**
0	0	Apagado
0	1	Período de oscilação de 500 ms
1	0	Período de oscilação de 750 ms
1	1	Período de oscilação de 1,2 s

Solução do Exemplo 1

- Diagrama do circuito elétrico:



Solução do Exemplo 1

- Código fonte do programa:

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                        LED OSCILANTE                                *
;*   AUTOR: GUSTAVO AZEVEDO                                            *
;*   VERSÃO: 1.0                              DATA: 10/11/14           *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                     DESCRIÇÃO DO ARQUIVO                            *
;*-----------------------------------------------------------------------*
;*   PROGRAMA PARA FAZER UM LED OSCILADOR COM FREQUENCIA DEFINIDA       *
;* PELA CHAVES S1 E S2 (EM RA0 E RA1)                                   *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                     ARQUIVOS DE DEFINIÇÕES                          *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
#INCLUDE <P16F628A.INC>  ;ARQUIVO PADRÃO MICROCHIP PARA 16F628A

__CONFIG _BOREN_ON & _CP_OFF & _PWRTE_ON & _WDT_OFF & _LVP_OFF & _MCLRE_ON & _XT_OSC

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                    PAGINAÇÃO DE MEMÓRIA                             *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;DEFINIÇÃO DE COMANDOS DE USUÁRIO PARA ALTERAÇÃO DA PÁGINA DE MEMÓRIA
#DEFINE BANK0    BCF STATUS,RP0   ;SETA BANK 0 DE MEMÓRIA
#DEFINE BANK1    BSF STATUS,RP0   ;SETA BANK 1 DE MAMÓRIA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                        VARIÁVEIS                                    *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS
; PELO SISTEMA
    CBLOCK  0x20     ;ENDEREÇO INICIAL DA MEMÓRIA DE USUÁRIO
        TEMPO1       ;VARIÁVEL USADA NO FUNÇÃO DELAY
        TEMPO2       ;VARIÁVEL USADA NO FUNÇÃO DELAY
    ENDC             ;FIM DO BLOCO DE MEMÓRIA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                          ENTRADAS                                   *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
#DEFINE S1   PORTA,0
#DEFINE S2   PORTA,1

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                         SAÍDAS                                      *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
#DEFINE LED_D1   PORTB,0
#DEFINE LED_D2   PORTB,1
```

Continuação:

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                        VETOR DE RESET                               *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    ORG 0x00             ;ENDEREÇO INICIAL DE PROCESSAMENTO
    GOTO    INICIO

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                     INÍCIO DA INTERRUPÇÃO                           *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    ORG 0x04             ;ENDEREÇO INICIAL DA INTERRUPÇÃO
    RETFIE

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                    ROTINAS E SUBROTINAS                             *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; ESTA ROTINA AGUARDA TANTOS MILISEGUNDOS QUANTO O VALOR PASSADO POR
; TEMPO2 POR EXEMPLO, SE TEMPO2=.200, ELA AGUARDARÁ 200 MILISEGUNDOS.
;
; O DELAY PRINCIPAL DURA 1ms, POIS POSSUI 5 INSTRUÇÕES (5us) E É
; RODADO 200 VEZES (TEMPO1). PORTANTO 200 * 5us = 1ms.
; O DELAY PRINCIPAL É RODADO TANTAS VEZES QUANTO FOR O VALOR DE TEMPO2

DELAY
    MOVWF   TEMPO2       ;INICIA TEMPO 2 COM O VALOR PASSADO POR W

DL1
    MOVLW   .200
    MOVWF   TEMPO1

DL2                      ;ESTE DELAY DURA 1ms (5*200)
    NOP
    NOP
    DECFSZ  TEMPO1,F     ;DECREMENTA TEMPO1. ACABOU?
    GOTO    DL2          ;NÃO, CONTINUA AGUARDANDO
                         ;SIM

    DECFSZ  TEMPO2,F     ;DECREMENTA TEMPO2. ACABOU?
    GOTO    DL1          ;NÃO, CONTINUA AGUARDANDO
                         ;SIM
    RETURN
```

Solução do Exemplo 1

- Código fonte do programa (Continuação):

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                     INICIO DO PROGRAMA                                *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
INICIO
    CLRF    PORTB       ;GARANTE QUE OS PINO RB0 E RB1 ESTARÃO EM 0
                        ;QUANDO SETADO COMO SAÍDA
    BANK1               ;ALTERA PARA O BANCO 1
    MOVLW '1B111111'
    MOVWF   TRISA       ;DEFINE ENTRADAS E SAÍDAS DO PORTA
    MOVLW '1B111100'
    MOVWF   TRISB       ;DEFINE ENTRADAS E SAÍDAS DO PORTB
    BANK0               ;RETORNA PARA O BANCO

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                     ROTINA PRINCIPAL                                  *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
MAIN
    MOVLW   B'00000001'
    XORWF   PORTB,F     ;1 XOR RB0 -> MUDA ESTADO DE RB0
                        ;0 XOR RB{1..7} -> MANTEM O ESTADO DE RB{1..7}
    BTFSS   S1
    GOTO    TEST_S2
    BTFSS   S2
    GOTO    T_500MS     ;QUANDO S1=1 E S2=0
    GOTO    T_1200MS    ;QUANDO S1=1 E S2=1

TEST_S2                 ;TESTA S2 QUANDO S1=0
    BTFSC   S2
    GOTO    T_750MS     ;QUANDO S1=0 E S2=1
    BCF     LED_D2      ;QUANDO S1=0 E S2=0
    GOTO MAIN

T_500MS
    BSF     LED_D2      ;ACENDE O LED
    MOVLW   .250        ;DEFINE TEMPO DO DELAY (250ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY
    BCF     LED_D2      ;APAGA LED
    MOVLW   .250        ;DEFINE TEMPO DO DELAY (250ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY
    GOTO MAIN
```

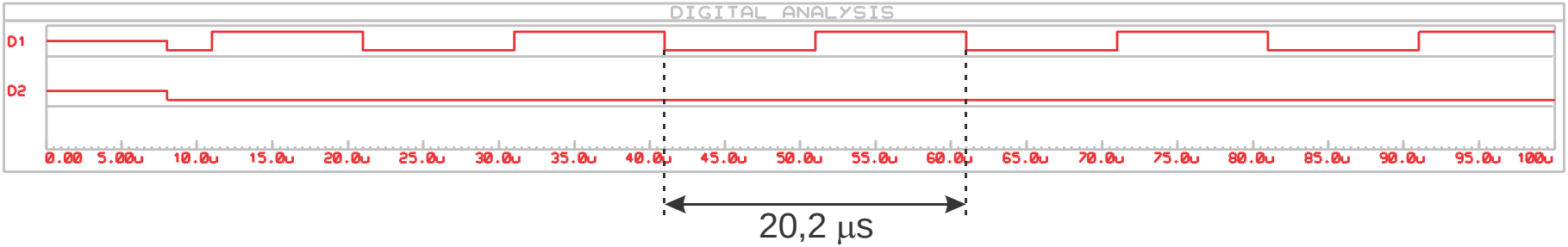
Continuação:

```
T_750MS
    BSF     LED_D2      ;ACENDE O LED
    MOVLW   .250        ;DEFINE TEMPO DO DELAY (250ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY PARA ESPERAR 250ms
    MOVLW   .125        ;DEFINE TEMPO DO DELAY (125ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY PARA ESPERAR MAIS 125ms
    BCF     LED_D2      ;APAGA LED
    MOVLW   .250        ;DEFINE TEMPO DO DELAY (250ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY
    MOVLW   .125        ;DEFINE TEMPO DO DELAY (125ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY PARA ESPERAR MAIS 125ms
    GOTO MAIN

T_1200MS
    BSF     LED_D2      ;ACENDE O LED
    MOVLW   .200        ;DEFINE TEMPO DO DELAY (200ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY PARA ESPERAR 200ms
    MOVLW   .200        ;DEFINE TEMPO DO DELAY (200ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY PARA ESPERAR MAIS 200ms
    MOVLW   .200        ;DEFINE TEMPO DO DELAY (200ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY PARA ESPERAR MAIS 200ms (TOTAL=600ms)
    BCF     LED_D2      ;APAGA LED
    MOVLW   .200        ;DEFINE TEMPO DO DELAY (200ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY PARA ESPERAR 200ms
    MOVLW   .200        ;DEFINE TEMPO DO DELAY (200ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY PARA ESPERAR MAIS 200ms
    MOVLW   .200        ;DEFINE TEMPO DO DELAY (200ms)
    CALL    DELAY       ;CHAMA A ROTINA DELAY PARA ESPERAR MAIS 200ms (TOTAL=600ms)
    GOTO MAIN

    END
```

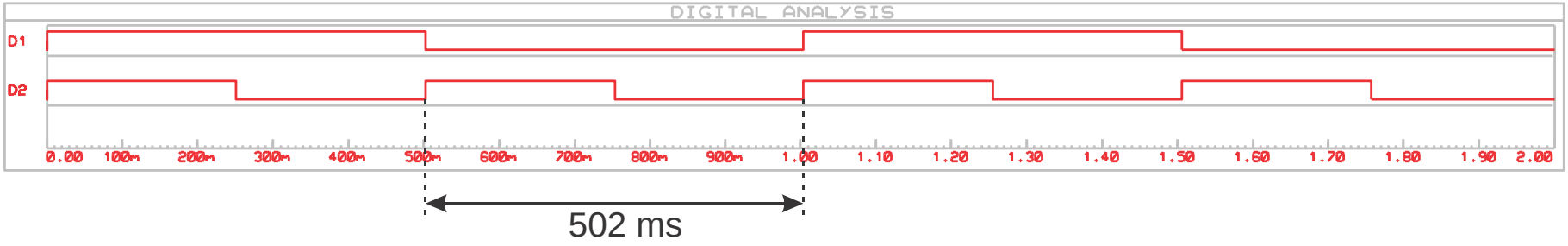
Uso da memória de programa desta versão do código:
66 Words.

Resultados do Exemplo 1

- Resultados para:
  - $S_2 = 0$; $S_1 = 0$



  - $S_2 = 0$; $S_1 = 1$



Solução do Exemplo 1

- Otimização do Código:

OBS.: Apenas a rotina principal é diferente em relação ao código mostrado anteriormente.

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                    ROTINA PRINCIPAL                                   *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
MAIN
    MOVLW   B'00000001'
    XORWF   PORTB,F     ;1 XOR RB0 -> MUDA ESTADO DE RB0
                        ;0 XOR RB{1..7} -> MANTEM O ESTADO DE RB{1..7}

    BCF     LED_D2      ;APAGA O LED
LOOP
    MOVF    PORTA,W
    ANDLW   B'00000011'
    ADDWF   PCL,F
    GOTO    MAIN
    GOTO    T_500MS
    GOTO    T_750MS     ;SETA O FLAG F_1200MS QUANDO S1=1 E S2=1

T_1200MS
    MOVLW   .225         ;DEFINE TEMPO DO DELAY PARA O CASO DE ELE SER UTILIZADO QUANDO F_1200MS ESTIVER ATIVO
    CALL    DELAY        ;CHAMA A ROTINA DELAY SE O FLAG F_1200MS ESTIVER ATIVO. O TEMPO SERÁ 250ms
T_750MS
    MOVLW   .125          ;DEFINE TEMPO DO DELAY PARA O CASO DE ELE SER UTILIZADO QUANDO F_750MS ESTIVER ATIVO
    CALL    DELAY        ;CHAMA A ROTINA DELAY SE O FLAG F_750MS ESTIVER ATIVO. O TEMPO SERÁ 75ms
T_500MS
    MOVLW   .250        ;DEFINE TEMPO DO DELAY (250ms)
    CALL    DELAY        ;CHAMA A ROTINA DELAY. ESTE DELAY VAI OCORRER PARA QUALQUER CASO

    BTFSC   LED_D2
    GOTO    MAIN        ;RETORNA PARA O MAIN. O LED É APAGADO NO MAIN
    BSF     LED_D2      ;ACENDE O LED E REPETE O DELAY DE UM SEMICICLO
    GOTO    LOOP

    END
```

Uso da memória de programa desta versão do código: 41 Words. **38% de redução da memória de programa.**

Operações Aritméticas Básicas

- A maioria dos programas implementados em μC necessitam de algum tipo de cálculo para que sua lógica funcione corretamente;
- Os PICs não possuem instruções para fazer cálculos complexos;
- Porém os PICs possuem os recursos necessários para implementar funções customizadas para cálculos muito mais avançados.

Operações Aritméticas Básicas

Somando

- Dois grupos de instruções:
  - Adição unitária: **INCF**, **INCFSZ**;
  - Adições diversas: **ADDWF**, **ADDLW**.

INCF	**Increment f**
Syntax:	[*label*] INCF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) + 1 → (dest)
Status Affected:	Z
Encoding:	00 | 1010 | dfff | ffff
Description:	The contents of register 'f' are incremented. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'.
Words:	1
Cycles:	1
Example	`INCF REG1, 1` Before Instruction REG1 = 0xFF Z = 0 After Instruction REG1 = 0x00 Z = 1

INCFSZ	**Increment f, Skip if 0**
Syntax:	[*label*] INCFSZ f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) + 1 → (dest), skip if result = 0
Status Affected:	None
Encoding:	00 | 1111 | dfff | ffff
Description:	The contents of register 'f' are incremented. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'. If the result is '0', the next instruction, which is already fetched, is discarded. A NOP is executed instead making it a two-cycle instruction.
Words:	1
Cycles:	1(2)

Operações Aritméticas Básicas

Somando

ADDWF	Add W and f
Syntax:	[*label*] ADDWF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(W) + (f) → (dest)
Status Affected:	C, DC, Z
Encoding:	00 | 0111 | dfff | ffff
Description:	Add the contents of the W register with register 'f'. If 'd' is '0', the result is stored in the W register. If 'd' is '1', the result is stored back in register 'f'.
Words:	1
Cycles:	1
Example	`ADDWF REG1, 0` Before Instruction W = 0x17 REG1 = 0xC2 After Instruction W = 0xD9 REG1 = 0xC2 Z = 0 C = 0 DC = 0

Significado dos bits de Status:

- C = *Carry* ("vai um"):
  - C=1 → quando "vai um" na soma do MSB;
  - C=0 → quando não há "vai um" na soma do MSB.
- DC = *Digit Carry*
  - DC=1 → quando "vai um" na soma do 4º bit do byte;
  - DC=0 → quando não há "vai um" na soma do 4º bit do byte.
- Z = Zero
  - Z=1 → quando o resultado da operação lógica ou aritmética é zero;
  - Z=0 → quando o resultado da operação lógica ou aritmética não é zero.

Operações Aritméticas Básicas

Somando

ADDLW	**Add Literal and W**			
Syntax:	[*label*] ADDLW k			
Operands:	$0 \le k \le 255$			
Operation:	(W) + k → (W)			
Status Affected:	C, DC, Z			
Encoding:	11	111x	kkkk	kkkk
Description:	The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register.			
Words:	1			
Cycles:	1			
Example	ADDLW 0x15 Before Instruction W = 0x10 After Instruction W = 0x25			

Operações Aritméticas Básicas

Subtraindo

- Dois grupos de instruções:
  - Subtração unitária: **DECF**, **DECFSZ**;
  - Subtrações diversas: **SUBWF**, **SUBLW**.

DECF	**Decrement f**
Syntax:	[*label*] DECF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) - 1 → (dest)
Status Affected:	Z
Encoding:	00 | 0011 | dfff | ffff
Description:	Decrement register 'f'. If 'd' is '0'. the result is stored in the W register. If 'd' is '1', the result is stored back in register 'f'.
Words:	1
Cycles:	1
Example	DECF CNT, 1 Before Instruction CNT = 0x01 Z = 0 After Instruction CNT = 0x00 Z = 1

DECFSZ	**Decrement f, Skip if 0**
Syntax:	[*label*] DECFSZ f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	(f) - 1 → (dest); skip if result = 0
Status Affected:	None
Encoding:	00 | 1011 | dfff | ffff
Description:	The contents of register 'f' are decremented. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'. If the result is '0', the next instruction, which is already fetched, is discarded. A NOP is executed instead making it a two-cycle instruction.
Words:	1
Cycles:	1(2)

Operações Aritméticas Básicas

Subtraindo

SUBWF	Subtract W from f			
Syntax:	[*label*] SUBWF f,d			
Operands:	$0 \le f \le 127$ $d \in [0,1]$			
Operation:	(f) - (W) → (dest)			
Status Affected:	C, DC, Z			
Encoding:	00	0010	dfff	ffff
Description:	Subtract (2's complement method) W register from register 'f'. If 'd' is '0', the result is stored in the W register. If 'd' is '1', the result is stored back in register 'f'.			
Words:	1			
Cycles:	1			

O significado dos bits de Status mudam!

- C = $\overline{\text{Borrow}}$ ("pede emprestado"):
  - C=1 → quando NÃO "pede emprestado";
  - C=0 → quando "pede emprestado".
- DC = $\overline{\textit{Digit}\ \text{Borrow}}$
  - DC=1 → quando NÃO "pede emprestado" na subtração do 4º bit do byte;
  - DC=0 → quando "pede emprestado" na subtração do 4º bit do byte.
- Z = Zero
  - Z=1 → quando o resultado da operação lógica ou aritmética é zero;
  - Z=0 → quando o resultado da operação lógica ou aritmética não é zero.

Operações Aritméticas Básicas

Subtraindo

- C pode ser usado para determinar o sinal do resultado:
  - $C = 0$ → O resultado é negativo e é representado em complemento de 2;
  - $C = 1$ → O resultado é positivo.
- Se o resultado for zero tem-se $Z = 1$ e $C = 1$;
- Exemplos:

Example 1:

```
SUBWF   REG1, 1
```

Before Instruction

REG1 = 3
W = 2
C = ?

After Instruction

REG1 = 1
W = 2
C = 1; result is positive
DC = 1
Z = 0

Example 2:

Before Instruction

REG1 = 2
W = 2
C = ?

After Instruction

REG1 = 0
W = 2
C = 1; result is zero
Z = DC = 1

Example 3:

Before Instruction

REG1 = 1
W = 2
C = ?

After Instruction

REG1 = 0xFF
W = 2
C = 0; result is negative
Z = DC = 0

Operações Aritméticas Básicas

Subtraindo

SUBLW	**Subtract W from Literal**
Syntax:	[*label*] SUBLW k
Operands:	$0 \le k \le 255$
Operation:	k - (W) → (W)
Status Affected:	C, DC, Z
Encoding:	11 | 110x | kkkk | kkkk
Description:	The W register is subtracted (2's complement method) from the eight-bit literal 'k'. The result is placed in the W register.
Words:	1
Cycles:	1

Example 1:
```
SUBLW   0x02
```
Before Instruction

W = 1
C = ?

After Instruction

W = 1
C = 1; result is positive

Example 2:

Before Instruction

W = 2
C = ?

After Instruction

W = 0
C = 1; result is zero

Example 3:

Before Instruction

W = 3
C = ?

After Instruction

W = 0xFF
C = 0; result is negative

Comparações

- Comparar se duas variáveis têm o mesmo valor:

Código típico em linguagem c:

```
if (x = y)
{
    //Código a ser executado se x = y
}
```

Como pode ser implementado em Assembly:

```
    MOVF    Y,W     ; CARREGA Y EM W
    XORWF   X,W     ; COMPARA X COM W(=Y)
    BTFSS   STATUS,Z
    GOTO SAIR_IF
;
    ;CÓDIGO A SER EXECUTADO SE X = Y
;
SAIR_IF
    ;PRÓXIMAS INSTRUÇÕES
```

Comparação usado a função XOR bit a bit. Se todos os bits forem iguais o resultado do XOR é zero e o *flag* Z é setado (Z = 1)

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

Comparações

- Comparar se uma variável é maior que outra:

Código típico em linguagem c:

```c
if (x > y)
{
    //Código a ser executado se x > y
}
```

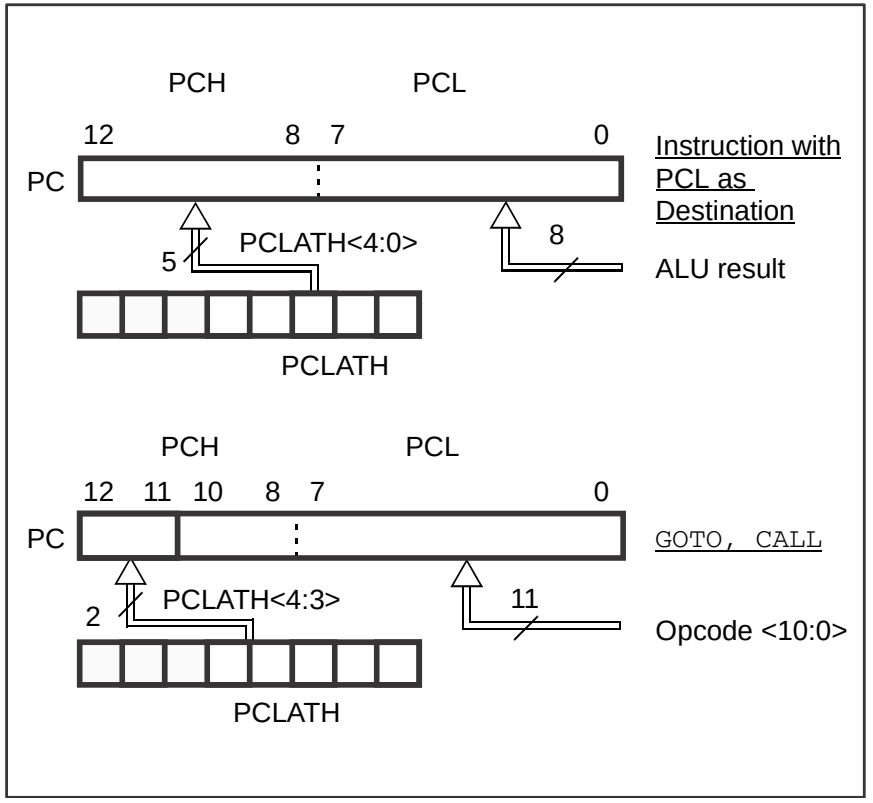
Como pode ser implementado em Assembly:

```
    MOVF    X,W     ; CARREGA Y EM W
    SUBWF   Y,W     ; SUBTRAI: W = Y - W(=X)
    BTFSC   STATUS,C    ; QUANDO O RESULTADO É NEGATIVO, C = 0
    GOTO SAIR_IF
;
    ;CÓDIGO A SER EXECUTADO SE X > Y
;
SAIR_IF
    ;PRÓXIMAS INSTRUÇÕES
```

Operando Diretamente com o registrador *Program Counter*

- O registrador *Program Counter* (**PC**) tem 13 bits de comprimento;
- O byte menos significativo do **PC** vem do registrador **PCL**, que é do tipo escrita/leitura;
- A parte alta do **PC** vem do registrador **PCLATH**;

Diferentes formas de carregar o PC



Operando Diretamente com o PC

- Se o **PCL** for alterado diretamente, muda-se o ponto de execução do programa;
- Deve-se ter cuidado para que o PCL não vá para um ponto desconhecido do programa;
- Só é possível avançar 255 posições (em relação ao endereço atual) na memória de programa usando o PCL.

Exemplo de Uso do PCL

- Usando o **PCL** para escolhe entre várias rotinas:

A variável TECLA recebe um valor entre 0 e 7, sendo que de 1 a 6 representa uma tecla que foi pressionada.

```
TRATA_TECLA             ; Rotina de tratamento da tecla
    MOVLW   B'00000111'
    ANDWF   TECLA,W     ; Mascara TECLA para limitar o valor em 7
    ADDWF   PCL,F       ; Soma TECLA ao PCL para pular para o ponto certo
    GOTO    SEM_TECLA   ; TECLA = 0 -> Não há tecla pressionada
    GOTO    TECLA1      ; TECLA = 1 -> Pula para a rotina TECLA1
    GOTO    TECLA2      ; TECLA = 2 -> Pula para a rotina TECLA2
    GOTO    TECLA3      ; TECLA = 3 -> Pula para a rotina TECLA3
    GOTO    TECLA4      ; TECLA = 4 -> Pula para a rotina TECLA4
    GOTO    TECLA5      ; TECLA = 5 -> Pula para a rotina TECLA5
    GOTO    TECLA6      ; TECLA = 6 -> Pula para a rotina TECLA6
    GOTO    ERRO        ; TECLA = 7 -> Não existe
```

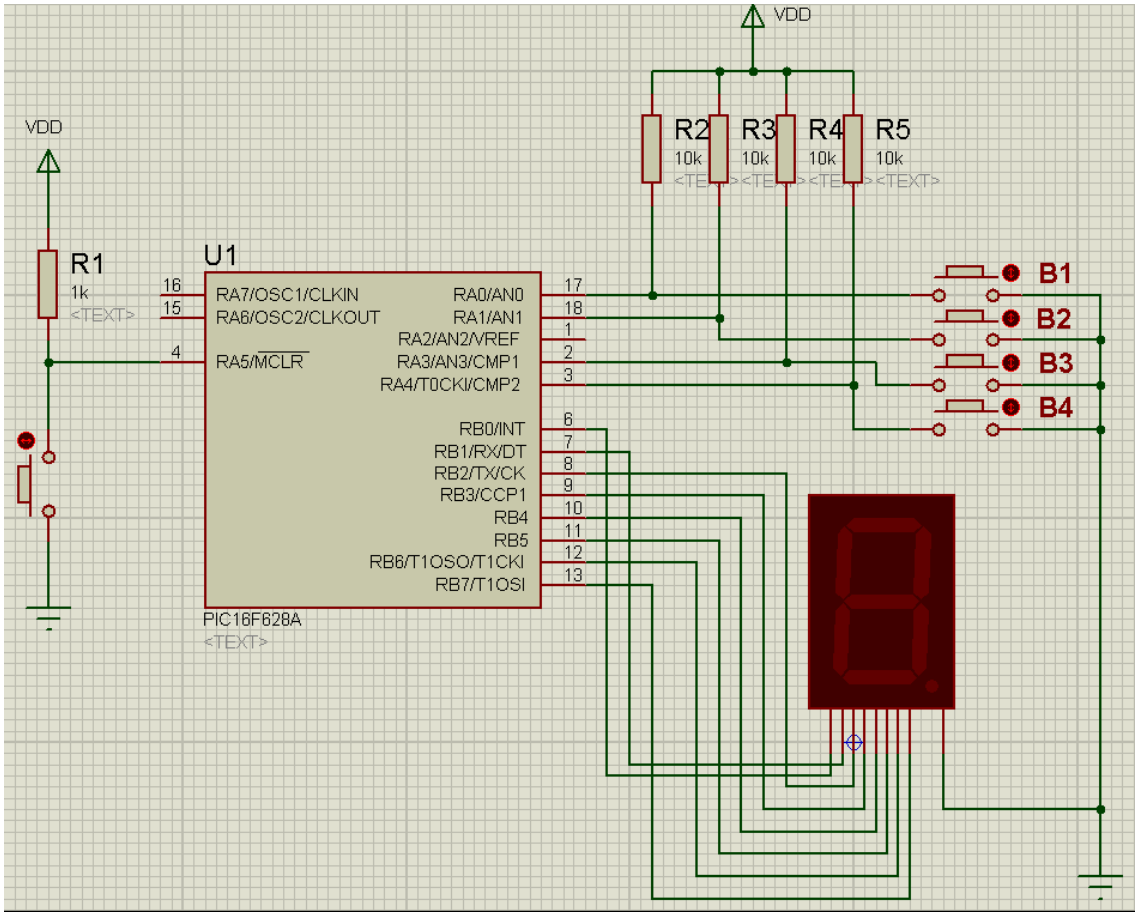
Exemplo de Uso do PCL

- Usando o **PCL** para montar uma tabela de valores:

```
CONVERTE
    MOVF    CONTADOR,W  ;COLOCA CONTADOR EM W
    ANDLW   B'00001111' ;MASCARA VALOR DE CONTADOR
                ;CONSIDERAR SOMENTE ATÉ 15
    ADDWF   PCL,F

            ;'EDC.BAFG' ; POSIÇÃO CORRETA DOS SEGUIMENTOS
    RETLW   B'11101110' ; 00 - RETORNA SÍMBOLO CORRETO 0
    RETLW   B'00101000' ; 01 - RETORNA SÍMBOLO CORRETO 1
    RETLW   B'11001101' ; 02 - RETORNA SÍMBOLO CORRETO 2
    RETLW   B'01101101' ; 03 - RETORNA SÍMBOLO CORRETO 3
    RETLW   B'00101011' ; 04 - RETORNA SÍMBOLO CORRETO 4
    RETLW   B'01100111' ; 05 - RETORNA SÍMBOLO CORRETO 5
    RETLW   B'11100111' ; 06 - RETORNA SÍMBOLO CORRETO 6
    RETLW   B'00101100' ; 07 - RETORNA SÍMBOLO CORRETO 7
    RETLW   B'11101111' ; 08 - RETORNA SÍMBOLO CORRETO 8
    RETLW   B'01101111' ; 09 - RETORNA SÍMBOLO CORRETO 9
    RETLW   B'10101111' ; 10 - RETORNA SÍMBOLO CORRETO A
    RETLW   B'11100011' ; 11 - RETORNA SÍMBOLO CORRETO b
    RETLW   B'11000110' ; 12 - RETORNA SÍMBOLO CORRETO C
    RETLW   B'11101001' ; 13 - RETORNA SÍMBOLO CORRETO d
    RETLW   B'11000111' ; 14 - RETORNA SÍMBOLO CORRETO E
    RETLW   B'10000111' ; 15 - RETORNA SÍMBOLO CORRETO F
```

Exercício

Circuito:



Programa:
Botão 1: Mostra 1
Botão 2: Mostra 2
Botão 3: Mostra 3
Botão 4: Mostra 4