TECNOLOGIA DO PIC
- 2006 -

Sétima Edição

FUNDAÇÃO INSTITUTO TECNOLÓGICO DE OSASCO

- ENGENHARIA -

- Disciplinas: -

SISTEMA DE PROGRAMAÇÃO

e

MICROPROCESSADORES

ALUNO : _____

Professor Eng. Paulo Takashi Miyatake

OBJETIVO DESTA PUBLICAÇÃO

Tem o propósito de dar um incentivo aos iniciantes no mundo da eletrônica microprocessada, e também aos que tem interesse em obter mais conhecimentos deste maravilhoso mundo, que é o do microcontrolador.

Acreditamos que é suficiente que um técnico conheça apenas um dos vários microprocessadores utilizados nos diversos aparelhos eletrônicos, que já servirá de "startup" para o desenvolvimento desta modalidade tecnológica.

Com o rápido desenvolvimento da eletrônica, a cada dia se torna difícil ignorar esta tecnologia, que está incorporada em quase todos os aparelhos que nos trazem confortos.

Não seria nenhum exagero se afirmarmos que o grau de conhecimento de um engenheiro eletrônico está ligado intimamente ao domínio do conhecimento dos microprocessadores.

Neste aprendizado, é imprescindível que o interessado conheça as noções básicas do compilador MPLAB da Microchip, e também tenha o domínio do Windows.

SOBRE O AUTOR

Paulo Takashi Miyatake é técnico em eletrotécnica, e também é formado em engenharia eletrônica e em telecomunicações.

Trabalha desde 1.989 na empresa Medical Cirúrgica Ltda, no setor da eletromedicina, onde vem desenvolvendo diversos aparelhos aliados à tecnologia, que atendem as normas internacionais de qualidade como a NBR-IEC/INMETRO; BPF e a ISO 9000.

É, também consultor técnico da empresa Digimatic Consultoria e Projetos S/C Ltda, onde é diretor desde 1.982, e vem prestando consultoria nas áreas de eletromedicina, eletrônica e telecomunicações.

.

Na F.I.T.O. - Fundação Instituto Tecnológico de Osasco, leciona as seguintes disciplinas : Medidas em Eletrônica, Informática e Eletrônica Digital, para os alunos do curso Técnico, e as disciplinas : Sistema de Programação e Microprocessadores, para os alunos do curso de Engenharia Eletrônica da FAC-FITO.

paulot@netpoint.com.br

Capítulo	Índice	Página

1- O QUE É "PIC"

É a marca registrada do fabricante *Microchip Tecnology Inc* dos Estados Unidos da América. Não se sabe ao certo o seu significado. Provavelmente: **P**eripheral **I**nterface **C**ontroller, ou **P**rogramable **I**ntegrated **C**ircuit.
Ao contrário dos microprocessadores, como por exemplo: 8085 ou Z80, que possuem mais de 100 instruções "set", o microcontrolador da linha "PIC", tem aproximadamente 35 instruções set de 14 "bits" cada.
Utiliza uma tecnologia conhecida como RISC (**R**educed **I**nstruction **S**et **C**omputer) que significa Computador com Set de Instruções Reduzido. O seu oposto é o CISC (**C**omplex **I**nstruction **S**et **C**omputer) que significa Computador com Set de Instruções Complexa.
Na nossa vida cotidiana, deparamos com diversos aparelhos que utilizam tais microcontroladores. Vejamos: no videocassete, no forno de microondas, no telefone, nos brinquedos, no controle remoto da televisão, no controle de ignição e alarme do carro, nos aparelhos de som, no mouse do seu computador, nos elevadores, etc. Vide a figura na página 79.

2 - A DIFERENÇA ENTRE mP E O mC

O hardware (conexões entre periféricos) baseado num m**P** (microprocessador), apresenta, externamente, conjunto de memórias RAM's e EPROM's, e possuem um Data Bus (via de dados) e um Address Bus (via de endereços).
Um "hardware" composto por um m**C** (microcontrolador) não apresenta nenhuma espécie de memória externa (há situações em que é possível e necessário utilizar memórias do tipo serial), e não possuem nenhum tipo de barramento (via de dados ou de endereços).
Esta simplificação se explica no fato de o mesmo possuir um "hadware" composto por memórias RAM's e EPROM's que ficam dentro da pastilha, e por possuir um mínimo de instruções "set" que facilitam a elaboração de projetos.

3 - AS MEMÓRIAS

RAM : (**R**andom **A**ccess **M**emory) Memória Programável do tipo volátil, ou seja, ao desligar a sua alimentação, perde os dados nela contidos.

PROM : (**P**rogrammable **R**ead **O**nly **M**emory) Memória programável apenas uma vez. Não é volátil.

EPROM : (**E**rasable **P**rogrammable **R**ead **O**nly **M**emory) Memória do tipo não volátil, reprogramável. Possui uma janela no seu topo para incidir uma luz ultravioleta que permite apagá-la.

EEPROM : (**E**lectrically **E**rasable **PROM**) Semelhante ao EPROM, porém é possível reprogramá-la, regravando sobre a informação anterior. Tem a inconveniência de custar

mais caro, porém nos modelos de "PIC" que utilizam este tipo de memória, é possível instalar memória externa que possibilita escrever ou ler dados no modo serial.

EEPROM FLASH : É um tipo de memória não volátil, contida nos PIC's que são identificados pela letra "**F**", como é o caso do PIC16**F**84.

4 - O MICROCONTROLADOR "PIC"

COM O PROPÓSITO DE FACILITAR AS EXPLICAÇÕES DESTA APOSTILA, VAMOS CONSIDERAR APENAS CARACTERÍSTICAS DO **PIC 16F84**, OPERANDO EM UMA FREQÜÊNCIA DE **4 MHz**. PORTANTO, QUALQUER REFERÊNCIA AO "**PIC**", ENTENDE-SE QUE SE TRATA PARTICULARMENTE DO **PIC 16F84**, daqui em diante, denominado apenas como **PIC**.

5 - A ALIMENTAÇÃO

Embora o manual do fabricante recomende utiliza-lo em 5Vcc, é possível alimentá-lo em uma tensão de 2,0 a 6,0 V, desde que seja uma tensão fixa, ou seja sem variações.
Um capacitor de 100 nF, fixado próximo aos pinos 14 e 5, evita que o microcontrolador sofra influência de ruídos externos.

6 - A FREQÜÊNCIA

O PIC aceita desde uma tensão contínua (freqüência zero) até, em alguns modelos, 20 Hz. (com sufixo A) . O PIC 16F84 aceita uma freqüência de oscilação de até 10 MHz.
O PIC permite ser programado para gerar a sua própria oscilação que resultará no sinal clock interno, ou então de receber sinal clock, vindo de um circuito externo.
Para se obter um freqüência de oscilação, devemos optar por uma das três técnicas:

1. **Circuito RC** – É o resultado de uma oscilação de um conjunto resistor/capacitor, e embora seja uma solução barata, não há precisão.
2. **Circuito RESSOADOR** – Embora seja uma técnica barata, o ressoador cerâmico tem uma estabilidade superior ao RC.
3. **Cristal Osciladora** – Trata-se de uma técnica alta de precisão, tão imprescindível para circuitos que requerem estabilidade como relógios, frequencímetros, cronômetros, etc.

7 - O CICLO DE MÁQUINA

O PIC divide a sua freqüência de oscilação (PIC 16F84 = 4 MHz) por quatro. Portanto, no caso do PIC16F84, o seu ciclo de máquina é de 1microsegundo (1μs) se utilizado um cristal de 4 MHz.

A palavra "**Clock**" utilizado em projetos microprocessados, refere se à freqüência de oscilação em que a CPU irá operar. Depende de cada modelo ou do tipo de microprocessador.
Evidentemente, quanto maior for o clock utilizado, mais veloz será o sistema projetado, e a escolha deverá ser avaliada pelo projetista.

8 - O PIC 16F84

Microcontrolador da família PIC da *Microchip* possui 1 Kbytes de memória EEPROM FLASH para programar; 68 bytes de RAM; 64 bytes de EEPROM para dados; 4 Interrupções: 1 Timer; 1 Watch Dog; 1 Prescaler e 13 Ports de I/O.

A sua memória de programação, da família FLASH, permite ser apagada e regravada automaticamente pelo gravador, sem a necessidade de apaga-la com a lâmpada ultravioleta.

9 - AS CARACTERÍSTICAS ELÉTRICAS DO PIC16F84

Voltagem de trabalho ..... 2,0 V até 6,0 V
Voltagem máxima (de Vdd a Vss) .....– 0,3 V até 7,5 V
Voltagem máxima nas demais entradas/saídas ..... –0,6 V até (Vdd+0,6 V)
Voltagem máxima no MCLR ..... –0,3 V até 7,5 V
Corrente máxima de saída do PORTA ..... 50 mA
Corrente máxima de entrada do PORTA ..... 80 mA
Corrente máxima de saída do PORTB .....100 mA
Corrente máxima de entrada do PORTB ..... 150 mA
Temperatura de operação ..... –55 ^{o}C até 125 ^{o}C
Temperatura de armazenamento ..... –65 ^{o}C até 150 ^{o}C
Freqüência de operação máxima ..... 10 MHz
Corrente de consumo típico em 5,0 V/4 MHz ..... < 2 mA
Corrente de consumo a 2,0 V/32 kHz ..... 60 µA
Corrente de consumo em "standy by" (modo SLEEP) ..... 26 µA

NOTAS :

- Todos os "bits" referentes ao PORTB, têm comportamentos de um circuito TTL. Com a exceção do bit RB0 que, quando estiver configurado como entrada para interrupções, o mesmo atuará com características de uma porta Schmidt Trigger.

- Todos os "bits" do PORTA, quando configurados como entradas, têm comportamento de um circuito TTL, com a exceção do bit RA4/TOCKI que se transforma num Schmidt Trigger, e como saída, num Dreno aberto (Open drain).

- Embora o microcontrolador PIC tenha um comportamento de um circuito TTL em suas saídas, trata-se de um "chip" de tecnologia da família C-MOS.

10 - O CONTADOR DE PROGRAMA

No PIC16F84 o contador de programa (**PC – P**rogram **C**ounter) "conta" iniciando de 000H até 3FFH, isto porque o mesmo possui apenas 1 Kbytes de memória EEPROM FLASH programável.
Qualquer referência que se possa ocorrer a uma posição superior a 3FFH, serão transladadas para posições inferiores a 400H. Por exemplo: 472H refere-se ao endereço 72H.
Ao levar o pino 4 (**MCLR\ -** Master Clear) ao nível de Vss (terra), o PIC obrigará o contador de programa a indicar a posição **000H** (posição inicial).
Quando o PIC for interrompido por software (TMR0; Watch Dog; fim de escrita em EEPROM) ou por hardware (RB0/INT; PORTB), o contador de programa sempre indicará a posição **004H**.
Os 8 "bits" menos significativos do **PC,** que indica a linha que está sendo executada, ficam armazenados num registro chamado **PCL.** E os 5 "bits" mais significativos ficam armazenados num registro conhecido por **PCLATH**.

11 - A PILHA (STACK)

Quando o PIC é solicitado para realizar uma subrotina através do comando do tipo CALL, o endereço seguinte que estava sendo executado (**PC+1**), será armazenado nesta pilha. Isto permite que o PIC memorize o endereço em que deverá retornar após executar uma subrotina.
Durante uma execução da subrotina, o PIC ao "encontrar" uma instrução do tipo "**RETURN**" automaticamente retornará para continuar a executar o programa que havia sido interrompido pela subrotina.
Esta capacidade de memorizar o endereço de retorno é de 8 níveis (8 CALL's), ou seja, caso ocorra de executar mais de 8 subrotinas consecutivas, o mais antigo endereço de retorno será perdido.

12 - O ENDEREÇO DE INTERRUPÇÃO

Embora já se tenha comentado a este respeito, vale salientar que toda as vezes que o PIC sofre uma interrupção, seja por software ou hardware, o contador de programa PC irá direcionar para o endereço **004H**. Este endereço é denominado **vetor de interrupção**.

13 - OS REGISTROS ESPECIAIS

Nos inícios dos endereços da RAM, encontra-se os registros especiais conhecidos por **FSR**'s (**F**ile **S**election **R**egister) que é utilizada para armazenar variáveis e registros utilizados pelos programas. Estes registros serão enfatizados mais à diante.

14 - OS ENDEREÇOS DE MEMÓRIA EEPROM (PROGRAMÁVEL)

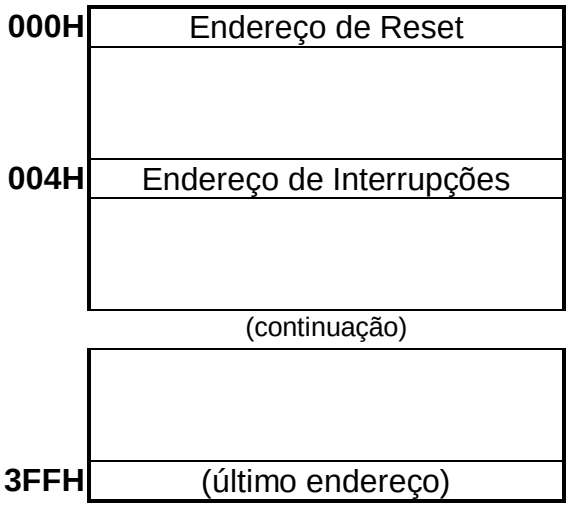


Tabela de mapa de memória programável (EEPROM)

15 - O CONCEITO DE "SET" E "CLR" OU "RESET"

Todas as vezes que se referir ao comando "**SET**', entende-se que se trata de nível lógico "**1**". O seu oposto é o comando "**CLR**" (resumo da palavra "clear"` = limpar) ou "**RESET",** que, obviamente, se trata de um nível lógico "**0**".
Portanto, quando alegamos que é preciso "**setar**" um bit, entende-se que é para mudar esse bit para nível lógico "**1**" O seu oposto seria : "**zera**r"; "**limpar**" ou "**resetar**"

16 – A MEMÓRIA DE DADOS (RAM)

NOTA : NO PIC16F84, A ÁREA DE USO GERAL NA RAM, VAI DO ENDEREÇO **0CH** ATÉ 0 ENDEREÇO **4F**, TOTALIZANDO 68 BYTES.

A RAM é utilizada para memorizar todas as variáveis e registros que são utilizados pelo programa. Esses dados formados por 8 bits, por estarem numa memória do tipo volátil, ao desligarmos o dispositivo, serão perdidos.
No PIC16F84, elas são divididos em 2 bancos denominados de **BANCO 0** e **BANCO 1**, e, para poder acessa-los, necessitam serem "**setados**" e "**resetados**" alternadamente.

Por exemplo : Estando no Banco 0, e para mudar o PORTB para saída de dados, será necessário acessar o BANCO 1 , onde fica o registro TRISB que precisa ser zerado.

```
BSF     STATUS, RP0     ; Seta o Banco 1 (RP0 = 1)
MOVLW   00H             ; Carrega w com 00H
MOVWF   TRISB           ; Converte PORTB para saída
BCF     STATUS, RP0     ; Volta para o Banco 0 (RP0 = 0)
```

BANCO 0 (BCF STATUS,RP0)		
00H	INDF	Endereçamento indireto
01H	TMR0	Registro de contagem do timer 0
02H	PCL	Parte baixo do PC
03H	STATUS	Registro de STATUS
04H	FSR	Ponteiro para endereçamento indireto
05H	PORTA	Registro dos pinos do PORTA
06H	PORTB	Registro dos pinos do PORTB
07H	----	Não implementado
08H	EEDATA	Dado lido ou gravado na EEPROM
09H	EEADR	Endereço para ler ou gravar na EEPROM
0AH	PCLATH	Parte alta do PC
0BH	INTCON	Registro do INTCON
BANCO 1 (BSF STATUS,RP0)		
80H	INDF	Endereçamento indireto
81H	OPTION_REG	Registro OPTION_REG
82H	PCL	Parte baixo do PC
83H	STATUS	Registro de STATUS
84H	FSR	Ponteiro para endereçamento indireto
85H	TRISA	Se TRISA = 1 entrada; se TRISA = 0, saída
86H	TRISB	Se TRISB = 1 entrada; se TRISB = 0, saída
87H	------	Não implementado
88H	EECON1	Controle da EEPROM
89H	EECON2	Controle da EEPROM
8AH	PCLATH	Parte alta do PC
8BH	INTCON	Registro do INTCON

Tabela de Mapa de Memória na RAM

OBS : Os Registros que aparecem nos dois Bancos têm os endereços em duplicata. Por exemplo : O endereço do PCL = 02H (BANCO 0) ou 82H (BANCO 1)

17 - O REGISTRO STATUS

Bit 7							Bit 0
IRP	**RP1**	**RP0**	**TO**	**PD**	**Z**	**DC**	**C**

Após o RESET : **00011XXX**

Endereço : 003H ou 083H

X = Indefinido

IRP – Seleciona Banco para endereçamento indireto
R/W **Manter em zero para o PIC16F84**

RP1 - Seleciona Bancos para endereçamento direto
R/W **Manter em zero para o PIC16F84**

RP0 - Seleciona Bancos para endereçamento direto
R/W Se = **0**, seleciona **Banco 0**
Se = **1**, seleciona **Banco 1**

TO - Bit indicador (flag) do **Time-out**
R É = 1 após um power-up; instrução CLRWD ou SLEEP
É = 0 quando ocorre um Time-out (transbordo) do Watch Dog

Z - Bit indicador (flag) do **ZERO**
R Se = 1, indica que a última operação (lógica ou aritmética) resultou em zero
Se = 0, indica que a última operação (lógica ou aritmética) não resultou em zero

PD - Bit indicador (flag) do **Power-down**
R É = 1, após power-up ou instrução CLRWD
É = 0 pela execução do SLEEP

DC - Digit-Carry (transbordo)
R/W
Se = 1, ocorreu um transbordo do 3^o para o 4^O bit da última operação
Se = 0, não ocorreu o transbordo (carry-out) da última operação

C - Carry (transbordo)
R/W Se = 1, ocorreu um transbordo do 7^o bit da última operação
Se = 0, não ocorreu transbordo (carry-out) da última operação

NOTAS :

- Após um **RESET**, causa o retorno para o Banco 0 (**RP0 = 0**) e o (**TO**) - Timer-out e o (**PD**) - Power- down serão setados.

- Os bits indicadores (flags) **TO** e o **PD** , permitem apenas a sua leitura. Os demais, poderão ser modificados, setando ou zerando-os.

18 - O REGISTRO OPTION_REG

Bit 7							Bit 0
RBPU	**INTEDG**	**TOCS**	**TOSE**	**PSA**	**PS2**	**PS1**	**PS0**

Endereço = 081H Após o RESET : **11111111**

RBPU \ - Habilita os resistores Pull-up do PORTB
R/W 1 = Resistores Pull-ups desabilitados
0 = Resistores Pull-ups habilitados
INTEDG – Configuração da sensibilidade da borda do sinal de interrupção no RB0

R/W 1 = Interrupção ocorrerá na borda de subida
0 = Interrupção ocorrerá na borda de descida

TOCS - Configuração do incremento do TMR0
R/W 1 = TMR0 é incrementado externamente pela mudança do RA4/TOCKI
0 = TMR0 é incrementado internamente pelo clock da CPU

TOSE - Configuração da sensibilidade da borda do sinal para incrementar TMR0
R/W 1 = O incremento ocorrerá na borda de descida do pino RA4/TOCKI
0 = O incremento ocorrerá na borda de subida do pino RA4/TOCKI

PSA - Configuração do uso do Prescaler
R/W 1 = Válido o Prescaler do Watch Dog
0 = Válido o Prescaler do TMR0

O ajuste da taxa de Prescaler deverá ser feita segundo a tabela abaixo

PS2	**PS1**	**PS0**	***DIVISÃO TIMER***	***DIVISÃO WATCH DOG***
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

19 - O REGISTRO INTCON

Bit 7							Bit 0
GIE	**EEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**

Endereço = 0BH ou 8BH Após o RESET : **0000000X**
X = Indefinido

GIE - **G**robal **I**nterrupt **E**nable ou Habilitação Global
R/W 1 = Habilita as interrupções que estiverem selecionadas individualmente
0 = Desabilita todas as interrupções

EEIE - Interrupções que ocorrem ao fim de escrita na EEPROM
R/W 1 = Habilita esta interrupção
0 = Desabilita esta interrupção

TOIE - Interrupções causadas por transbordo (overflow) do TMR0
R/W 1 = Habilita esta interrupção
0 = Desabilita esta interrupção

INTE - Interrupção externa via RB0/INT (pino no. 6)
R/W 1 = Habilita esta interrupção
0 = Desabilita esta interrupção

RBIE - Interrupção por mudança de nível lógico no PORTB
R/W 1 = Habilita esta interrupção
0 = Desabilita esta interrupção

TOIF - Indica o transbordo (overflow) do TMR0
R/W 1 = Indica que ocorreu o transbordo do TMR0
0 = Indica que ainda não ocorreu o transbordo do TMR0

INTF - Indica a interrupção externa através do pino do RB0/INT
R/W 1 = Indica que ocorreu um pedido externo de interrupção (RB0/INT)
0 = Não ocorreu nenhum pedido de interrupção externa

RBIF - Indica a interrupção por mudança de nível lógico no PORTB
R/W 1 = Indica que pelo menos um bit do PORTB mudou de nível lógico
0 = Não houve nenhuma alteração no nível lógico nos bits do PORTB

NOTAS:

- Os bits indicadores (flags) TOIF, INTF e RBIF deverão ser zerados via software, logo após as respectivas interrupções.

- Quando ocorre uma operação de RESET, o bit GIE será zerado

- Durante o atendimento de uma interrupção, o GIE será automaticamente zerado pelo CPU, e após o término desta, será reabilitado. Portanto, jamais ocorrerá o encravamento de duas interrupções.

- A instrução de retorno RETFIE é uma instrução set de retorno de uma subrotina que reabilita o GIE, setando-o novamente.

- Nunca utilize a instrução RETFIE para retorno de subrotinas que não seja uma conseqüência de uma interrupção.

- Sempre que estiver executando uma instrução de escrita na EEPROM, por precaução, deveremos desabilitar momentaneamente o GIE. Não se esqueça de reabilitá-lo, ao término desta tarefa.

- Mesmo com o GIE = 0 e o T0IE = 0, o bit indicador (flag) T0IF será setado sempre que o TMR0 sofrer o transbordo (overflow). Este bit permanecerá setado até que seja zerado via software (instrução BCF INTCON, T0IF)

20 - O WATCH DOG

Num chi PIC existe um contador cujo clock é independente da máquina, resultado de um conjunto RC interno, que incrementa automaticamente.

Conhecido por "Cão de guarda", este timer, quando estiver habilitado (**WDT_ON**) na chave do **__CONFIG**, a cada 18 ms aproximados (para prescaler 1:1), irá causar o RESET, o contador de programa indicará a posição **000H,** e o bit **TO** do registro STATUS será setado.
O propósito deste recurso é para não permitir que o programa, por um falha qualquer, entre em um loop sem fim (trave). Nesta situação, ao superar o intervalo do timer Watch dog, o mesmo causará o reinicio do sistema (PC = 000H).

Portanto, estando o Watch Dog habilitado, é necessário que o mesmo seja resetado em intervalos inferiores ao do seu transbordo. Basta utilizar o comando **CLRWDT** regularmente.

A outra instrução que obriga o "resetamento" do timer do Watch Dog é o SLEEP que faz o PIC entrar no modo Power-down (standy by).

NOTAS:

- **O Watch dog timer só poderá ser ativado na gravação do chip, e não poderá ser desativado por software.**

- O Watch Dog é um timer cujo o clock é independente da máquina, porisso continuará atuando, mesmo no modo SLEEP (standy by).

- Para aumentar o intervalo de resetamento do timer Watch Dog, utilize o prescaler que inicia com taxa de divisão de 1:1 até 1: 128. Por exemplo , utilizando a taxa de 1:128 fica : 128 vezes 18 ms = 2,304 s.

- Recomenda-se não gerar reset do timer Watch Dog utilizando as interrupções periódicas, pois, existem casos em que o programa fica travado e atendendo normalmente as interrupções. Nesta situação, o timer Watch Dog não sofrerá transbordo, e, consequentemente, o programa continuará travado.

- Só habilite o timer Watch Dog, caso o software permita resetá-lo periodicamente, dentro do intervalo previsto.

- Se o prescaler estiver habilitado para Watch Dog, é possível ajusta-lo em : 1:1 = 18 ms ou até ao máximo de 1:128 (128 X 18 ms = 2.304 ms), em valores aproximados, 2,3 segundos.

- O timer Watch Dog varia em função da temperatura , da tensão de alimentação e do processo de fabricação do chip.

21 - O USO DO PULL-UPS

Pull-ups são os resistores que estão fixados internamente nos bits do PORTB ao Vdd configurados como entradas.
Para que isto ocorra, será necessário que o bit RBPU\ do registro OPTION_REG esteja com o valor zero.
Quando ocorre uma operação de RESET, esta habilitação é anulada, pois, esse bit é setado automaticamente.

Os bits do PORTB que forem configurados individualmente como entradas, terão os seus pull-ups automaticamente habilitados, desde que o bit RPBU\ do registro OPTION_REG esteja zerado (habilitado).

22 - O USO DO POWER - DOWN (SLEEP)

Popularmente conhecido por SLEEP, faz com que o PIC entre num estado conhecido por "standy by", onde o seu consumo cai para aproximadamente 60 μA.
Tem como objetivo reduzir o consumo de energia, para poder utiliza-lo em sistemas portáteis à bateria. Lembrando que em uso normal, o seu consumo se elevaria para próximo de 7 mA.

Um exemplo típico de aplicação é o controle remoto de alguns aparelhos domésticos, como o da televisão, onde as pilhas têm longa durabilidade.

Para entrar neste modo de operação, basta inserir no programa a instrução set **SLEEP**. O "PIC" reconhecerá esta instrução, irá zerar o bit **PS**, setará o bit **TO** do registro de **STATUS**, e em seguida entrará em um estado de "hibernação".

Estando no modo SLEEP, o oscilador do PIC pára de funcionar; logo, o timer TMR0 não atuará, pois cessará a contagem, porém, é importante que o Watch Dog não esteja habilitado, do contrário, o mesmo será "acordado" a cada transbordo desse timer.

NOTAS:

- Uma instrução set SLEEP não interrompe a operação de escrita na EEPROM; ela é finalizada normalmente.

- As funções de entradas e de saídas dos PORT's A e B, manterão inalterados durante o modo SLEEP

- **Não ocorrerá o tranbordo do timer 0 (TMR0), porque não há o clock durante o modo SLEEP.**

Para o PIC sair do modo SLEEP, deverá ocorrer uma das três situações descritas abaixo:

- Um reset por hardware pelo pino 4 – Master Clear (MCLR\) levado ao nível zero.
- Resetamento por transbordo do timer Watch dog, caso esteja habilitado.

- Interrupção externa causada pelo pino 6 – RBO/INT, devido à alteração de nível lógico.

- Interrupção causada pelo fim operação de escrita na EEPROM.

NOTAS:

- Todas as interrupções descritas anteriormente não dependem do bit GIE do registro INTCON estar setado, porém, necessitam que as suas interrupções estejam habilitadas individualmente.

A instrução SLEEP apresenta dois modos diferentes de retorno da "hibernação" :

a) Estando o **GIE = 0**

O PIC ao ser "acordado" por uma das interrupções, executa a instrução que fica logo após a instrução SLEEP.
Tudo se passa como se a CPU estivesse retornando de uma subrotina que o obrigou a fazer uma pausa.

b) Estando o **GIE =1**

O PIC ao ser "acordado" por uma das interrupções, executa a instrução que fica logo após a instrução SLEEP, e só então, a CPU atenderá a interrupção presente.
Em caso de necessitar que a CPU atenda imediatamente a interrupção, sem que execute a instrução que está imediatamente após o SLEEP, basta inserir a instrução NOP, logo após a instrução SLEEP.

23 - O POWER-ON RESET (POR)

O PIC 16F84 possui internamente um sistema que gera um delay (PWRT) de 72 ms aproximados, a partir do final do POR (Power-on reset).
Portanto, o chip permanecerá resetado por um período de 72 ms, logo após ligar sua alimentação, para garantir o início do seu funcionamento, somente após a estabilização da tensão.

Um circuito mínimo para gerar o POR, consiste em conectar o pino 4 (MCLR\) ao 14 (Vdd), eliminando-se, assim, o circuito RC.

Estando o PIC habilitado no modo XT, LP ou HS, o timer OST (**O**scilador **S**tart-up **T**imer) é automaticamente acionado logo após o término do PWRT (**P**o**w**er **R**eset **T**imer), adicionando 1024 períodos de espera para sair do reset.
A combinação do OST e do PWRT, aliada a uma configuração semelhante ao do circuito exemplo (Página 80), garantirá um reset eficiente, mesmo para uma fonte sujeita às flutuações e ruídos.

24 - OS MOTIVOS DAS INTERRUPÇÕES

Para falarmos deste capítulo, é necessário recapitularmos as funções do registro conhecido por **INTCON**.

Bit 7							Bit 0
GIE	**EEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**

Endereço = 0BH ou 8BH

Após o RESET : **0000000X**
X = Indefinido

O microcontrolador PIC possui quatro sinalizadores (Flags) internos de interrupções:

1- INTF- Interrupção externa pelo pino RB0/INT
2- RBIF- Mudança de nível lógico nos pinos RB4 a RB7
3- Fim de escrita na EEPROM
4- TOIF - Transbordo (Overflow) do Timer 0

NOTAS:

- Quando ocorre um RESET, com a exceção do bit RBIF, todos os demais serão resetados.

- Todos os sinalizadores (Flags), após a sua sinalização, deverão ser resetados pelo próprio usuário.

Com a exceção da última, escrita na EEPROM que será analisada mais a diante, todos os demais bits de controle e de sinalização estão no registro INTCON.

O **GIE** tem a função semelhante a uma chave geral que permite validar as interrupções selecionadas. Portanto, para o PIC aceitar uma interrupção selecionada, será necessário o **GIE = 1**.

Enquanto o PIC estiver atendendo uma interrupção, o GIE estará zerado para a CPU não atender uma segunda interrupção. Ao término desta interrupção, automaticamente o GIE será setado novamente pela instrução RETFIE (**Ret**urn **F**rom **I**nterrupt **E**nable).

Durante a operação de escrita na EEPROM, é importante que o GIE esteja zerado para não ocorrer interrupções que causam erros de escrita.

Certifique-se de que o GIE esteja realmente desabilitado, e que o mesmo não foi momentaneamente zerado para atender uma interrupção.
Não basta apenas zerar o GIE e acreditar que o mesmo ficará zerado; execute a seqüência abaixo:

```
        GIE_ZERO:

(a)     BCF         INTCON, GIE     ; Desabilita as interrupções (GIE=0)

(b)     BTFSC       INTCON,GIE      ; Se GIE = 0 pula uma linha
(c)     GOTO        GIE_ZERO        ; Aguarda GIE ficar zero

(d)     (O programa prossegue.....) ;  Seqüência do programa
```

Caso o GIE esteja atendendo uma interrupção antes do (a), mesmo resetado, poderá voltar a assumir o valor igual a um no intervalo (a) e (b). Caso isso venha a ocorrer, o mesmo será resetado (novamente) através da repetição do (d).

Mesmo que o GIE = 0, e TOIE = 0, o bit do TOIF será setado sempre que o timer 0 sofrer um transbordo (Overflow). E assim permanecerá setado até que seja zerado pelo usuário.

Portanto, antes de habilitar uma interrupção, certifique se de que não há nenhuma interrupção anteriormente selecionada (pendente), pois um atendimento imediato pode não ser interessante.

24.1 - INTERRUPÇÃO EXTERNA VIA RB0/INT

Para que o pino 6 (RB0/INT) seja usado como entrada para sinal externo de interrupção, é necessário que os bits INTE e o GIE do registro INTCON estejam setados.

É necessário também, selecionar o sentido de atuação (se de 0 para 1 ou de 1 para 0) através do bit INTEDG do registro OPTION_REG.

Se INTEDG = 1, uma variação de 1 para 0 no RB0/INT fará o bit INTF = 1 (do registro INTCON), sinalizando um pedido de interrupção.

Se INTEDG = 0, uma variação de 0 para 1 no RB0/INT fará o bit do INTF = 1 (do registro INTCON), sinalizando um pedido de interrupção.

NOTA:

O RB0/INT, quando usado com entrada para interrupção externa, fica um Schmidt Trigger.

Para habilitar os resistores Pull-ups, basta zerar o bit correspondente a RPBU\ do registro OPTION_REG .

24.2 - INTERRUPÇÃO POR MUDANÇAS NO PORTB

Ao selecionar o bit RBIE = 1 no registro INTCON e deixando o GIE =1, uma mudança de estado em um dos bits RB4 a RB7, previamente selecionados como entradas, setará o bit RBIF do registro INTCON, causando uma interrupção.

Somente os pinos que estiverem configurados como entradas serão capazes de detectar as mudanças de estado, pois o anterior fica memorizado num latch.

Para habilitar os resistores Pull-ups, basta zerar o bit correspondente a RPBU\ do registro OPTION_REG .

24.3 - INTERRUPÇÃO DE FIM DE ESCRITA DA EEPROM

Se os bits EEIE e GIE do registro INTCON estiverem setados, todas as vezes que ocorre uma finalização de escrita na EEPROM, o bit EEIF do registro EECON1 será setado e ocorrerá a interrupção.

24.4 - INTERRUPÇÃO DO TIMER 0

Para que o Timer 0 cause uma interrupção, será necessário que os bits TOIE e GIE do registro INTCON estejam setados.

Ao ocorrer o transbordo (Overflow) do contador TIMER 0, o bit T0IF do registro INTCON será setado e ocorrerá a interrupção caso o GIE = 1.

O bit sinalizador TOIF será setado sempre que ocorrer o transbordo (Overflow) do timer 0, não dependendo do T0IE e GIE estarem setados ou não.

O bit sinalizador T0IF do registro INTCON ao ser setado pelo transbordo deste timer, deverá ser resetado pelo usuário. Do contrário, assim permanecerá até ocorrer um RESET.

25 - O RESET E AS SINALIZAÇÕES (FLAGS)

Convém relembra-lo de que:

- Todas as vezes que ocorre o **RESET**, o contador de programa PC (**P**rogram **C**ounter) "vai" para a posição de endereço **000H.**

- Todas as vezes que ocorre uma **interrupção**, o contador de programa PC (**P**rogram **C**ounter) "vai" para a posição de endereço **004H**.

- As respectivas interrupções só ocorrerão se estiverem selecionadas, e se o **GIE = 1**.

- Os bits sinalizadores (Flags) serão setados mesmo que o GIE = 0, embora não ocorra nenhuma interrupção.

- Os sinalizadores INTF, RBIF, EEIF e T0IF deverão ser zerados pelo usuário antes de executar a instrução RETFIE, para permitir uma nova interrupção.

- Somente a instrução **RETFIE**, usada para o retorno de uma subrotina que faz o tratamento de interrupções, é que **restabelece** o **GIE = 1**.

- Um RESET via pino 4 (MCLR\) ou um POR (ao ligar a alimentação) deixa sempre o GIE e os demais bits do registro INTCON resetados, com exceção do RBIF que assume um valor indefinido (X).

- O BANCO 0 será reabilitado automaticamente após um RESET.

- Após um RESET, o PRESCALER selecionado será do Watch Dog na taxa de divisão de 1:128, e os resistores Pull-Ups do PORT B serão desabilitados.

- Os PORTA e PORTB ficam configurados como entradas, sempre que ocorre um RESET.

- O RESET ainda afetará:

a) Interrupções externas, aceitas na subida dos pulsos.

b) Timer 0 com sinal de clock externo, contando na descida do sinal no pino RA4/TOCKI.

26 – O PRESCALER

Trata-se de um registrador de 8 bits, que permite divisões de até 256 vezes na freqüência do sinal de entrada.

A taxa de divisão dependerá de escolher entre “Divisão Timer” e “Divisão Watch Dog”. Vide capítulo 18.

O ajuste do Prescaler é muito útil, por exemplo, quando se deseja alterar o intervalo de resetamento do Watch Dog. Assim, é possível ajusta-lo desde 18ms até 2,3 s.

Todos os ajustes do Prescaler são feitos no registro OPTION_REG, porém, o registro onde ocorre o incremento da contagem não é acessível ao usuário.

Quando for desejável que não ocorra nenhuma divisão do sinal para o Timer 0, basta selecionar o Prescaler para Watch Dog, na taxa 1:1.

Este registro será zerado sempre que efetuar uma escrita qualquer no registro TMR0.

27 – O USO DO TIMER 0

PIC16F84 possui internamente um Contador/Temporizador de 8 bits conhecido por Timer 0.

No modo Contador, a cada pulso na entrada pino 3 (RA4/TOCKI) este contador se incrementará.
Quando ocorre o transbordo (Overflow) de 255 para 0, o hardware seta o TOIF gerando uma solicitação de interrupção do Timer 0.

No modo Timer, a cada ciclo de instrução em que ocorre a freqüência Fosc/4, este contador se incrementará.
Quando ocorre o transbordo (Overflow) de 255 para 0, o hardware seta o TOIF gerando uma solicitação de interrupção do Timer 0.

Se resetarmos o bit TOSC = 0 do registro OPTION_REG, o Timer 0 será incrementado a cada ciclo de instrução. Por exemplo : para 4 MHz, o incremento ocorrerá a cada 1 micro-segundo.

Nesta situação, o incremento ocorrerá livre e de forma contínua, sem a possibilidade de nenhum controle.

NOTA:

- Quando é escrito um novo valor do registro TMR0, ocorrerá um atraso de 2 ciclos de máquina, que deverá ser levado em consideração na elaboração do software.

28 – O ENDEREÇAMENTO INDIRETO

Um registro conhecido por **FSR** (**F**ile **S**election **R**egister), localizada na posição 00H da memória RAM, tem o objetivo de direcionar uma instrução set, através de um endereço variável (registro) conhecido por **INDF**.

Por exemplo: Deseja-se limpar uma posição da memória RAM cujo endereço é 30H.

Logo :

```
MOVLW   30H      ; Carrega W com o valor 30H
MOVWF   FSR      ; Copia o valor de W em FSR (30H)
CLRF    INDF, F  ; Limpa a posição 30H da memória RAM
```

Observe que os registros FSR e INDF não sofrem nenhuma alteração após a operação.

29 – O EEPROM DE DADOS

NOTA: O PIC 16F84 POSSUI UMA ÁREA DE 64 BYTES NO EEPROM DE DADOS, QUE VAI DO ENDEREÇO **00H** ATÉ O ENDEREÇO **3FH.**

PIC16F84 possui 64 bytes de espaço na memória EEPROM, onde é possível gravar dados que necessitam ser mantidos, mesmo que a alimentação seja interrompida.
Um exemplo típico de aplicação seria os ajustes utilizados em televisões, videocassetes, rádios, etc, onde há necessidade de memorizar os ajustes como : cor; som; estação; canais; etc.

A memória EEPROM se assemelha a uma memória EPROM, onde se grava eletricamente os dados, que não se perdem numa eventual falta de energia elétrica. Tem a vantagem sobre a segunda, por não necessitar apagá-la com a exposição à luz ultra-violeta, antes de iniciar a regravação.

Portanto, podemos destacar como principal vantagem de se utilizar memórias EEPROM, pelo fato de o mesmo permitir o seu apagamento (eletricamente) sem a necessidade de retira-lo do soquete.

A gravação nesta EEPROM apresenta alguns aspectos técnicos indesejáveis. A primeira, seria a sua velocidade de gravação que é bem inferior a da memória RAM que leva apenas 0,4 ms, contra esta, que demora 10 ms (a 10 MHz).
Uma operação de leitura na EEPROM, a 4 MHz, leva apenas um ciclo de máquina que corresponde a 1 micro segundo.

Estes 64 bytes de espaços de EEPROM, ao contrário da RAM, só são endereçáveis através de 4 registros específicos.
São eles :

- **EEADR** : Endereço da EEPROM, onde se pretende ler ou escrever.
- **EEDATA** : Dado para escrever ou ler.
- **EECON1** : Registro do Controle 1
- **EECON2** : Registro do Controle 2

Vamos entender melhor esses registros, acima mencionados:

29.1 - EEADR - Endereço da EEPROM

É um registro de 8 bits localizada no endereço **09H** da memória RAM, possibilita escolher **256 posições** para ler ou escrever.

Portanto, é nesse registro que iremos escrever o endereço da EPROM onde se deseja ESCREVER ou LER um dado.

29.2 - EDITA – Dado da EEPROM

O dado para ESCREVER na EEPROM, cuja posição já está definida no EEADR, deverá ser escrito neste registro de 8 bits, que fica na posição 08H da RAM.

Da mesma forma, para LER um dado da EEPROM, cuja posição já está definida no EEADR, deverá ser lido neste registro de 8 bit, que fica na posição 08H da RAM.

29.3 - EECON1 - REGISTRO 1 DE CONTROLE DA EEPROM

Trata-se de um registro, localizado em 88H, com apenas 5 bits implementados, e é o responsável pelo controle das operações de escrita e leitura na EEPROM.

Bit 7							Bit 0
----	**----**	**----**	**EEIF**	**WRERR**	**WREN**	**WR**	**RD**

Endereço = 88H Após o RESET : **0000X000**
X = Indefinido

Os bits 5, 6 e 7 não são implementados, são interpretados como "0"

EEIF - Bit sinalizador de fim de escrita da EEPROM
R/W 1 = Já ecreveu na EEPROM (precisa ser zerado pelo usuário)
0 = Ainda não terminou de escrever na EEPROM

WRERR - Bit sinalizador de erro de escrita da EEPROM
R/W 1= Interrompido por Reset ou Watch Dog
0 = Operação de escrita realizada com sucesso.

WREN – Bit sinalizador de habilitação de escrita na EEPROM
W/R 1 = Autorizado a escrever na EEPROM
0 = Não autorizado a escrever na EEPROM

WR – Bit sinalizador de início de escrita na EEPROM
(*) 1 = Começa a escrita de 1 ciclo (o usuário deverá zerar ao fim da escrita)
0 = Fim da operação de escrita

RD – Bit sinalizador de leitura na EEPROM
(*) 1 = Começa a leitura na EEPROM (é zerado ao fim da leitura automaticamente)
0 = Não começa a operação de leitura.

NOTAS:

- Como o processo de escrita e de leitura (*) não ocorre instantaneamente, os bits RD e WR só permitem seta-los, não podem ser zerados pelo software, isso ocorrerá automaticamente ao término da operação.

- Convém recordar que, após uma finalização na escrita na EEPROM, o bit EEIF do registro EECON1 será setado, gerando o pedido de interrupção caso o os bits EEIE e GIE estiverem setados.

- bit sinalizador (Flag) EEIF do registrador EECON1 deverá ser zerado através do software antes de encerrar a rotina de interrupção.

- **Durante a ESCRITA na EEPROM, as interrupções devem ser desligadas para evitar conflitos (GIE = 0)**

Para executar uma operação de ESCRITA, basta carregar o EEDATA com o valor da constante, preencher o EEADR com o endereço da posição da memória EEPROM, e setar o bit WR.

Para evitar que ocorra uma operação de escrita indesejável, é recomendável que o bit WREN esteja zerado pelo software, e setado somente quando for gravar algum dado. Ao término da gravação, não devemos nos esquecer de zerar novamente este bit.

NOTA : O bit WREN não afeta a operação de leitura na memória EEPROM.

Durante uma operação de ESCRITA, caso ocorra um Reset via pino 4 (MCLR\) ou por causa do Watch Dog, a mesma não se realizará com sucesso, e assim o bit WRERR será setado indicando um erro na execução de ESCRITA.

Somente num Reset, cuja origem não foi uma queda de energia elétrica, após a constatação do bit WRERR (erro na escrita), o programa poderá reiniciar a gravação, já que os dados contidos em EEADR e EEDATA não sofrem danos.

29.4 – EECON2 - REGISTRO 2 DE CONTROLE DA EEPROM

Este registrado, localizado em 89H, não necessita de nenhum ajuste, porém é utilizado para o processo de ESCRITA na EEPROM. Veja no exemplo abaixo

A título de exemplificação, valemos de um programa simplificado que escreve uma constante 35H no endereço 28H da EEPROM.

```
        BCF STATUS, RP0         ; Seleciona o Banco 0 (EEADR e EEDATA)
        MOVLW  28H              ; Carrega W com o endereço 28H
        MOVWF  EEADR            ; Carrega EEADR com o valor de W
        MOVLW  35H              ; Carrega W com a constante 35H
        MOVWF  EEDATA           ; Carrega EEDATA com o valor de W
        BSF    STATUS, RP0      ; Seleciona o Banco 1 (EECON1 e 2)
        BCF    INTCON, GIE      ; Desabilita  o GIE (Global)
        BSF    EECON1, WREN     ; Permite a escrita na EEPROM

(1)     MOVLW  55H              ; Carrega W com 55H
(2)     MOVWF  EECON2           ; Carrega EECON2 com o valor de W
(3)     MOVLW  0AAH             ; Carrega W com AAH (não esquecer o "0" !)
(4)     MOVWF  EECON2           ; Carrega EECON2 com o valor de W
(5)     BSF    EECON1,WR        ; Seta o WR para escrever

        BCF  EECON1, WREN       ; Desabilita a escrita na EEPROM (segurança)
        ( O programa continua..... )
```

As seqüências de instruções de 1 a 5 (em negrito) são necessariamente obrigatórias, sem as quais o PIC não reconhecerá o processo de escrita.

Os 64 bytes da memória de dados EEPROM estão mapeados à partir do endereço inicial de 00H a té 63H.

29.4– O RECONHECIMENTO DE FIM DE ESCRITA NA EEPROM

Conforme já descrito, o processo de escrita na EEPROM com um clock de 10 MHz leva aproximadamente 10 ms e o de leitura só leva um ciclo de máquina, que é de 0,4 ms.
Como o PIC, num intervalo de 10 ms, tem condições de efetuar aproximadamente 25.000 instruções para um clock de 10 MHz, a maneira ideal de se reconhecer o FIM de ESCRITA da EEPROM seria:

- Habilitar todas as interrupções (GIE = 1) e zerar o bit EEIF do registro EECON1, logo após a o peração de escrita.

- Ocorrendo uma interrupção, o bit EEIF do registro EECON1 deverá ser testado, se foi proveniente de um FIM de ESCRITA de EEPROM.

Assim, no exemplo anterior, fica :

```
(4)   MOVWF  EECON2          ; Carrega EECON2 com o valor de W
(5)   BSF  EECON1,WR         ; Seta o WR para escrever

      BCF  EECON1, WREN      ; Desabilita a escrita na EEPROM

      BSF    INTCON, GIE     ; Habilita todas as interrupções (GIE = 1)

      (O programa continua .... )
```

E, para executar uma operação de LEITURA, basta carregar o EEADR com o endereço da posição da memória EEPROM, setar o bit RD no registro EECON1, e ler o conteúdo do EEDATA.

Exemplificando: Deseja-se ler um dado contido na memória EEPROM, no endereço 35H. Este dado deverá ser salvo, posteriormente, na memória RAM, cujo o endereço destinado é 50H.

```
BCF    STATUS, RP0     ; Seleciona o Banco 0  (EEADR)
MOVLW  35H             ; Carrega W com endereço 35H
MOVWF  EEADR           ; Copia o valor de W  em EEADR
BSF    STATUS, RP0     ; Seleciona o Banco 1 (EECON1)
BSF    EECON1, RD      ; Seta o bit de leitura (RD)
```

```
BCF    STATUS, RP0        ; Seleciona o Banco 0 (EEDATA)
MOVF   EEDATA, W          ; Copia em W o dado lido no EEDATA
MOVWF  50H                ; Escreve o valor de W em RAM 50H

(O programa continua..... )
```

NOTA S:

- O processo de leitura na memória EEPROM (RD = 1), leva apenas um ciclo de máquina, após o qual, se resetará automaticamente.

- Se esta operação for uma subrotina, **jamais reabilite o GIE**, pois, após o término desta, ocorrerá a reabilitação automática do GIE devido a presença do RETIFIE.

30 – OS PORT'S COMO ENTRADA/SAÍDA

O PIC possui 13 PORST'S, que permitem ser configurados cada um dos seus bits como entrada ou saída.

Esta configuração é feita nos registros **TRISA** e **TRISB** que ficam no **BANCO 1,** de tal modo que :

- **Estando qualquer um dos bits do TRISA ou TRISB "setado", o mesmo atuará como ENTRADA.**

- **Estando qualquer um dos bits do TRISA ou B "resetado", o mesmo atuará como SAÍDA**

30.1 – OS REGISTROS TRISA E TRISB

É importante frisar que os registros TRISA e TRISB estão localizados no BANCO 1, portanto, para configurar o PORT, seja A ou B, é necessário, antes, seleciona-lo.

O procedimento de ESCRITA no TRIS A ou B, para que o PORT A ou B seja configurado como ENTRADA ou SAÍDA, consiste em modificar as funções dos bits , presentes.

Por exemplo: Deseja-se configurar os bits do PORTB, No. 1; 2; 4 e 5 em saídas, e os demais em entradas.

O número hexadecimal correspondente para esta configuração será : 36H (B'**00110110**').

Portanto, considerando que o PORTB está configurado, inicialmente como ENTRADA, ao efetuar uma instrução do tipo :

```
BSF   STATUS, RP0        ; Seleciona o BANCO1 (TRISB)

MOVLW  36H               ; Carrega W com uma constante 36H
MOVWF  TRISB             ; Configuras os bits do PORTB em E/S (36H)

BCF   STATUS, RP0        ;  Retorna para o BANCO 0
```

Após estas instruções, o PORTB assumirá, conforme o valor 36H (B '**00110110**'), onde os bits " **0** " representam saídas, e os bits " **1** ", entradas.
Assim permanecerá até que ocorra uma nova instrução que obrigue o PORTB a assumir uma outra configuração.

3				**6**			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 SAÍDA	**0 SAÍDA**	**1 ENTRADA**	**1 ENTRADA**	**0 SAÍDA**	**1 ENTRADA**	**1 ENTRADA**	**0 SAÍDA**

PORT B após as instruções anteriores.

31 – A DIRETRIZ __CONFIG

No endereço 2007H da memória de programa do PIC16F84, está implementado um conjunto de 14 bits, onde devemos gravar a configuração de trabalho do PIC.
Esses bits são acessíveis somente durante os processos de gravação ou de verificação. Enquanto o PIC estiver apagado, todos esses bits ficam em nível "1".

NOTAS:

- Uma operação de RESET não altera os bits do __CONFIG (gravado na EEPROM)

- O usuário não precisará se preocupar com os endereços, e nomes do __CONFIG, pois os gravadores do PIC permitem escolher apenas as opções desejadas e efetuarão a gravação automaticamente.

- Caso o software não contenha o __CONFIG, no início do processo de gravação, será solicitado automaticamente a sua definição.

__CONFIG

Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CP	**CP**	**CP**	**CP**	**CP**	**CP**	**CP**	**CP**	**CP**	**CP**	**PWRTE**	**WDTE**	**FOSC1**	**FOSCO**

Bit 13 – 4 **CP** – Bit de Proteção de Código
1 = Código está desprotegido (default)
0 = Código está PROTEGIDO

Bit 3 **PWRTE \ -** Bit de Habilitação do POWER-UP TIMER
1 = Power-Up habilitado
0 = Power-Up desabilitado

Bit 2 **WDTE** – Bit de habilitação do Watch Dog
1 = Watch Dog HABILITADO
0 = Watch Dog desabilitado

Bit 1 (**FOSC1**)	Bit 0 (**FOSC0**)	Bits de seleção modo de Oscilação
1	1	Modo RC EXTERNO
1	0	CRISTAL ou RESSONADOR (HS)
0	1	CRISTAL ou RESSONADOR (XT)
0	0	CRISTAL de baixa Potência (LP)

RC - Oscilador resultante do conjunto resitor/capacitor externos.
HS - Cristal ou ressonador de alta velocidade, superior a 4 MHz.
XT - Cristal ou ressonador de baixa velocidade, de 0,1 MHz a 4 MHz.
LP – Cristal de baixa potência, cuja frequência é menor a 0,2 MHz.

Portanto, a sintaxe da diretriz do __CONFIG fica :

__CONFIG _CP_OF*F* & _PWRTE_ON & _WDT_OFF & _XT_OSC

Vamos entender o significado da sintaxe acima:

__CONFIG : Palavra chave que define as demais. (tem **2 traços** subscritos)

_CP_OFF : Código de Proteção **desligado** (para PIC16f84 é indiferente)

_PWRTE_ON : Power-Up Timer **ligado**

_XT_OSC : Oscilador do tipo XT (**Cristal de 4 MHz**)

_WDT_OFF : Watch Dog **desligado**

Nota : É necessário **manter um espaço** entre cada instrução, acima descrita

32 – A GRAVAÇÃO DE IDENTIFICAÇÃO (ID)

O PIC possui 4 bytes (2000H a 2003H), embora não sejam acessíveis, permitem gravar uma identificação do chip. Mesmo com o código de proteção ativado, esta identificação pode ser lida, facilitando-se assim o rastreamento da versão deste software.
A gravação é feita apenas na parte baixa destes 4 bytes, portanto só está disponível um total de 16 bits.

O usuário não necessitará se preocupar com detalhes desta gravação de ID (identificação), pois os gravadores de PIC providenciam uma janela com um campo para ser preenchido. Durante a gravação do programa, automaticamente ela será gravada.

Por exemplo: O meu ID = **2E14** = **0010 1110 0001 0100**
2 E 1 4
Ela ficará assim :

2000H	XXXXXXXXXX**0010**	(2H)
2001H	XXXXXXXXXX**1110**	(EH)
2002H	XXXXXXXXXX**0001**	(1H)
2003H	XXXXXXXXXX**0100**	(4H)

33 – A REPRESENTAÇÃO DE UMA CONSTANTE

O usuário poderá representar uma constante, dentro de um assembler do PIC, das seguintes formas : Decimal, Binário, Hexadecimal ou ASCII.

Exemplificando, o número quarenta e dois pode ser representado nas seguintes formas:

- DECIMAL : **D‘42’**, **d‘42’, .42** ou simplesmente, **42** (para padrão “ Radix dec”)
- BINÁRIO : **B‘01000010’**
- HEXADECIMAL : **2AH**, **0x2A** ou **H‘2A’**
- ASCII : **A‘B’** (Representa a letra B, cujo equivalente em hexadecimal = 42H).

NOTAS:

- **Todas as constantes em hexadecimais iniciadas por letras (A ,B, C, D, E e F) deverão ser precedidas por zero. Exemplo : 0FAH.**
- Em todos os exemplos aqui apresentados, o padrão do compilador será DECIMAL (“**Radix dec**”), logo, qualquer constante sem a forma definida será considerada uma representação decimal.

34 – O CONJUNTO DE INSTRUÇÕES SET DO PIC

O PIC16F84 é composto por um conjunto de 35 Instruções Set, todas formadas por palavras de 14 bits.

Os compiladores MPASM e MPASMWIN reconhecerão as seguintes convenções:

CAMPO	*DESCRIÇÃO*
f	Registro compreendido de 0 a 127 (0 a 7FH)
w ou W	Registro W (**W**ork)
b	Bit do registro utilizado pela operação (0 à 7)
k	Constante ou Label (00H – FFH)
d	Se d = 0 ou W, o resultado da operação é armazenado em W Se d = 1ou f, o resultado da operação fica no próprio registro **Se "d" não for indicado, o resultado é armazenado em W**

CONJUNTO DE INSTRUÇÕES SET DO PIC

Operações de 1 Byte

OPERANDO		*DESCRIÇÃO*	*CICLOS*	*BITS AFETADOS*
ADDWF	f,d	Soma W e f	1	C,DC,Z
ANDWF	f,d	AND entre W e f	1	Z
CLRF	f	Zera f	1	Z
CLRW		Zera W	1	Z
COMF	f,d	Complementa f	1	Z
DECF	f,d	Decrementa f	1	Z
DECFSZ	f,d	Decrementa f e pula se f =0	1 (2)	---
INCF	f,d	Incrementa f	1	Z
INCFSZ	f,d	Incrementa f e pula se f = 0	1 (2)	---
IORWF	f,d	OR entre W e f	1	Z
MOVF	f,d	Move o f	1	Z
MOVWF	f	Move o W para f	1	---
NOP		Não executa nada	1	---
RLF	f,d	Roda à esquerda pelo Carry	1	C
RRF	f,d	Roda à direita pelo Carry	1	C
SUBWF	f,d	Subtrai W de f	1	C,DC,Z
SWAPF	f,d	Troca os NIBLES em f	1	---
XORWF	f,d	XOR entre W e f	1	Z

CONTINUAÇÃO DAS INSTRUÇÕES DO PIC

Operações de 1 Bit

OPERANDO		*DESCRIÇÃO*	*CICLOS*	*BITS AFETADOS*
BCF	f',b	Zera o Bit "b" em f	1	---
BSF	f,b	Seta o Bit "b" em f	1	---
BTFSC	f,b	Se Bit "b" em f = 0, pula	1 (2)	---
BTFSS	f,b	Se Bit "b" em f = 1, pula	1 (2)	---

Operações com constantes e de Controle

ADDLW	k	Soma W com k	1	C,DC,Z
ANDLW	k	AND entre W e k	1	Z
CALL	k	Chama uma sbrotina	2	---
CLRWDT		Zera o timer do Watch Dog	1	TO\,PD\
GOTO	k	Desvia o programa para o k	2	---
IORLW	k	OR entre W e k	1	Z
MOVLW	k	Carrega W com a constante k	1	---
RETFIE		Retorna da interrupção (GIE = 1)	2	---
RETLW	k	Retorna com W = k	2	---
RETURN	k	Retorna de uma subrotina	2	---
SLEEP		Entra no modo SLEEP	1	TO\,PD\
SUBLW	k	Subtrai k de W	1	C,DC,Z
XORLW	k	XOR entre W e k	1	Z

NOTAS :

- Cada instrução set leva um ciclo de máquina (1 μs para uma freqüência de 4 MHz), porém, aquelas que estão condicionadas ao “pulo”, poderá levar 2 ciclos.

- Se numa instrução condicionada ao teste (Exemplo: BTFSC) for verdadeira, ocorrerá o “pulo”, logo demandará 2 ciclos de máquina.

35 – A PADRONIZAÇÃO DE UM PROGRAMA

Para elaborarmos um programa (software) do PIC, deveremos obedecer a seguinte padronização :

Exemplo :

```
LABEL     OPCODE   OPERANDO (S)        COMENTÁRIOS

LOOP :    MOVLW    50H          ; Carrega uma constante 50H em W
          MOVWF    PORTB        ; Copia em PORTB o valor de W
```

LABEL : (Pronuncia-se : "leibol")

- É o rótulo, é facultativo, e serve para indicar posições particulares do programa. O " **:** " colocado após o label não é obrigatório. Escreva sem nenhuma tabulação.

OPCODE :

- É o código de operação ou "mnemônico", sempre existirá, e indica qual a operação a ser executada pelo programa. Escreva a partir da 1º tabulação.

OPERANDO (S) :

- Só existirá se for necessário para a execução da instrução representado por OPCODE. Escreva a partir da 2º tabulação.

COMENTÁRIO :

- O comentário, embora facultativo, é importante para auxiliar o entendimento do programa. O compilador ignorará tudo que preceder o " **;** ". Escreva, preferencialmente, a partir da **3º tabulação**.

36 – OS DETALHAMENTOS DAS INSTRUÇÕES SET

ADDLW **k**	Soma ao W a constante k W = (W + k)	1 Ciclo
Flags afetados :	C, DC e Z	
Exemplo :	Se W = 01H ADDLW 22H fica : **W = 23H**	

ADDWF **f,d**	Soma ao W o valor de f d = (W + f)	1 Ciclo
Flags afetados :	C, DC e Z	
Exemplo :	Se W = 01H e f (10H) = 22H ADDWF 10H, f fica : **f = 23H**	

ANDLW **k**	AND entre W a constante k W = (W AND k)	1 Ciclo
Flags afetados :	Z	
Exemplo :	Se W = 33H e k = A6H ANDLW A6H fica : **W = 22H**	W = 0011 0011 (33H) k = 1010 0110 (A6H) **W = 00100010 (22H)**

ANDWF	**f,d**	AND entre W e f d = (W AND f)	1 Ciclo
Flags afetados :		Z	
Exemplo :		Se W = 33H e f (10H) = A6H ANDLW 10H,w fica : **W = 22H**	f (10H) = 10100110 (A6H) W = 00110011 (33H) **W = 00100010 (22H)**

BCF	**f,b**	Zera o bit "b"no registro f (**B**it **C**lear **F**ile)	1 Ciclo
Flags afetados :		---	
Exemplo :		Se PORTB = B'11111111' (FFH) BCF PORTB,7 fica : **PORTB = B'01111111' = 7FH**	

BSF	**f,b**	Seta o bit "b" no registro f (**B**it **S**et **F**ile)	1 Ciclo
Flags afetados :		---	
Exemplo :		Se PORTB = B'00001111' (0FH) BCF PORTB,6 fica : **PORTB = B'01001111' = 4FH**	

BTFSC	**f,b**	Pula uma linha do programa se o bit "b" no registro = 0	1 (2) Ciclo(s)
Flags afetados :		--- (**B**it **T**est **F**ile **S**kip if **C**lear)	
Exemplo :		Se o Port B = B'**0**1010001' (51H) **BTFSC PORTB,7** ; Se o bit 7 do PortB for = 0, pula uma linha. GOTO FIM ; (Como neste exemplo o bit 7 = 0, pula esta linha) CALL DELAY ; (O programa executa esta linha de instrução)	

BTFSS	**f,b**	Pula uma linha do programa se o bit "b" no registro = 1	1 (2) Ciclo(s)
Flags afetados :		--- (**B**it **T**est **F**ile **S**kip if **S**et)	
Exemplo :		Se o Port B = B'01010**0**01' (51H) **BTFSS PORTB,2** ; Se o bit 2 do PortB for = 1, pula uma linha. GOTO FIM ; (Como neste exemplo o bit 2 = 0, não pula esta linha) CALL DELAY ; (O programa não executa esta linha de instrução)	

CALL **k**	Chama uma subrotina cujo endereço é k	2 Ciclos

Flags afetados : ---

Exemplo :

```
                10H     MOVLW  35H
                11H     CALL   20H      ; Prossegue no endereço 20H
                12H     BCF DATA, 5
                (continua....)

:               20H     MOVWF  PORT A ; ( O programa continua aqui )
                21H     CLRF PORT B,F
                22H     RETURN          ; O programa retorna em 12H
```

CLRF **f**	Zera o registro f (**Clear F**ile)	1 Ciclo

Flags afetados : Z

Exemplo : Se PORTB = 57H
CLRF PORTB fica : **PORTB = 00H**

CLRW	Zera o registro W (**Clear W**)	1 Ciclo

Flags afetados : Z

Exemplo : Se W = 45H
CLRW fica : **W = 00H**

CLRWDT	Reseta o Watch Dog (**Clear W**atch **D**og **T**imer)	1 Ciclo

Flags afetados : T0\, PD\

Exemplo : A instrução CLRWDT deverá ser intercalada num intervalo menor que o do timer do Watch Dog, ao longo do programa.

COMF **f,d**	Os bits do registro f são complementados (invertidos)	1 Ciclo

Flags afetados : Z

Exemplo : Se PORTB = B‘11110000’ (F0H)
COMF PORTB,f fica : **PORTB = B‘00001111’ = 0FH**

DECF	**f,d**	Diminui em 1 unidade o valor do f	1 Ciclo
Flags afetados :	Z	(**Dec**rement **F**ile)	
Exemplo :	Se registro TIMER = 35H DECF TIMER,f fica : **TIMER = 35H**		

DECFSZ	**f,d**	Diminui em 1 unidade o valor de f, e pula uma linha se f = 0	1 (2) Ciclo(S)
Flags afetados :	Z	(**Dec**rement **F**ile **S**kip if **Z**ero)	
Exemplo :	Se registro TIMER = 35H **DECFSZ TIMER,f** ; Decrementa TIMER, se ficar = a zero, pula 1 linha GOTO FIM ; Como TIMER não zerou, o programa vai para FIM. CALL LOOP ; (Esta linha não será executada)		

GOTO	**k**	Desvia o programa para o endereço k	2 Ciclos
Flags afetados :	---		
Exemplo :	MOVLW 20H ; **GOTO FIM** ; (O programa vai para o Label chamado FIM) (continua...) **FIM** : CALL LOOP ; (O programa continua aqui)		

INCF	**f,d**	Aumenta em 1unidade o valor de f	1 Ciclo
Flags afetados :	Z		
Exemplo :	Se PORTB = F0H INCF PORTB,f fica : **PORTB = F1H** (f = f + 1)		

INCFSZ	**f,d**	Aumenta em 1unidade o valor de f, e pula 1 linha se f = 0	(2) Ciclo(s)
Flags afetados :	Z		
Exemplo :	Se PORTB = FFH fica: W = FFH + 1 = 00H **INCFSZ PORTB,w** ; Incrementa o valor de PORTB e guarda em W CALL LOOP ; (o programa não executa esta linha, pula) GOTO FIM ; (o programa pula para esta linha W = 00H)		

IORW	**k**	OR entre W e a constante k	(**I**nclusive **OR**)	1 Ciclo
Flags afetados :	Z			
Exemplo :	Se W = F0H e k = 1FH IORW F0H fica : **W = B‘11111111’ = FFH**			W = 11110000 k = 00011111 **W = 11111111**

IORWF	**f,d**	OR entre W e f	(**I**nclusive **OR**)	1 Ciclo
Flags afetados :	Z			
Exemplo :	Se W = F0H e LINE = 0FH IORWF LINE,f fica : **f = B‘11111111’ = FFH**			W = 11110000 f = 00001111 **f = 11111111**

MOVLW	**k**	O valor da constante k é copiado em W	1 Ciclo
Flags afetados :	---	(**Mov**e **L**iteral em **W**)	
Exemplo :	Se k = 25H MOVLW k fica : **W = 25H**		

MOVF	**f,d**	O valor de f é copiado em d	(**Mov**e **F**ile)	1 Ciclo
Flags afetados :	Z (Na operação MOVF TM1,f o Z = 1 se TM1 = 0) – Faz o teste do zero			
Exemplo :	Se TIM1 (f) = 35H e W = 00H MOVF TIM1,w fica **: W = 35H**			

MOVWF	**f**	O valor de W é copiado em f	(**Mov**e **W** em **F**ile)	1 Ciclo
Flags afetados :	---			
Exemplo :	Se W = 32H MOVWF PORTB fica : **PORTB = 32H**			

NOP	A instrução não executa nenhuma operação	1 Ciclo
Flags afetados :	--- MOVLW 50H	
Exemplo :	**NOP** ; Esta instrução atrasa 1 ciclo apenas CALL LOOP	

RETFIE	No final de subrotina, devido a interrupção, retorna com GIE = 1	2 Ciclos

Flags afetados : --- (**Ret**urn of **I**nterrupt for **F**lag **I**nterrupt **E**nable)

Exemplo :
```
MOWF  PORTB
(continua.... )
RETFIE          ; Retorna desta subrotina e habilita GIE.
```

RETLW k	No final de subrotina, retorna com W = k	2 Ciclos

Flags afetados : --- (**Ret**urn **L**iteral **W**)

Exemplo :
```
MOWF  PORTB
(continua...)
RETLW   20H     ; Retorna da subrotina com W = 20H
```

RETURN	No final de subrotina, retorna para o programa principal	2 Ciclos

Flags afetados : ---

Exemplo :
```
MOWF  PORTB
(continua...)
RETURN          ; Retorna desta subrotina (PC = PC + 1).
```

RLF f,d	Rotaciona o conteúdo de f 1 vez à esquerda pelo Carry	1 Ciclo

Flags afetados : C (**R**otate **L**eft through Carry **F**ile)

Exemplo : Seja PORTB = B‘01010001’ (51H)

RLF PORTB,w fica : **W = B‘10100010 (A2H)**

RRF f,d	Rotaciona o conteúdo de f 1 vez à direita pelo Carry	1 Ciclo

Flags afetados : C (**R**otate **R**ight through Carry **F**ile)

Exemplo : Seja PORTB = B‘01010010’ (52H)

RRF PORTB,w fica : **W = B‘00100001 (21H)**

SLEEP	Obriga o PIC a entrar em Standy By	1 Ciclo

Flags afetados : TO\ e PD\

Exemplo :
```
MOWF  PORTB
(continua...)
SLEEP           ; A CPU entra no modo SLEEP e o oscilador para.
```

SUBLW	**k**	Subtrai o conteúdo do W da constante k	W = (W – k)	1 Ciclo

Flags afetados : C,DC e Z (**Sub**tract **W** from **L**iteral)

Exemplo : Se W = 22H

SUBLW 20H fica : **W = 02H** W = (22H – 20H) = 02H

SUBWF	**f,d**	Subtrai o conteúdo do W do registro f	W = (W – f)	1 Ciclo

Flags afetados : C,DC e Z (**Sub**tract **W** from **F**ile)

Exemplo : Se W = 22H e PORTB = 21H

SUBWF PORTB,f fica : **f = 01H** W = (22H – 21H) = 01H

SWAPF	**f,d**	Troca os níveis (bits 7a 4 com bits 3 a 0)	1 Ciclo

Flags afetados : --- (**Swap** nibles of **F**ile)

Exemplo : Se PORTB = B'00001111' (0FH)

SWAPF PORTB,f fica : **f = B'11110000' (F0H)**

XORWF	**f,d**	XOR entre o valor de W e de f (Exclusive OR)	1 Ciclo

Flags afetados : Z

Exemplo : Se W = 22H e PORTB = 20H

XORWF PORTB,f fica : **f = 02H**

W = 00100010 (22H)
PORTB = 00100000 (20H)
f = 00000010 (02H)

XORLW	**k**	XOR entre o valor de W e a constante k (Exclusive OR)	1 Ciclo

Flags afetados : Z

Exemplo : Se W = 22H e k = 20H

XORLW k fica : **f = 82H**

W = 00100010 (22H)
k = 10100000 (A0H)
f = 10000010 (82H)

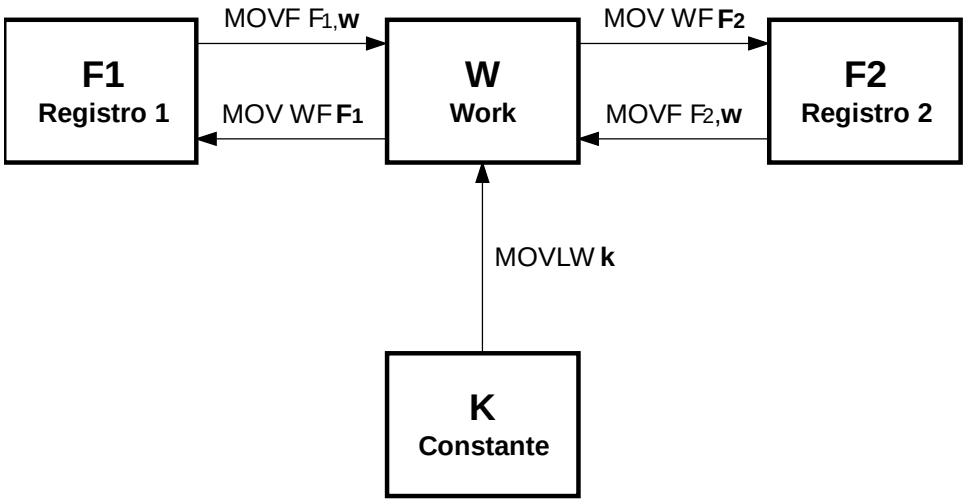
RECOMENDA-SE NÃO UTILIZAR AS DUAS INSTRUÇÕES ABAIXOS, PARA MANTER A COMPATIBILIDADE COM AS VERSÕES FUTURAS.
Estas instruções foram implementadas para compatibilizar os PIC da família 16C5X

OPTION e TRIS

37 - O FLUXO DE DADOS ENTRE REGISTROS E W

Para copiar um dado contido no registro F1 para outro registro F2, é necessário que este dado seja copiado antes no registro W, para depois repassa-lo no registro F2.

E para fazer a cópia de um valor constante em qualquer registro, é necessário, antes, copiá-lo em W.
Veja o fluxograma abaixo.



Exemplos:

1) Deseja-se copiar o valor lido no PORTB em um registro denominado PORTAL.

```
MOVF    PORTB,W   ; Copia o valor de PORTB em W
MOVWF   PORTAL    ; Copia o valor de W em PORTAL
```

2) Deseja-se carregar o registro TIME com o valor da constante 1FH.

```
MOVLW   1FH       ; Copia o valor da constante 1FH em W
MOVWF   TIME      ; Carrega o registro TIME com o valor 1FH
```

NOTAS **:**

- **Só é possível executar operações entre os registros F1 e F2, através do W**.

- **Só é possível carregar o valor da constante (k) em um registro F, através do W.**

Estas duas operações, acima mencionadas, com o recurso do MACRO, que iremos analisar mais a diante, tornam-se uma tarefa mais simplificada.

A partir dos próximos capítulos, iremos enfatizar assuntos específicos, que são imprescindíveis para elaborar e entender os programas de um projeto.

38 - O FORMATO DE UM PROGRAMA PADRÃO

O formato padrão para se editar um programa de um projeto à base de um microprocessador da linha PIC, segue o exemplo abaixo.

```
; (1) ================================================================
; TODOS OS PROGRAMAS NECESSITAM DE UM CABEÇALHO IGUAL A ESTE,
; ONDE ESCREVEMOS O TÍTULO; O OBJETIVO, O FUNCIONAMENTO E OS DETALHES
; QUE VOCÊ ACHAR IMPORTANTE.          USE SEMPRE LETRAS MAIÚSCULAS
; Autor : Paulo                Versão : 1.0                Data : 27/06/2000
; ================================================================
;(2)
        list p = 16F84
        Radix dec
        Include < PIC16F84.INC>

        __CONFIG  _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_ON

;(3) ---------- definições de constantes e espaços na RAM ----------

SAÍDA_1:    EQU     07          ; Referência ao bit 7 do PORTB
SAÍDA_2:    EQU     05          ; Referência ao bit 5 do PORTB

TMS1:       EQU     012         ; Timer 1  (1º posição da RAM)
TMS2:       EQU     013         ; Timer 2
W2:         EQU     014         ; Salva o W (anterior à interrupção)
STATUS2:    EQU     015         ; Salva do STATUS  (anterior à int.)

;(4) ---------- endereço do RESET ----------
; (O RESET OBRIGA  O PIC  A REINICIAR AQUI ! )

            ORG     00      ; Define o endereço inicial na EEPROM

            GOTO    INÍCIO  ; Endereça para o INÍCIO do programa

;(5) ---------- endereço das INTERRUPÇÕES ----------
; ( TODAS AS INTERRUPÇÕES OBRIGAM O PIC A REINICIAR AQUI ! )

            ORG     04          ; Endereço vetorizado das interrupções

;(6)         ---------- SALVA O STATUS E O W ----------

            MOVWF   W2          ; Salva o valor de W original em W2
            MOVF    STATUS,w    ; Copia o valor de STATUS original em W
            MOVWF   STATUS2     ; Salva o STATUS original em STATUS2

;(7)         ---------- FILTRA AS INTERRUPÇÕES ----------

            BTFSC   INTCON,T0IF ; Se T0 = 1, vai para T0_INT
            GOTO    T0_INT

            BTFSC   INTCON,INTF  ; Se INTF = 1, vai para EXT_INT
            GOTO    EXT_INT

            BTFSC   INTCON,RBIF ; Se RBIF = 1, vai para RB_INT
            GOTO    RB_INT

            BSF     STATUS,RP0  ; Vai para o BANCO1 (EECON1)
```

```
                BTFSC       EECON1,EEIF   ; Se EEIF = 1, vai para WREP_INT
                BCF         STATUS,RP0    ; Volta para o BANCO 0 (EECON1)

                GOTO        WREEP_INT

;(8)            ---------- RESTAURA STATUS E O W ----------

FIM_INT :

                MOVF        W2,w          ; Copia o valor de W original em W
                MOVF        STATUS2,w     ; Restaura o valor de STATUS original
                MOVWF       STATUS        ; Restaura o valor original de STATUS

                RETFIE                    ; Retorna da interrupção  e faz GIE = 1

;(9) ---------- INÍCIO DO PROGRAMA ----------

INÍCIO : (TODAS AS CONFIGURAÇÕES DE FUNCIONAMENTO DO PIC  SÃO ESCRITOS AQUI ! )


;(10)           ---------- HABILITA AS INTERRUPÇÕES ----------

                MOVLW       B'11111000'   ; Carrrega W com F8H
                MOVWF       INTCON        ; Habilita todas as interrupções

;(11)           ---------- HABILITA : AS ENTRADAS / SAÍDAS E O TIMER ----------

                CLRF        PORTB         ; Limpa os PORT A e B
                CLRF        PORTA         ; Evita acionamentos indesejáveis

                BSF         STATUS,RP0    ; Vai para  BANCO 1 (TRISA/B e OPTION_REG )
                MOVLW       B'00011111'   ; Carrega W com 1FH
                MOVWF       TRISA         ; Habilita PORTA como entrada

                MOVLW       B'00000001'   ; Carrega w com 01H (só o bit 0 = entrada)
                MOVWF       TRISB         ; Habilita PORTB como saída, menos o bit 0

                MOVLW       B'11011111'   ; Pull-ups desabilitados (bit 7 = 1)
                                          ; Int. Externa com RB0/INT na subida (bit 6 = 1)
                                          ; Clock interno (bit 5 = 0)
                                          ; (bit 4  é indiferente, pois bit 5 = 0)
                MOVWF       OPTION_REG    ; Prescaler Watch Dog  (Divide 1: 128)

                BCF         STATUS,RP0    ; Volta para o BANCO 0


;(12) ==========ROTINA PRINCIPAL ==========

PRINCIPAL:


;               (Neste espaço deverá conter o corpo principal do programa.)


;(13) ========== (FIM DA ROTINA PRINCIPAL) ==========
```

```
;(14) ---------- TRATAMENTOS DAS INTERRUPÇÕES ----------

;           ---------- INTERRUPÇÃO POR TIMER 0 ----------

T0_INT:
            BCF     INTCON,T0IF         ; Limpa o bit flag  TOIF
;           (Neste espaço deverá conter o tratamento referente a esta interrupção)
            GOTO    FIM_INT             ; Retorna para FIM_INT

;           ---------- INTERRUPÇÃO EXTERNA ----------

EXT_INT:
            BCF     INTCON,INTF         ; Limpa o bit flag INTF
            (Neste espaço deverá conter o tratamento referente a esta interrupção)
            GOTO    FIM_INT             ; Retorna para FIM_INT

;           ---------- INTERRUPÇÃO POR RB0 ----------

RB_INT:
            BCF     INTCON,RBIF         ; Limpa o bit flag RBIF
            (Neste espaço deverá conter o tratamento referente a esta interrupção)
            GOTO    FIM_INT             ; Retorna para FIM_INT

;           ---------- INTERRUPÇÃO POR FIM DE ESCRITA NA EEPROM ----------

WREEP_INT:
            BCF     EECON1,EEIF         ; Limpa o flag EEIF (já estava em BANCO1)
            (Neste espaço deverá conter o tratamento referente a esta interrupção)
            BCF     STATUS,RP0          ; Volta para o BANCO0
            GOTO    FIM_INT             ; Retorna para o FIM_INT

            END
```

38.1 – DETALHAMENTO DO PROGRAMA PADRÃO

O programa padrão apresentado anteriormente contém indicações que servem apenas para orientar este detalhamento. Estas indicações estão escritos em itálicos (letras inclinadas), portanto, não fazem parte integrante do programa.

Observe que no canto esquerdo, existem indicações numéricas que estão entre parênteses, e em negrito, que servirão como referências dos grupos de linhas de instruções.

(1) – CABEÇALHO

Escreva preferencialmente em letras maiúsculas, fazendo uma separação bem visível. Não se esqueça de colocar, no início de cada escrita ou símbolo, o sinal de " **;** " , que é imprescindível para o compilador desconsidera-lo.

É desejável que contenha o nome do autor; a versão e a data, que, futuramente, serão úteis para o rastreamento deste programa.

(2) - São informações importantes para o compilador.

list p = 16F84 : Define o modelo do processador 16CXX para o compilador.

Radix dec : Define como DECIMAL todos os números sem a indicação de base.

Include < PIC16F84.INC> : O compilador inclui o arquivo P16F84.INC

__CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_ON

__CONFIG : São fusíveis de **CONFIG**URAÇÕES internas, formados por 14 bits.

_CP_OFF : **C**ódigo de **P**roteção desprotegido

_WDT_OFF : **W**atch **D**og **T**imer desligado

_XT_OSC : Oscilador do tipo XT (crital ou ressonador de até 4 MHz)

_PWRTE_ON : **P**O**W**E**R** – on **T**imer **E**nable habilitado

NOTA : Após o "__CONFIG" e entre os demais, **não esquecer de deixar um espaço.**

(3) – Definições de constantes e espaços na RAM

A diretriz " **EQU** " não é uma instrução do PIC, porém, serve para o compilador atribuir um nome ao valor de uma constante, ou um nome a um endereço definido na RAM.

Desta forma, fica fácil alterarmos o valor da constante ou do conteúdo de um nome, cujo o endereço, já está definido na RAM através da diretriz "EQU".

- **SAÍDA_1: EQU 07** :
  Entende-se que o nome "SAÍDA_1" é um nome de registro, cujo valor, é igual a 07.

- **TMS1: EQU 012** :
  Entende-se que "TMS1" é um nome de registro, cujo endereço está na posição 12 (0CH é a primeira posição) da RAM.

(4) – Endereço de RESET

Todas as vezes que o PIC sofrer uma operação de RESET, o seu PC – Contador de Programa é **resetado**, ou seja, "**vai**" para o endereço inicial (**000H**).

Portanto, neste endereço (000H) deverá conter um comando do tipo "GOTO" que forçará o Contador de Programa – PC a saltar num endereço do tipo "INÍCIO".

O "**ORG**" não faz parte de instrução do PIC, mas permite ao usuário iniciar um trecho do programa em um endereço escolhido dentro da memória de programa.

Portanto, neste exemplo, temos o **ORG 00H** para o endereço do **RESET** e o **ORG 04** para o endereço das **INTERRUPÇÕES**.
Em ambos os casos, o compilador entenderá que estas instruções deverão ser escritas à partir à dos endereços indicados logo após a palavra "ORG".

(5) – Endereço das Interrupções

Quando o PIC recebe um pedido de interrupção, e se a mesma estiver individualmente e globalmente (GIE) habilitada, o seu PC – Contador de Programa saltará para o endereço 04H.

Vide a explicação do "ORG 04" no índice anterior (4).

(6) – Salva o STATUS e o W

Portanto, antes que o PIC atenda um pedido de interrupção, é necessário que o valor de W e de STATUS sejam salvos, pois, uma interrupção é imprevisível, e poderá ocorrer durante uma operação que esteja dependendo destes registros.

(7) – Filtra Interrupções

O PIC ao atender uma interrupção, deverá reconhecer qual o motivo desta interrupção, para depois tomar as providências cabíveis.

(8) – FIM_INT

É no FIM_INT (**FIM** de **INT**ERRUPÇÃO), que ocorrem duas coisas :

- Restaura os valores de STATUS e do W.
- Retorna à programação que havia sido interrompido, e restabelece GIE = 1 através da instrução RETFIE.

Ao término de uma subrotina causada por uma interrupção, o PC, contador de Programa "consultará" a pilha STACK e direcionará a continuação do programa, a partir de onde havia sido interrompido.

(9) - INÍCIO DO PROGRAMA

O PIC ao ser **RESETADO**, inicia a sua execução de programa com o PC – Contador de Programa endereçado em **000H.**

Como já havíamos visto, no endereço 000H, existe uma instrução "GOTO INÍCIO" que direciona o PIC a continuar no endereço identificado pelo label "INÍCIO"

(10) Habilita as Interrupções

Neste trecho, todas as configurações do PIC são definidos.
A primeira configuração é a habilitação das INTERRUPÇÕES., onde o registro INTCON é carregado com o valor F8H (B'11111000'), que habilita todos os modos de interrupções do PIC16F84.

(11) – Habilita as ENTRADAS/SAÍDAS e o TIMER

Por uma questão de segurança, inicialmente zeramos os PORT A/B

- BCF PORTA ; Limpa o PORTB para evitar acionamentos indesejáveis.
- BCF PORTB ; Limpa o PORTA para evitar acionamentos indesejáveis.

O PORTA é configurado, neste exemplo, como entradas.

```
BSF     STATUS,RP0   ; Vai para o BANCO1 (TRISA/B e OPTION_REG )
MOVLW   B'00011111'  ; Carrega W com 1FH
MOVWF   TRISA        ; Habilita PORTA como entrada
```

O PORTB é configurado saída, porém, de tal modo que o bit 0 permaneça como entrada,

```
MOVLW   B'00000001'  ; Carrega w com 01H (só o bit 0 = entrada)
MOVWF   TRISB        ; Habilita PORTB como saída, menos o bit 0
```

E, finalmente, são feitos os ajustes do registro OPTION_REG, onde ajustamos : Pull-Ups, RBO/INT, Clock interno/externo, Prescaler TMR0/WD e Divisão de prescaler.

```
MOVLW   B'11011111'  ; Pull-ups desabilitados (bit 7 = 1)
                     ; Int. Externa com RB0/INT na subida (bit 6 = 1)
                     ; Clock interno (bit 5 = 0)
                     ; (bit 4  é indiferente, pois bit 5 = 0)
MOVWF   OPTION_REG   ; Prescaler Watch Dog  (Divide 1: 128)
```

Após estes ajustes, é importante retornar para o BANCO 0.

(12) – ROTINA PRINCIPAL

Preferencialmente, utilize barras duplas para destacar este trecho de programação, pois, o mesmo é o corpo principal da programação.

Esta rotina, conhecida como "PRINCIPAL", é identificada por um label do mesmo nome. É nesta rotina que o PIC deverá ficar continuamente em loop, aguardando alguma solicitação (interrupção).

(13) – Vale a mesma a observação anterior, onde se recomenda utilizar barras duplas para diferenciar este trecho dos demais.

(14) – TRATAMENTO DAS INTERRUPÇÕES

Neste exemplo, existem 3 filtros que testam os bits flags de interrupções. Para cada um, existe uma rotina ser seguido. Porém, todos tem aspectos semelhantes, no tocante às seguintes necessidades:

- RESETAR o bit Flag que causou o pedido de interrupção.
- Executar as tarefas individuais (rotinas) através do comando "GOTO" ou "CALL".
- Ao término da execução da tarefa, se direcionam para o FIM_INT, que termina com a instrução RETFIE.

O programa termina com a instrução "END", que também não faz parte das instruções do PIC, apenas informa ao compilador o fim deste arquivo fonte.

39 – O USO DE UMA ROTINA

O PIC permite dois tipos de rotinas desvios, conhecidos popularmente por "pulos".

O primeiro utiliza a instrução "GOTO", onde o número de linha é substituído por um LABEL. Nesta rotina, a pilha STACK não memoriza o endereço de retorno (PC+1), portanto, ao término desta rotina, não haverá a possibilidade de um retorno automático.

Ao contrário do "GOTO", as rotinas chamadas por "CALL" memorizam na pilha STACK, o endereço de retorno (PC+1). Portanto, ao término desta, basta conter a instrução "RETURN" ou "RETLW", e ela será denominada de SUBROTINA.

NOTA:

SOMENTE EM UMA ROTINA DE INTERRUPÇÃO, APÓS O TÉRMINO DESTA, A INSTRUÇÃO "RETFIE" RESTAURA O GIE, E FAZ O RETORNO AUTOMÁTICO NO ENDEREÇO (PC+1).

39.1 – OS DESVIOS DO TIPO "GOTO $"

O símbolo "$" é visto pelo compilador como a linha em execução, portanto, são válidas as instruções descritas assim:

```
GOTO    $ + 0   ; O programa não salta, porém, executa 2 ciclos.
GOTO    $ + 1   ; Faz o programa saltar uma linha para baixo. (2 ciclos)
GOTO    $ + 3   ; Faz o programa saltar 3 linhas para baixo. (2 ciclos)
GOTO    $ - 2   ; Faz o programa saltar 2 linhas para cima. (2 ciclos)
```

40 - OS NOMES DAS ROTINAS (LABEL)

Os nomes utilizados em rotinas, conhecidos por LABEL, não podem conter espaços, portanto, utilize o sublinhado para compor um nome que faça algum sentido.
Os dois pontos ("**:**"), colocados após o LABEL, não é obrigatório.

Por exemplo:

```
ROTINA_DELAY_250 :      ; Rotina Delay de 250 ms

SALVA_STATUS :          ; Rotina para salvar o valor do registro STATUS
```

NOTA:

- Jamais utilize a instrução GOTO para acessar uma SUBROTINA, pois, ao término desta, a instrução RETURN ou RETLW não causará o seu retorno automático.

41 – O USO DE UMA SUBROTINA

Numa programação, quando ocorre a necessidade de se repetir diversas vezes uma determinada seqüência de instruções, é recomendável que se crie uma SUBROTINA.
Havendo a necessidade de se utilizar essa SUBROTINA, bastará chama-la utilizando-se a instrução CALL anexada ao nome da subrotina.

Por exemplo : CALL DELAY , CALL FILTRO; etc.

É importante não se esquecer de encerrar as SUBROTINAS com instruções do tipo : RETURN ou RETLW, para ocorrer o retorno automático ao endereço contido no STACK (PC + 1).

PORTANTO, UMA SUBROTINA É UMA ROTINA QUE SE ENCERRA EM INSTRUÇÕES COMO : "RETURN" OU "RETLW", E TEM, COMO PRINCIPAL CARACTERÍSTICA, A PROPRIEDADE DE RETORNO AUTOMÁTICO, DESDE QUE ELA TENHA SIDO CHAMADA POR UM "CALL"

NOTA:

- Há casos em que, antes de chamar uma SUBROTINA, há necessidade de salvar os valores contidos nos registradores W e no STATUS.

Este procedimento já foi visto no capítulo 38.

41.1 - A SUBROTINA DELAY

Numa programação, sempre há a necessidade de se utilizar uma rotina que provoque um retardamento para a execução de determinadas instruções. São conhecidos por SUBROTINAS de atrasos de tempo (delay).

Em nosso exemplo, onde se utiliza um clock de 4 MHz, o ciclo de máquina é de apenas 1 micro segundos. A rotina apresentado à seguir, permite um atraso (delay) de 250 ms. Atente para a explicação abaixo.

```
DELAY_250:    ; (A instrução CALL = 2 Ciclos)  = 2 µs

        MOVLW  251    ; 1µs
        MOVWF  TM1    ; 1µs   (2 + 1 + 1 = 4 µs) ..... Total_1 = 4 ms

LOOP1: MOLW   248     ; 1µs
        MOVWF  TM2    ; 1µs  (1 + 1 = 2 µs ) ..... Total _2 = 2 ms

LOOP2: NOP            ; 1µs
        DECFSZ  TM2   ; 1µs : 247x4 + (1 NOP + 2 DECFSZ  TM2 quando  Z=1) = 991 µs)
                                                          Total_3  = 991 ms
        GOTO LOOP2    ; 2µs :
        DECFSZ  TM1   ; 1µs  : Enquanto TM1>0  (1 + 2 = 3  µs) ..... Total_4 = 3 ms

        GOTO  LOOP1   ; 2 µs : 250 x (Total_2 + Total_3 + Total_4) + Total_2 + Total_3 + 2  quando
                      ; TMS1 = 0
                                          fica ..... Total_5 = 249. 995 ms

        RETURN        ; 2 µs : O Delay = Total_1 + Total_5 + 2 (Return) = ..... 250. 001 ms
```

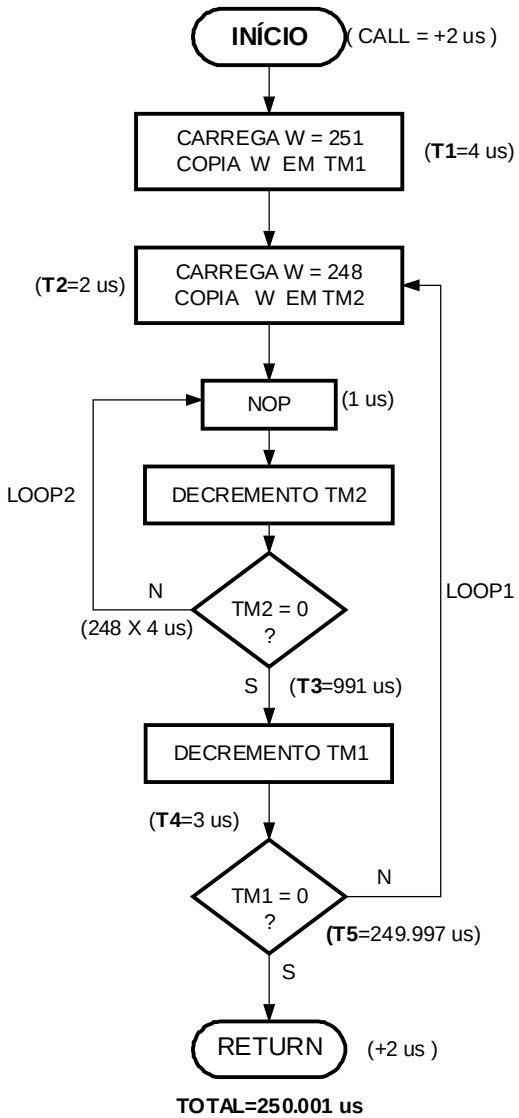
Em resumo :

4 + 250.{[2 + (247 . 4) +1 + 2] + 1 + 2}+ 2 + 991 + 2 = **250.001** ms
a b c d e f g h i j k l

a) - Total_1 = 4 µs (CALL + MOVLW 250 + MOVWF TM1) = (2 + 1 + 1 = 4 µs)
b) - LOOP1 = 250 (251-1 = 250)
c) - Total_2 = 2 µs (MOVLW 248 + MOVWF TM2) = (1 + 1 = 2 µs)

d) - LOOP2 = 248 (248-1 = 247)
e) - NOP + DECFSZ + GOTO LOOP2 = (1 + 1+ 2 = 4 µs)
f) - NOP = 1µs
g) - DECFSZ TM2 = 2 µs, quando TM2 = 0, pula linha
h) - DECFSZ TM1 = 1 µs, para TM1>0
i) - GOTO LOOP1 = 2 µs
j) - MOVLW 248 + MOVWF TM2
k) - [(247.4) + 1 + 2] = 991µs
l) - RETURN = 2 µs

Delay total é de aproximadamente 250 ms.



NOTA :

- A instrução DECFSZ demanda 2 ciclos de máquina, quando ela executa o "pulo" de uma linha de instrução.

41.2 – DETALHAMENTO DA SUBROTINA DELAY

A SUBROTINA delay funciona com o decremento de dois registros, denominados TM1 e TM2 de modo contínuo e em cascata. Ao término desta subrotina, estes dois registros estarão zerados, e em conseqüência, decorrido um tempo aproximado de 250 ms.

a) Inicialmente os registradores TM1 e TM2 são carregados através do W, com os seus respectivos valores: 251 e 248.

b) O registro TM2 é decrementado até que seja zebrado, quando a instrução DECFSZ TM2 causará o "pulo" de uma linha para decrementar o registro TM1.

c) O decremento do registro TM1 é causado pela instrução DECFSZ TM1, e caso este registro seja zerado, ocorrerá o "pulo" de uma linha, e a última instrução, RETURN, encerrará esta SUBROTINA.

NOTAS:

- Os registros TM1 e TM2 deverão ser reservados no espaço RAM utilizando a instrução EQU.

  Exemplo :

```
TM1:    EQU  12     ; Reserva a posição 12 da memória RAM para TM1
TM2:    EQU  13     ; Reserva a posição 13 da memória RAM para TM2
```

- A instrução NOP introduzida dentro do LOOP2, tem o propósito de compensar o valor do produto final, de modo que o TOTAL_3 seja = 991 μs.

- Todas as vezes que a instrução DECFSZ causa um "pulo", demanda 2 ciclos de máquina

- As instruções CALL e RETURN demandam cada um, 2 ciclos de máquinas, e numa SUBROTINA, deverão ser considerados.

- Nem sempre é possível de se obter um tempo exato para uma SUBROTINA delay.

Para obter um delay com um mínimo de atraso, ou para os casos em que há necessidade de se fazer um ajuste fino numa SUBROTINA delay, é interessante utilizar os seguintes recursos :

```
NOP
NOP         ; Delay  com 2 ciclos de máquina
```

ou , então :

```
GOTO $ + 0  ; Delay com 2 ciclos de máquina
```

42 – AS DEFINIÇÕES DE UM “EQU” E “DEFINE”

A diretriz “EQU” (pronuncia-se “equeite”), que já foi visto no capítulo 38.1 item (3), permite associarmos um nome a um número. Isto facilita a programação, pelo fato de podermos referenciar uma variável, apenas pelo seu nome. Não haverá a necessidade de lembrarmos do seu endereço na memória.

É muito útil também ao definirmos uma constante, que serão utilizadas ao longo do programa. Quando ocorrer a necessidade de se alterar o valor desta constante, bastará modificar o número relacionado a este nome.

Por exemplo :

```
TIME:   EQU     07H     ; O nome “TIME” está associado ao número 07H

HORA:   EQU     0CH     ; A variável “HORA” está locada no end. 0CH da RAM
```

Baseada nos dois EQU’s do exemplo acima, entendemos que:

```
        DECF  TIME,f    ; O “TIME” fica (07H – 01H) = 06H

        MOVLW  25       ; Carrega o W com a constante 25
        MOVWF  HORA     ; O endereço 0CH da RAM é carregado com o valor 25
```

Resumindo, o compilador não é capaz de distinguir a diferença de uma constante e de um endereço.
Caberá ao usuário fazer esta distinção durante o desenvolvimento do software.
Portanto, se, acidentalmente, no programa exemplo, executar as seguintes instruções:

```
        INCF  HORA,w    ; O registro W  fica  com (0CH+1) = 0DH
        MOVWF  TIME     ; O valor 0DH não será copiado no endereço 07H, pois a
                        ; RAM começa em 0CH. Vide capítulo 16.
```

42.1 – A DIRETRIZ “DEFINE”

Esta diretriz é muito utilizada para definirmos os pinos de entrada ou de saída de um sistema. (pronuncia-se “defáine”)

Por exemplo :

```
# DEFINE      BOTON       PORTA, 3
```

A partir desta definição “#DEFINE”, o pino RA3 do PORTA poderá ser substituído pelo nome “BOTON”. E todas as vezes que o compilador encontrar a palavra “BOTON”, ele irá substituí-lo pelo endereço referente ao pino 3 do PORTA.
Isto em muito facilitará, quando houver a necessidade de se alterar o endereço do “BOTON”, bastará redefinir o novo #DEFINE, sem a necessidade de rever toda a programação.

A outra aplicação da diretriz #DEFINE permite que um programa funcione, por exemplo, no modo simulado, para tornar compatível com a velocidade do microcomputador que está sendo usado.

Em muitos casos, a simulação utilizada no ambiente do compilador MPLAB, torna-se muito lenta. Nestas situações, para acelerarmos o processo durante a simulação, é possível desconsiderar as rotinas delay's, rodando no modo SIMULADO.

Veja o exemplo abaixo:

```
#DEFINE   SIMULADO     ; (Definindo o modo de funcionamento : SIMULADO)

DELAY:                 ; Esta é uma subrotina delay

#IFDEF    SIMULADO     ; (Se definiu o"SIMULADO", executa a próxima linha de instrução)

          RETURN       ; ( Neste exemplo, o"SIMULADO" não executa a subrotina delay)

#ELSE                  ; (Se não definiu o "SIMULADO", então executa a próxima linha)

          MOVWF  TM1   ;
LOOP3:    MOVLW  248   ;
          MOVWF  TM2   ;

          ( CONTINUA.... )

          RETURN

#ENDIF                 ; (Aqui termina o bloco, sujeito à definição do "SIMULADO")
```

Para **cancelar** a diretriz #DEFINE SIMULADO, basta colocar o "**;**" em sua frente, assim :

```
;#DEFINE SIMULADO     ; (0 " ;" está cancelando a definição!)
```

Neste caso, as instruções que ficam imediatamente abaixo da diretriz #ELSE, serão executados.

ASSIM, PARA CADA MODO DE FUNCIONAMENTO, É IMPORTANTE COMPILAR NOVAMENTE O ARQUIVO PROJETO.

Resumindo :

#DEFINE : Define o modo de simulação : com ou sem a subrotina delay.

#IFDEF : ("Se definiu") Executa a linha abaixo, se #DEFINE não estiver "**;**".

#ELSE : ("Então") Bloco abaixo deverá ser executado, se #DEFINE estiver "**;**".

#ENDIF : Indica o fim do bloco de execução condicionado a uma definição.

43 – AS DIRETRIZES “ORG” ; “END” E O INCLUDE

Estas diretrizes não pertencem ao grupo das Instruções Set, porém, são muito úteis na montagem do programa, pois são totalmente compatíveis com o compilador do MPLAB.

43.1 - O ORG

A diretriz “ORG” permite ao usuário escolher o endereço inicial para ser gravado na memória de programa.
Por exemplo: deseja-se gravar o label INÍCIO a partir do endereço 0200H. Basta escrever assim:

```
ORG  0200H        ; 200H é o endereço inicial deste programa exemplo.

INÍCIO :

    MOVLW  0FFH   ; Carrega W com o valor 0FFH
    MOVWF  PORTB  ; Copia o valor de W em PORTB
    ( continua ..... )
```

NOTA : Recomenda-se utilizar apenas a representação em Hexadecimal, para evitar incompatibilidade de interpretação do endereço do “ORG” pelo compilador.

43.2 – O END

O compilador, quando detecta a presença da diretriz “END”, encerra a sua tarefa de compilação.
Portanto, para terminar um programa, a presença da diretriz “ORG” é indispensável para o compilador.

43.3 – O INCLUDE <nomearq>

A diretriz “INCLUDE” serve para informar ao compilador para anexar à atual fonte, durante uma compilação, o arquivo <nomearq>.
Portanto, é usado no início do programa, para incluir definições específicas contidas em arquivos auxiliares anexos. Esses arquivos que têm as mesmas formatações do código fonte, possuem somente definições de variáveis e de constantes.
Os símbolos “<” e “>” podem ser substituídos por aspas (“ ”).
Por exemplo:

```
INCLUDE “DELTA.ASM”  ; Anexa a esta fonte, o arquivo “DELTA.ASM”
```

ou, então :

INCLUDE "C:\ FITO\ PESQUISA.ASM"

Neste exemplo, devido ao path "C:\", o arquivo "PESQUISA.ASM" será buscado na raíz "C", mais precisamente, no diretório "FITO".

Caso o path não esteja especificado, a ordem de busca será : o diretório atual, o diretório dos arquivos de código fonte e finalmente, o diretório do MPASM.

44 – AS OUTRAS DIRETRIZES ÚTEIS

Dentre as diversas diretrizes da linguagem MPASM, algumas já vista destacaremos as mais utilizadas.

44.1 – CBLOCK <expr> E ENDC

Define uma série de constantes. O primeiro nome é associado ao valor da <expr>. O segundo é associado ao valor seguinte e assim sucessivamente. Caso não esteja determinado o valor do incremento, a próxima constante receberá o valor imediato.
Por exemplo:

```
CBLOCK    0CH          ; Início da locação de endereços na memória RAM

          TM1          ; Variável TM1 = endereço 0CH na RAM
          TM2          ; Variável TM2 = endereço 0DH na RAM
          TM3          ; Variável TM2 = endereço 0EH na RAM

ENDC                   ; Fim das definições das variáveis na RAM
```

44.2 – DATA <expr>

Preenche a memória RAM com números ou texto, começando na posição atual e cobrindo quantas forem necessárias, dependendo do tamanho do texto ou dos números.

Os textos em ASCII são colocados dois caracteres em cada posição de memória, e caracteres unitários ocupam um em cada posição. Caso o texto possua um número ímpar de caracteres, um zero é complementado no último byte.

Por exemplo:

```
DATA      0,1,2,3,4,5          ; Números
DATA      "FITO - OSASCO"      ; Texto
DATA      'F'                  ; Código ASCII
```

44.3 – DB <expr>, ...<expr>

Preenche a memória RAM com o valor determinado por <expr> , iniciando na atual posição. guardando byte a byte, quantos forem necessários. Se a quantidade de <expr> for impar, um zero é complementado no último byte.

Por exemplo:

DB 02, 20H, 'F', Paulo

44.4 – DE <expr> , ... <expr> ou DE "texto"

Preenche a memória com valor determinado por <expr> ou "texto", iniciando na atual posição, byte a byte, quantos forem necessários. No caso do <texto>, cada caracter será gravado em um byte.

44.5 – DT <expr>, ...<expr> ou "texto"

Esta diretriz tem a particularidade de preencher a memória com uma série de instruções RETLW para cada <expr>,ou caracter do texto, começando na atual posição, até quantos forem necessários.

Por exemplo:

DT "FITO"

Equivale a escrever :

```
RETLW   'F'
RETLW   'I'
RETLW   'T'
RETLW   'O'
```

44.6 – DW <expr> ou <expr>, ..."texto"

Preenche a memória com valor determinado por <expr> ou "texto", iniciando na atual posição, byte a byte, quantos forem necessários. No caso do <texto>, cada caracter será gravado em um byte. Se o texto possuir um número ímpar de caracteres, um zero é escrito no último byte.

Exemplo:

DW "FITO",30

44.7 – MACRO E ENDM

A diretriz “MACRO” define o início de um bloco de uma macro, e o “ENDM” finaliza o bloco de uma macro.
Uma macro é utilizada para economizar a digitação em uma seqüência de instruções repetitivas.

Por exemplo:

Para se acessar o BANCO1, é necessário digitar : BSF STATUS,RP0.
E, para voltar ao BANCO0, é necessário digitar : BCF STATUS,RP0.

Definindo macros, fica:

```
BANK_1:   MACRO            ; Início do bloco de uma macro
          BSF  STATUS,RP0  ; Seta o bit RP0 do registro STATUS
          ENDM             ; Fim do bloco de uma macro

BANK_0:   MACRO            ; Início do bloco de uma macro
          BCF  STATUS,RP0  ; Zera o bit do registro STATUS
          ENDM             ; Fim do bloco de uma macro
```

À partir das definições das macros BANK_0 e BANK_1, fica:

Para acessar o BANCO1, é necessário digitar apenas : BANK_1
E, para voltar ao BANCO0, é necessário digitar apenas : BANK_0

NOTA : O compilador ao encontrar uma instrução definida em macro, substitui-a pela seqüência de instruções normalizadas.

45 – OS MACROS DEFINIDOS (INSTRUÇÕES ESPECIAIS)

Existem instruções especiais que, embora não façam partes das instruções do PIC, por já estarem definidas em macro pelo próprio compilador, são muitos utilizados.

Recomenda-se conhecê-las, para possibilitar o entendimento e, se necessário, de fazer uma possível alteração de outros programas escrito por terceiros.

TABELA DE INSTRUÇÕES ESPECIAIS

Instruções	***Descrições***	***Instruções Equiv.***	***Status***
BNZ k	Pula (blanch) para posição definida pelo k, caso tenha ocorrido um zero.	BTFSS STATUS, Z GOTO k	---
ADDDCF f,d	Se houver digit carry, incrementa o f, guarda em d.	BTFSC STATUS, DC INCF f,d	Z

Continuação da tabela de instruções especiais			
B k	Pula (blanch) para posição definida pelo k	GOTO k	---
BC k	Pula (blanch) para posição definida pelo k, caso tenha ocorrido um carry.	BTFSC STATUS, C GOTO k	---
BDC k	Pula (blanch) para posição definida pelo k, caso tenha ocorrido um digit carry.	BTFSC STATUS, DC	---
ADDCF f,d	Se houver carry, incrementa o f, e guarda em d.	BTFSC STATUS, C INCF f,d	Z
BNDC k	Pula (blanch) para posição definida pelo k, caso não tenha ocorrido um digit carry.	BTFSS STATUS, DC GOTO k	---
NEGF f,d	Ajusta o valor negativo do resultado de uma conta.	COMF f,F INCF f,d	Z
BZ k	Pula (blanch) para posição definida pelo k, caso tenha ocorrido um zero.	BTFSC STATUS, Z GOTO k	---
BNC k	Pula (blanch) para posição definida pelo k, caso não tenha ocorrido um carry.	BTFSS STATUS, C GOTO k	---
CLRC	Limpa o bit carry.	BCF STATUS, C	C
CLRDC	Limpa o bit digit carry.	BCF STATUS, DC	DC
CLRZ	Limpa o bit de zero.	BCF STATUS, Z	Z
LCALL k	Chamada de subrotina distante (k). Acerta o PCLATH automaticamente.	BCF/BSF PCLATH, 3 BCF/BSF PCLATH, 4 CALL k	---
LGOTO k	Desvio de programa distante (k). Acerta o PCLATH automaticamente.	BCF/BSF PCLATH,3 BCF/BSF PCLATH,2 GOTO k	---
MOVFW f	Move o valor do registrador f para W.	MOVF f,d	Z
SETC	Seta o bit de carry.	BSF STATUS, C	C
SETDC	Seta o bit de digit carry.	BSF STATUS, DC	DC
SETZ	Seta o bit de zero.	BSF STATUS, Z	Z
SKPC	Pula a próxima linha, se houver um carry.	BTFSS STATUS, C	---
SKPDC	Pula a próxima linha, se houver um digit carry.	BTFSS STATUS, DC	---
SKPNC	Pula a proxima linha, se não houver um carry.	BTFSC STATUS, C	---
SKPNDC	Pula a próxima linha, se não houver um digit carry	BTFSC STATUS, DC	---
SKPNZ	Pula a próxima linha, se não houver um zero.	BTFSC STATUS, Z	---
SKPZ	Pula a próxima linha, se houver um zero.	BTFSS STATUS, Z	---
SUBCF f,d	Se houve carry, decrementa f e o resultado fica em d.	BTFSC STATUS, C DECF f,d	Z
SUBDCF f,d	Se houve digit carry, decrementa f e o resultado fica em d.	BTFSC STATUS, DC DECF f,d	Z
TSTF f	Testa o registrador f para verificar se é zero.	MOVF f,F	Z

46 – AS SIMULAÇÕES NO MPLAB

Para facilitar o entendimento das instruções set do PIC deste capítulo em diante, serão apresentados como exemplos, diversas situações de uma programação.
Convém saber que o comportamento da velocidade destas simulações, dependem da CPU do microcomputador que estiver em uso.

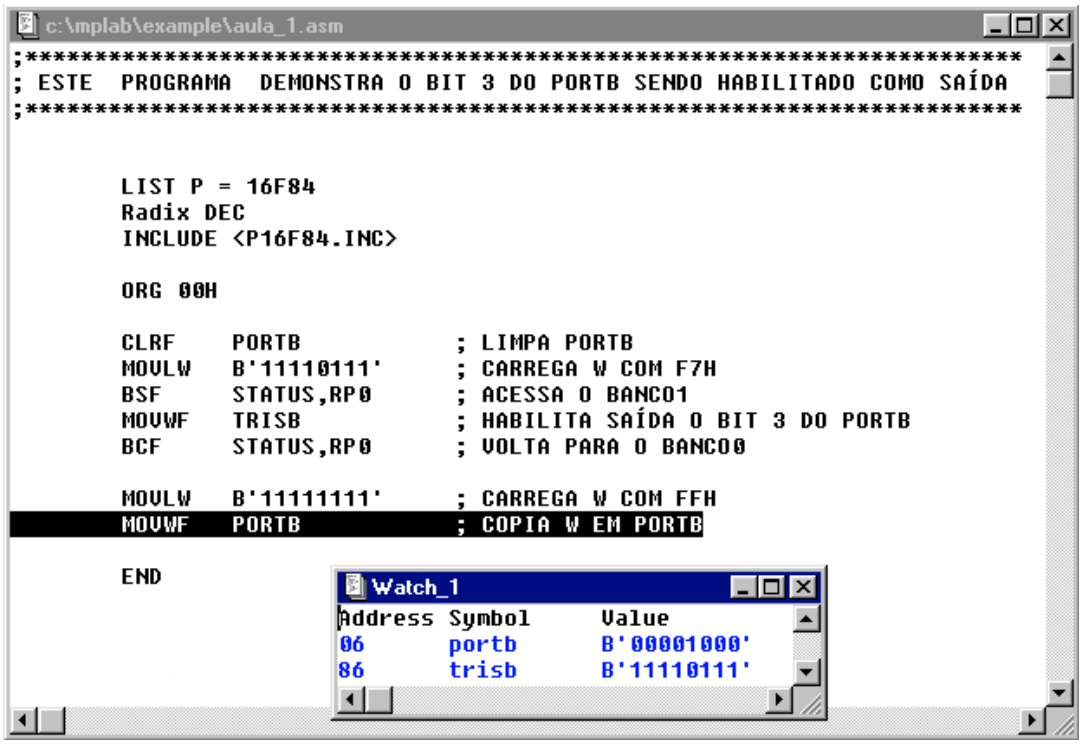
46.1 – HABILITANDO UM BIT DO PORTB COMO SAÍDA

O primeiro exemplo tem como o objetivo de tornar o bit 3 do PORTB como saída do sistema.
Assim, ao tentar escrever um valor B‘11111111’ (FFH) neste PORT, somente o bit que está habilitado como saída será setado.

Digite no ambiente do MPLAB o programa abaixo, compile-o, abra a janela “Watch Windows” com TRISB e o PORTB em binário, e através da tecla F7 comprove.

```
ORG 00H

CLRF        PORTB           ; LIMPA PORTB
MOVLW       B'11110111'     ; CARREGA W COM F7H
BSF         STATUS,RP0      ; ACESSA O BANCO1
MOVWF       TRISB           ; HABILITA SAÍDA O BIT 3 DO PORTB
BCF         STATUS,RP0      ; VOLTA PARA O BANCO0

MOVLW       B'11111111'     ; CARREGA W COM FFH
MOVWF       PORTB           ; COPIA W EM PORTB

END
```

A apresentação no ambiente do MPLAB, fica assim:



NOTA : Após rodar o programa acima, somente o bit 3 do PORTB será setado, e o TRISB apresenta o bit 3 zerado, comprovando que o mesmo está configurado como saída.

46.2 – RODANDO UM BIT À ESQUERDA

O programa abaixo, configura o PORTB como saída, copia o valor B‘00000001’ nela, e começa à rodar à esquerda, bit por bit.
O intervalo de mudança de um bit para outro, dependerá da velocidade da CPU do microcomputador utilizado. Portanto, ainda neste exemplo, por não constar a rotina delay, a velocidade da simulação dependerá do desempenho do microcomputador.

Digite o programa abaixo, compile-o e simule visualizando no Watch Windows o PORTB.

```
;*****************************************************************************
;* O PROGRAMA DEMONSTRA  A SAIDA PORTB RODANDO UM BIT À ESQUERDA ATÉ ATINGIR O CARRY  *
;* QUANDO OCORRE O CARRY, REINICA TUDO NOVAMENTE                               *
;*****************************************************************************
;                                                                  29/04/2005
        LIST P=16F84
        Radix DEC
        INCLUDE <P16F84.INC>

        ORG 000H        ; INÍCIO DA GRAVAÇÃO NA EEPROM


        CLRF    PORTB           ; ZERA A SAÍDA PORTB
        BSF     STATUS,RP0      ; BANCO 1
        CLRF    TRISB           ; HABILITA SAIDA PORTB
        BCF     STATUS,RP0      ; BANCO 0

loop2:  BSF     PORTB,0         ; SETA O PRIMEIRO BIT PORTB
        BCF     STATUS,C        ; ZERA O FLAG CARRY (NO REGISTRO DO STATUS)

LOOP1:  RLF     PORTB           ; RODA UMA VEZ O PORTB À ESQUERDA
        BTFSS   STATUS,C        ; SE NÃO HÁ CARRY, PULA UMA LINHA
        GOTO    LOOP1           ; ENQUANTO NÃO HÁ CARRY, CONTINUA EM LOOP1
        GOTO    LOOP2           ; SE CARRY = 1, REINICIA EM LOOP2

        END
```

Watch_1

Address	Symbol	Value
06	portb	B'00000001'
86	trisb	B'00000000'

46.3 – RODANDO UM BIT À ESQUERDA E À DIREITA

Neste exemplo, o PORTB que está configurado como saída e apresenta um bit que roda da direita à esquerda, ao chegar ao último bit, retorna rodando à direita.

Este processo é contínuo e a velocidade do deslocamento dependerá da CPU do microcontrolador em uso.

```
;*********************************************************************
;* O PROGRAMA DEMONSTRA A SAIDA PORTB RODANDO UM BIT  À   ESQUERDA *
;* AO ATINGIR O CARRY, REINICIA O RETORNO À DIREITA, CONTINUAMENTE *
;*********************************************************************

        LIST P = 16F84
        RADIX DEC
        INCLUDE <P16F84.INC>

        ORG 0000H

        CLRF    PORTB           ;ZERA A SAÍDA PORTB
        BSF     STATUS,RP0      ;BANCO 1
        CLRF    TRISB           ;HABILITA SAIDA PORTB
        BCF     STATUS,RP0      ;BANCO 0

        BSF     PORTB,0         ;SETA O PRIMEIRO BIT PORTB
        BCF     STATUS,C        ;ZERA O FLAG CARRY

LOOP:   RLF     PORTB           ;RODA O PORTB A ESQUERDA
        BTFSS   STATUS,C        ;SE CARRY=1, SALTA P/ LOOP1
        GOTO    LOOP            ;CONTINUA A RODAR À ESQUERDA

LOOP1:  RRF     PORTB           ;RODA PORTB P/ A DIREITA
        BTFSS   STATUS,C        ;SE CARRY=1, SALTA P/ LOOP
        GOTO    LOOP1           ;CONTINUA A RODAR À DIREITA
        GOTO    LOOP            ;REINICIA TUDO EM LOOP

        END
```

Watch_1

Address	Symbol	Value
06	portb	B'00010000'

46.4 – SIMULANDO UM BARGRAPH

O último programa exemplo desta série consiste de um programa que faz do PORTB uma saída, onde os bits começam a setar, um a um da direita para à esquerda, e quando todos os bits estiverem setados, começam a zerar no sentido contrário.

```
;*********************************************************************
;*  O PROGRAMA DEMONSTRA A SAÍDA PORTB RODANDO UM À BIT ESQUERDA   *
;* ACOMPANHADO DOS DEMAIS, PROGRESSIVAMENTE,  REPETINDO  À  DIREITA *
;*********************************************************************

        LIST P = 16F84
        RADIX DEC
        INCLUDE <P16F84.INC>

        ORG   000H              ; INÍCIO DO ENDEREÇO DA GRAVAÇÃO

        CLRF    PORTB           ; ZERA A SAÍDA PORTB
        BSF     STATUS,RP0      ; BANCO 1 (DEVIDO AO TRISB)
        CLRF    TRISB           ; HABILITA SAIDA PORTB
        BCF     STATUS,RP0      ; BANCO 0 (PADRÃO)
```

```
        BSF     PORTB,0     ; SETA O PRIMEIRO BIT DO PORTB
        BCF     STATUS,C    ; ZERA O FLAG CARRY

LOOP2:  RLF     PORTB,0     ; RODA O VALOR PORTB A ESQUERDA E SALVA EM W
        IORWF   PORTB,0     ; CARREGA NO PORTB W  (W OR PORTB)
        MOVWF   PORTB       ; CARREGA O VALOR W EM PORTB
        BTFSS   STATUS,C    ; SE CARRY = 1, SALTA EM LOOP1
        GOTO    LOOP2       ; REPETE O LOOP2

LOOP1:  BCF     STATUS,C    ; LIMPA O CARRY
        RRF     PORTB,0     ; RODA O VALOR DO PORTB À DIREITA, E SALVA EM W
        ANDWF   PORTB,0     ; FAZ AND ENTRE W E O VALOR DO PORTB
        MOVWF   PORTB       ; CARREGA O VALOR W EM PORTB
        BTFSS   STATUS,Z    ; SE Z=1, SALTA UMA LINHA
        GOTO    LOOP1       ; CONTINUA A RODAR À DIREITA O VALOR DO PORTB
        GOTO    LOOP2       ; REINICIA TUDO EM LOOP2

        END
```

46.5 – SIMULANDO UMA SUBROTINA DELAY

Esta simulação, baseada em uma subrotina delay, que já foi vista no capítulo 44.1, comprova o retardo causado pelos loops envolvidos. Através de paradas geradas pelos "Break points" é possível de se cronometrar, de forma precisa, todos os tempos envolvidos.

O cronômetro do simulador do MPLAB, chamado de "Stopwatch", permite medir os intervalos reais de tempos.

Digite o programa exposto, compile-o e, na simulação, siga o roteiro descrito.

```
;*******************************************************************************
;* ESTE  PROGRAMA TEM A FINALIDADE DE SIMULAR UMA ROTINA DELAY DE 250 ms *
;*******************************************************************************


        LIST P=16F84
        Radix DEC
        INCLUDE <P16F84.INC>

        ORG     000H    ; INÍCIO DA GRAVAÇÃO NA EEPROM

TM1:    EQU     0CH     ; RESERVA A POSIÇÃO 0CH NA RAM
TM2:    EQU     0DH     ; RESERVA A POSIÇÃO 0DH NA RAM


DELAY:                  ; NOME DO LABEL DESTA SUBROTINA

        MOVLW   251     ; CARREGA W COM O VALOR 251
        MOVWF   TM1     ; COPIA O W NA VARIÁVEL TM1
```

```
        LOOP1: MOVLW    248     ; CARREGA W COM O VALOR 248
               MOVWF    TM2     ; COPIA O W NA VARIÁVEL TM2

LOOP2: NOP                      ; DEMORA UM CICLO DE MÁQUINA
               DECFSZ   TM2     ; DECREMENTA TM2, SE Z=1, PULA UMA LINHA
               GOTO  LOOP2      ; CONTINUA A DECREMENTAR TM2 EM LOOP2

               DECFSZ   TM1     ; DECREMENTA TM1, SE Z=1, PULA UMA LINHA
               GOTO  LOOP1      ; RETORNA PARA LOOP1

               RETURN           ; VOLTA PARA A ROTINA PRINCIPAL

               END
```

ROTEIRO :

1- Abra as janelas do “Stopwacth” e do Watch Window contendo os registros TM1 e TM2, com representações em decimais.

2- Marque com um “Break Point” a última instrução : RETURN.

3- Simule no modo passo-a-passo, através da tecla F7. (Use F6 para resetar).

4- Repita a simulação através das teclas < CONTROL> F9.

5- Observe que o registro TM2 estará decrementando de modo acelerado, enquanto o TM1, somente a cada ciclo completo do TM2.

6- Acione a tecla F5 (stop) para interromper, e depois o F6 para resetar o sistema.

7- Pressione apenas a tecla F9, para uma simulação rápida.

8- Aguarde o cronômetro encerrar a contagem. Estará acusando : 250,00 ms (249.997 ciclos), mais precisamente, 249,997 ms.

NOTA:

- EM UMA ROTINA DELAY, É IMPORTANTE COMPUTAR OS ATRASOS CAUSADOS PELAS INSTRUÇÕES : CALL E RETURN QUE DEMANDAM UM TOTAL DE 4 μs. PORTANTO, O DELAY TOTAL = 250,001 ms.

O ajuste dos tempos em uma subrotina delay requer um conhecimento detalhado do comportamento dos loops envolvidos.

Para que não fique nenhuma dúvida sobre a rotina delay do exemplo anterior, iremos analisá-lo em partes.

```
                MOVLW   251          X
                MOVWF   TM1
LOOP1:          MOVLW   248
                MOVWF   TM2          Y
LOOP2:          NOP
                DECFSZ  TM2          LOOP2      LOOP1
                GOTO    LOOP2

                DECFSZ  TM1
                GOTO    LOOP1

                RETURN
```

NOP — DEFSZ TM2 (Z=1)

LOOP2: (247 . 4) + 1 + 2 = **991** ms

(NOP + DEFSZ TM2 (Z=0) + GOTO LOOP2)

DECFSZ TM1 (Z=0) — **Y**

LOOP1: 250 . (2 + LOOP2 + 1 + 2) + 2 + LOOP2 + 2 = **249.995** ms

Y — GOTO LOOP1 — ECFSZ TM1 (Z=1)

No simulado, o cronômetro registra : DELAY = X + LOOP2 = **249. 997** ms

X

CALL DELAY — **RETURN**

DELAY TOTAL = 2 + 1 + 1 + LOOP2 + 2 = **250,001 ms**

MOVLW 251

MOVWF TM1

46.6 – UTILIZANDO UMA SUBROTINA DELAY

Neste exemplo, estão anexados os exemplos dos capítulos 46.1 com o 46.5, ou seja, a subrotina delay de 250 ms gera um atraso na varredura dos bits no PORTB, que permite visualizá-lo.

Assim sendo, o PIC ao ser gravado com este programa, e se inserido na placa de ensaio proposto nesta publicação, alternará os led's em um intervalo de 250 ms.

Para dobrar o tempo desse intervalo, experimentalmente, insira duas instruções seguidas de "CALL DELAY".

```
;*************************************************************************************************
;* O PROGRAMA DEMONSTRA A SAÍDA PORTB RODANDO UM BIT  À  ESQUERDA, AO ATINGIR  *
;* O CARRY, REINICIA O RETORNO À DIREITA,  CONTINUAMENTE. ESTÁ  INCORPORADO  UMA  *
;* UMA ROTINA DELAY DE 250 ms.                  APLICÁVEL NO CIRCUITO EXPERIMENTAL  *
;*************************************************************************************************

        LIST P=16F84
        Radix DEC
        INCLUDE <P16F84.INC>

        ORG     000H            ; INÍCIO DA GRAVAÇÃO NA EEPROM

;------------------------------ definicões de espaços na RAM -----------------------------------

TM1:    EQU     0CH             ; RESERVA A POSIÇÃO 0CH NA RAM
TM2:    EQU     0DH             ; RESERVA A POSIÇÃO 0DH NA RAM

; ---------------------------------- início do programa principal ----------------------------------

        CLRF    PORTB           ; ZERA A SAÍDA PORTB
        BSF     STATUS,RP0      ; BANCO 1
        CLRF    TRISB           ; HABILITA SAIDA PORTB
        BCF     STATUS,RP0      ; BANCO 0

        BSF     PORTB,0         ; SETA O PRIMEIRO BIT PORTB
        BCF     STATUS,C        ; ZERA O FLAG CARRY

LOOP3:  CALL    DELAY           ; DELAY  250 ms
        RLF     PORTB           ; RODA O PORTB A ESQUERDA
        BTFSS   STATUS,C        ; SE CARRY=1, SALTA P/ LOOP4
        GOTO    LOOP3           ; CONTINUA A RODAR À ESQUERDA

LOOP4:  CALL    DELAY           ; DELAY 250 ms
        RRF     PORTB           ; RODA PORTB P/ A DIREITA
        BTFSS   STATUS,C        ; SE CARRY=1, SALTA P/ LOOP3
        GOTO    LOOP4           ; CONTINUA A RODAR À DIREITA
        GOTO    LOOP3           ; REINICIA TUDO EM LOOP3

;----------------------------------- subrotina delay  de 250 ms  ---------------------------------

DELAY:                          ; NOME DO LABEL DESTA SUBROTINA DELAY

        MOVLW   251             ; CARREGA W COM O VALOR 251
        MOVWF   TM1             ; COPIA O W NA VARIÁVEL TM1

LOOP1:  MOVLW   248             ; CARREGA W COM O VALOR 248
        MOVWF   TM2             ; COPIA O W NA VARIÁVEL TM2

LOOP2:  NOP                     ; PERDE UM CICLO DE MÁQUINA
        DECFSZ  TM2             ; DECREMENTA TM2, SE Z=1, PULA UMA LINHA
        GOTO    LOOP2           ; CONTINUA A DECREMENTAR TM2 EM LOOP2

        DECFSZ  TM1             ; DECREMENTA TM1, SE Z=1, PULA UMA LINHA
        GOTO    LOOP1           ; RETORNA PARA LOOP1

        RETURN                  ; VOLTA PARA A ROTINA PRINCIPAL

        END
```

46.7 – EXEMPLO DE UMA INTERRUPÇÃO POR TMR0

No exemplo apresentado, a interrupção causada pelo estouro (overflow) do TIMER 0, leva à uma rotina que incrementa o PORTB. Assim, o valor do PORTB, que antes da interrupção era D'00', vai para D'01' no primeiro estouro do tmr0.
O intervalo de interrupção depende da CPU do microcomputador usado para realizar esta simulação.

Insira na janela do Watch Window, os registros PORTB em DECIMAL, e INTCON em binário. E abra também o Stopwatch (cronômetro).

```
;*****************************************************************************************
; ESTE PROGRAMA UTILIZA A INTERRUPÇÃO POR TIMER ZERO. O PORTB É INCREMENTADO
; QUANDO OCORRE UM ESTOURO (TRANSBORDO à cada 25 us) DO TMR0
;  Para verificar, basta observar o PORTB no modo Decimal se incrementando à cada interrupção do TMR0
;*****************************************************************************************

        LIST P = 16F84
        RADIX DEC
        INCLUDE <P16F84.INC>

W2:     EQU   0CH                   ; CÓPIA DO W NO INSTANTE DA INTERRUPÇÃO
STATUS2:EQU   0DH                   ; CÓPIA DO STATUS NO INSTANTE DA INTERRUPÇÃO

        ORG   000H                  ; DEFINE O ENDEREÇO INICIAL DA EEPROM

        GOTO  INÍCIO                ; PONTO DE ENTRADA DESTA PROGRAMAÇÃO

        ORG   004H                  ; ENDEREÇO VETORIZADO DE INTERRUPÇÕES

        MOVWF       W2              ; SALVA O CONTEÚDO DO W EM W2
        MOVF        STATUS,W        ; COPIA O CONTEÚDO DO STATUS EM W
        MOVWF       STATUS2         ; SALVA O CONTEÚDO DO STATUS EM STATUS2
        GOTO        T0_INT          ; OCORREU INTERRUPÇÃO, SALTA PARA T0_INT

FIM_INT:
        BCF         INTCON,T0IF     ; LIMPA O FLAG T0IF
        MOVF        W2,W            ; RESTAURA O CONTEÚDO DO W ANTERIOR
        MOVF        STATUS2,W       ; COPIA O CONTEÚDO DO STATUS2 EM W
        MOVWF       STATUS          ; RESTAURA O STATUS ANTERIOR À INTERRUPÇÃO
        MOVLW       250             ; CARREGA W COM A CONSTANTE 250
        MOVWF       TMR0            ; O TMR0 INICIA EM 250 (256-250 = 006)
        RETFIE                      ; RETORNA PARA O PROGRAMA PRINCIPAL (LOOP)

INÍCIO:
        MOVLW       B'10000000'     ; CARREGA W COM 80H (GIE=1 e T0IE0= 0)
        MOVWF       INTCON          ; HABILITA APENAS O GIE (T0IE=0)
        CLRF        PORTB           ; LIMPA O PORTB (SEGURANÇA)
        BSF         STATUS,RP0      ; ACESSA O BANCO0 (TRISB e OPPTION_REG)
        CLRF        TRISB           ; HABILITA TODOS OS BITS DO PORTB COM SAÍDA
        MOVLW       B'11010001'     ; CARREGA W COM 0D1H
        MOVWF       OPTION_REG      ; PRESCAL.: TMR0 (PSA=0), CLK INT (TOCS=0) e 1:4 (PS0=1)
        BCF         STATUS,RP0      ; RETORNA PARA O BANCO0
        BCF         INTCON,T0IF     ; LIMPA O FLAG DE INTERRUPÇÃO POR TMR0 (T0IF=0)
        MOVLW       250             ; CARREGA W COM A CONSTANTE 250
        MOVWF       TMR0            ; O TMR0 INICIA EM 250 (256-250 = 006)
        BSF         INTCON,T0IE     ; HABILITA A INTERRUPÇÃO TMR0 NESTE INSTANTE

LOOP:   NOP                         ; FICA EM LOOP AGUARDANDO INTERRUPÇÃO POR TMR0
        GOTO        LOOP
```

```
T0_INT: INCF    PORTB     ; INCREMENTA O PORTB
        GOTO    FIM_INT   ; FIM DA INTERRUPÇÃO

        END               ; FIM DESTA PROGRAMAÇÃO
```

NOTAS:

1- No “New Watch”, insira :

-“**portb**” e o “**tmr0**” em DECIMAL, o “**intcon**” e o “**option_reg**” em BINÁRIO

2- Observe que à cada 25us aproximados, ocorre uma interrupção por TMR0, que é o resultado : (4 x 1us) x 6 = 24us + 1us em variações = 25us.
Sendo o 4 do Prescaler (1:4) e 6 do (256-250 = 6)

3- Com o “Stopwath” (cronômetro), verifique que o tempo de transbordo é de (39-14)us = 25us.

- O tempo até : “LOOP: NOP” = 14us
- O tempo até : “MOVWF W2” = 39us

Utilize os recursos do “Break point(s)” clicando o mouse no botão direito.

46.8 – EXEMPLO DE UMA MONTAGEM DE UMA TABELA

Neste exemplo de rotina, o registro W retorna com o caracter em código ASCII da tabela, e apresenta um a um no PORTB. O registro “CONTAD” indica a quantidade de caracter necessária para o W transportar até o PORTB. Quando o “CONTAD” for zerado, o programa entenderá que chegou ao fim desta execução.

O outro registro, o “OFFSET”, informa ao W a posição exata do caracter para ser transportado, iniciando do 00 até o último, que é 18. Este número é adicionado ao PCL (os dois dígitos baixos do PC) numa operação ADDWF.

O valor do PCL é exatamente a linha onde se encontra o primeiro caracter, no qual é acrescentado o valor do “OFFSET”.

NOTA: ESTA TABELA NUNCA PODERÁ CONTER UMA TRANSIÇÃO DE ENDEREÇO “00H”. POR EXEMPLO : DE FFH PARA 100H.
POIS, O “ADDWF” NÃO INCREMENTA O “PCLATH” (PARTE ALTA DO PC)

```
;=====================================================================
; Objetivo : Gravar em um espaço da MEMÓRIA uma frase em código  ASCII,
; que será colocada na saída do PORTB, caracter por caracter.
;=====================================================================
        list p = 16F84
        Radix dec
        Include <P16F84A.INC>
;------------------------------ define fusíveis-------------------------
        __config  _xt_osc & _cp_off & _wdt_off & _pwrte_on
;------------------------- espaço na RAM ------------------------------
OFFSET:         EQU   012       ; ENDEREÇO DE REFERÊNCIA DOS CARACTERES
CONTAD:         EQU   013       ; REGISTRO VARIÁVEL DO CONTADOR
;------------------------- endereço de RESET --------------------------
        ORG     00H
        GOTO    INICIO          ; VAI PARA O INICIO
;------------------------- endereço das INTERRUPÇÕES -------------------
        ORG     04              ; NÃO FOI IMPLEMENTADO NENHUMA INTERRUPÇÃO
        GOTO    INICIO
;--------------------- condições iniciais do PIC ----------------------

INICIO: CLRF    PORTA           ; LIMPA PORTA
        CLRF    PORTB           ; LIMPA PORTB
        BSF     STATUS,RP0      ; ACESSA O BANCO1
        CLRF    TRISB           ; HABILITA PORTB COMO SAÍDA
        BCF     STATUS,RP0      ; VOLTA PARA O BANCO0

;------------------------- PROGRAMA PRINCIPAL--------------------------

        MOVLW   19              ; CARREGA CONTADOR COM VALOR 19
        MOVWF   CONTAD          ; INICIA COM CONTADOR = 19 (QD.CARACT.)
        CLRF    OFFSET          ; ZERA OFFSET (REF. DE POSIÇÃO CARACT.)
REPETE:
        CALL    TABELA          ; VAI ATÉ TABELA, VOLTA C/ CARACT. EM W
        MOVWF   PORTB           ; COLOCA O VALOR DE W EM PORTB
        INCF    OFFSET          ; INCREMENTA OFFSET (PRÓXIMO CARACTER)
        DECFSZ  CONTAD          ; DECREMENTA CONTADOR, SE Z=0, SALTA
        GOTO    REPETE          ; SE NÃO, CONTINUA
        GOTO    INICIO          ; VOLTA PARA O INICIO

;---------------------------- tabela de caracteres ----------------------
TABELA:
        MOVF    OFFSET,W        ; CARREGA W COM OFFSET
        ADDWF   PCL,F           ; RETORNA COM PRÓXIMO CARAC. EM W

        RETLW   'F'
        RETLW   'I'
        RETLW   'T'
        RETLW   'O'
        RETLW   ' '
        RETLW   'n'
        RETLW   'a'
        RETLW   ' '
        RETLW   'e'
        RETLW   'r'
        RETLW   'a'
        RETLW   ' '
        RETLW   'D'
        RETLW   'I'
        RETLW   'G'
        RETLW   'I'
        RETLW   'T'
        RETLW   'A'
        RETLW   'L'

        END             ; Fim desta programação
```

A tabela anterior também poderia ser escrita assim :

```
DT      "FITO na era DIGITAL"
```

Digite o programa anterior, compile-o, abra as janelas Watch Windows, contendo: PORTB em Código ASCII, OFFSET em decimal, CONTAD em decimal, TRISB em binário e W em hexadecimal.

Com o auxílio da tecla F7 na simulação passo-a-passo, observe que no PORTB será transportado os caracteres da tabela .

Experimente rescrever a tabela na outra forma apresentada. Note que nada mudará para o efeito de compilação, porém, dificultará o entendimento do mecanismo na simulação.

Substitua o "ORG 04H" pelo "ORG 0E0H", compile-o novamente e repita a simulação. Note que, ao chegar na posição do caracter "I", da palavra "DIG**I**TAL" , o programa se direciona repentinamente para a posição INICIO (00H).

Isto ocorre, pois, o PCL somado ao último OFFSET, resultou num endereço 100H. Como já havia dito, o **PCLATH** (parte alta do PC) **não é atualizado**, logo o PIC entenderá o endereço 100H como endereço 00H (início).

No compilador MPLAB, na barra superior, no "Window", abra o "Absolute Listing", nele poderá constatar que o endereço 100H se localiza na linha do caracter "I".

46.9 – EXEMPLO DE UMA VARREDURA DE TECLADO

Em um projeto, sempre há a necessidade de se elaborar um programa que faz a "leitura" de um determinado número de botões ou de teclas. Esta operação, conhecida por "SCAN" ou "VARREDURA", é comandada por uma interrupção gerada pelo CPU, causada pelo estouro de uma contagem (overflow) do timer TMR0.
A prática é quem aponta qual o intervalo ideal de varredura, pois, ela está relacionada com a quantidade de pontos de "leitura", e do intervalo mínimo ou máximo envolvido.

No exemplo exposto, um led aceso irá se deslocar em uma direção definida, com uma velocidade ajustável .
A cada 16 ms ocorre o estouro do timer TMR0, que interrompe o deslocamento do led, e verifica se existe alguma tecla apertada. Se nenhuma tecla estiver apertada, esta interrupção termina em RETFIE, que retoma o curso do programa que foi interrompido.

Esta retomada, que ocorre sem causar falhas, após o retorno de uma "leitura", se justifica pelo seguinte:

- antes de o programa seguir para rotina de leitura de TECLAS, o W e o STATUS, são salvos em W2 e STATUS2 respectivamente.
- Logo após o retorno da rotina TECLAS, o W e o STATUS, são restabelecidos com valores anteriores à interrupção. O comando RETFIE é o responsável pela memorização da posição do endereço no PC.

```
;=====================================================================
; Objetivo : É implementar um software que aceita o comando das 4 teclas. A tecla 0,
; roda um led à esquerda;  a tecla 1,  à direita; a tecla 2,  aumenta a velocidade , e     a
; tecla 3 reduz a velocidade.                    APLICÁVEL  NO CIRCUITO EXPERIMENTAL
;=====================================================================
            list p = 16F84
            Radix dec
            Include <P16F84A.INC>
;------------------------------------------ define fusíveis ----------------------------------------
            __config  _xt_osc & _cp_off & _wdt_off & _pwrte_on
;------------------------------------------- define simulação ----------------------------------------
;#DEFINE  SIMULADO                ";" cancela #DEFINE SIMULADO
;---------------------------------- espaço na RAM ----------------------------------------------
TEMP_1:   EQU       012                 ; REGISTRO1 TEMPORÁRIO VARIÁVEL
TM1:      EQU       013                 ; VARIÁVEL1 DO TIMER
TM2:      EQU       014                 ; VARIÁVEL2 DO TIMER
STATUS2:  EQU       015                 ; COPIA DO STATUS
W2:       EQU       016                 ; COPIA DO W
VELOC:    EQU       017                 ; CONSTANTE DE VELOCIDADE
;         ------------------------------------- endereço de RESET ------------------------------------
            ORG       000H
            GOTO      INICIO              ; VAI PARA O INICIO
;           ------------------------------------- endereço da INTERRUPÇÃO -------------------------------

            ORG       04H                 ; A interrupção por T0 cai aqui

            MOVWF     W2                  ; SALVA W ORIGINAL
            MOVF      STATUS,W            ; COPIA STATUS ORIGINAL E W
            MOVWF     STATUS2             ; SALVA STATUS ORIGINAL EM STATUS2
            GOTO      TECLAS              ; VERIFICA AS TECLAS
FIM_INT:
            MOVF      W2,W                ; RESTAURA O W ANTERIOR
            MOVF      STATUS2,W           ; COPIA EM W O STATUS2
            MOVWF     STATUS              ; RESTABELECE O VALOR ORIGINAL DE STATUS
            MOVLW     B'10100000'         ; CARREGA W COM B'10100000'
            MOVWF     INTCON              ; REABILITA A INTERRUPÇÃO
            RETFIE                        ; FIM DA INTERRUPÇÃO

;           ------------------------------------- define PRESCALER ----------------------------------------
INICIO:
            CLRF      INTCON              ; DESABILITA INTERRUPÇÃO
            BSF       STATUS,RP0          ; ACESSA BANCO1
#IFDEF    SIMULADO
            MOVLW     B'10000000'         ; WD INT SEM DIVISÃO
#ELSE
            MOVLW     B'10000101'         ; T0 INT DIVISÃO 1:64 (16 ms)
#ENDIF
            MOVWF     OPTION_REG          ; DEFINE O MODO DO PRESCALER
            BCF       STATUS,RP0          ; RETORNA AO BANCO0
            MOVLW     150                 ; CARREGA W COM 150
            MOVWF     VELOC               ; CARREGA VELOC COM DELAY INICIAL

;           ------------------------------------- condições iniciais do PIC ---------------------------------

                                          ; O PORTA fica como entrada (default)
            CLRF      PORTA               ; LIMPA PORTA
            CLRF      PORTB               ; LIMPA PORTB
            BSF       STATUS,RP0          ; ACESSA O BANCO1
            CLRF      TRISB               ; HABILITA PORTB COMO SAÍDA
            BCF       STATUS,RP0          ; VOLTA PARA O BANCO0
            BSF       PORTB,0             ; INICIA COM O LED0 ACESO

;           ------------------------------------- habilitação da interrupção ---------------------------------

            MOVLW     B'10100000'         ; CARREGA W COM STATUS DE INTERRUPÇÃO
            MOVWF     INTCON              ; HABILITA A INTERRUPÇÃO T0

;====================== PROGRAMA PRINCIPAL==============================

RODA_ESQ: ; Roda Led à esquerda e fica aguardando uma interrupção T0

            MOVLW     B'10100000'         ; CARREGA W COM STATUS DE INTERRUPÇÃO
            MOVWF     INTCON              ; HABILITA INTERRUPÇÃO T0
```

```
LOOP3:  BCF     STATUS,C        ; LIMPA O FLAG CARRY
        CALL    DELAY           ; DELAY VARIÁVEL
        RLF     PORTB,W         ; RODA O PORTB A ESQUERDA E FICA EM W
        BTFSC   STATUS,C        ; SE NÃO EXISTE CARRY, PULA LINHA
        GOTO    CONT1           ; SE NÃO, REINICIO
        MOVWF   PORTB           ; ATUALIZA O VALOR DE PORTB
        GOTO    LOOP3           ; CONTINUA A RODAR À ESQUERDA
CONT1:  CALL    DELAY           ; DELAY VARIÁVEL
        CLRF    PORTB           ; LIMPA O PORTB
        BSF     PORTB,0         ; ASCENDE O LED0
        GOTO    LOOP3           ; REINICIA NOVAMENTE
;       ---------- tratamento das teclas ----------
TECLAS: ; uma interrupção T0, cai aqui para identificar a tecla pressionada

        MOVF    PORTA,W         ; COPIA O VALOR DO PORTA EM W
        MOVWF   TEMP_1          ; SALVA EM TEMP_1
        BTFSS   TEMP_1,0        ; SE A TECLA 0 ACIONADA, VAI PARA RODA_ESQ
        GOTO    RODA_ESQ        ; SE NÃO, PULA LINHA
        BTFSS   TEMP_1,1        ; SE A TECLA 1 ACIONADA, VAI PARA RODA_DIR
        GOTO    RODA_DIR        ; SE NÃO, PULA LINHA
        BTFSS   TEMP_1,2        ; SE A TECLA 2 ACIONADA, VAI PARA VELOC_S
        GOTO    VELOC_S         ; SE NÃO, PULA LINHA
        BTFSS   TEMP_1,3        ; SE A TECLA 3 ACIONADA, VAI PARA VELOC_D
        GOTO    VELOC_D         ; SE NÃO, PULA LINHA
        GOTO    FIM_INT         ; RETORNA AO PROGRAMA EM CURSO
;       ---------- operações programadas por teclas ----------
RODA_DIR:       ; Roda Led à direita e aguarda uma interrupção T0

        MOVLW   B'10100000'     ; CARREGA W COM STATUS DE INTERRUPÇÃO
        MOVWF   INTCON          ; HABILITA INTERRUPÇÃO T0
LOOP4:  BCF     STATUS,C        ; LIMPA O FLAG CARRY
        CALL    DELAY           ; DELAY VARIÁVEL
        RRF     PORTB,W         ; RODA PORTB P/ A DIREITA E SALVA EM W
        BTFSC   STATUS,C        ; SE NÃO EXISTE CARRY, PULA LINHA
        GOTO    CONT2           ; SE NÃO, REINICIO
        MOVWF   PORTB           ; ATUALIZA O VALOR DE PORTB
        GOTO    LOOP4           ; CONTINUA A RODAR À DIREITA
CONT2:  CALL    DELAY           ; DELAY VARIÁVEL
        CLRF    PORTB           ; LIMPA O PORTB
        BSF     PORTB,7         ; ACENDE O LED7
        GOTO    LOOP4           ; REINICIA NOVAMENTE
;       ----------
VELOC_S:        ; Aqui, a velocidade de movimento do led aumenta

        BCF     STATUS,Z        ; LIMPA FLAG ZERO
        DECFSZ  VELOC,W         ; DECREMENTA VELOC, SE Z=1, PULA
        MOVWF   VELOC           ; SE NÃO, ATUALIZA VELOC
        GOTO    FIM_INT         ; ENCERRA
;       ----------
VELOC_D:        ; Aqui, a velocidade de movimento do led cai

        BCF     STATUS,Z        ; LIMPA FLAG ZERO
        INCFSZ  VELOC,W         ; INCREMENTA VELOC, SE Z=1, PULA
        MOVWF   VELOC           ; SE NÃO, ATUALIZA VELOC
        GOTO    FIM_INT         ; ENCERRA
;       ---------- subrotina delay ----------
DELAY:                          ; Delay com tempo variável
#IFDEF  SIMULADO
        RETURN                  ; NO MODO SIMULADO, NÃO USA DELAY
#ELSE
        MOVF    VELOC,W         ; CARREGA W COM O VALOR VELOC
        MOVWF   TM1             ; COPIA O VELOC NA VARIÁVEL TM1

LOOP1:  MOVF    VELOC,W         ; CARREGA W COM O VALOR VELOC
        MOVWF   TM2             ; COPIA O VELOC NA VARIÁVEL TM2
LOOP2:
        DECFSZ  TM2,F           ; DECREMENTA TM2, SE Z=1, PULA UMA LINHA
        GOTO    LOOP2           ; CONTINUA A DECREMENTAR TM2 EM LOOP2
        DECFSZ  TM1,F           ; DECREMENTA TEMP_1, SE Z=1, PULA  LINHA
        GOTO    LOOP1           ; RETORNA PARA LOOP1
        RETURN
#ENDIF
;       ----------
        END                     ; Fim desta programação
```

47 – AS PARTICULARIDADES DE UMA SUBTRAÇÃO

Numa operação aritmética entre dois números quaisquer, ocorrem particularidades que requerem uma compreensão dos projetistas dos software. Veja o porquê:
Por exemplo, seja dois números quaisquer, denominados NUM_1 e NUM_2. Ao subtrair o primeiro do segundo, os flags Carry e Zero, poderão assumir um valor zero (0) ou um valor um (1), conforme o resultado final desta operação.
Vamos elaborar um programa que irá simular esta situação:
Subtrai dois números quaisquer, e só aceita em RESP_POS o resultado que der positivo.
Veja como fica:

```
CLRF        RESP_POS    ; LIMPA A RESP_POS   ANTES
MOVF        NUM_1,W     ; CARREGA EM W O VALOR DE NUM_1
SUBWF       NUM_2,W     ; SUBTRAI NUM_1 – NUM_2 =  W
BTFSC       STATUS,C    ; SE CARRY=0, SALTA (NEGATIVO, SALTA)
MOVWF       RESP_POS    ; SE O FOR POSITIVO, SALVA EM RESP_POS
GOTO        SEGUE       ; O PROGRAMA CONTINUA....
```

Portanto, é importante lembrar que, numa subtração, se o resultado der:

- **NEGATIVO** : O Carry será zero (0).
- **POSITIVO** : O Carry será um (1).
- **ZERO** : O Carry será um (1)e o flag do Zero também será um (1).

NOTA:
- O valor de um resultado negativo, não será diretamente um número negativo, e sim, a sua diferença para 256.

COM UMA OPERAÇÃO DE SUBTRAÇÃO, É POSSÍVEL CONSTATAR QUE UM NÚMERO CONHECIDO, EM RELAÇÃO AO OUTRO, É : MAIOR, MENOR , OU IGUAL.

48) – SIMULANDO UM CONTADOR DIGITAL

Este programa apresenta o display da placa experimental contando de “zero” até “F” em intervalo de 1 segundo (aproximado).

Na subrotina “CONVERTE”, A instrução “ADDW PCL,F” já vista anteriormente, retorna em W com o valor decodificado correspondente em sete segmento (tabela) e carrega no PORTB.

```
;*****************************************************************************************
; ESTE PROGRAMA DEMONSTRA A CONVERSÃO DE UM VALOR HEXADECIMAL EM DIPLAY DE 7 SEGMENTOS.
; PARA FACILITAR A DEMONSTRAÇÃO,     FOI IMPLEMENTADO UM CONTADOR QUE APRESENTA NO DISPLAY
; DE "0" ATÉ "F" (16), COM INTERVALO DE 1,0 SEGUNDO.
;                                                    APLICÁVEL NO CIRCUITO EXPERIMENTAL
;*****************************************************************************************
;                          a
;                        xxxxxxx                 a - RB0
;                       x       x                b - RB1
;                    f  x       x  b             c - RB2
;                       x   g   x                d - RB3
;                        xxxxxxx                 e - RB4
;                       x       x                f - RB5
;                    e  x       x  c             g - RB6
;                       x   d   x                P - RB7
;                        xxxxxxx  X P
;-----------------------------------------------------------------------------------------------------------------
            list p = 16F84
```

```
        Radix dec
        Include <P16F84A.INC>
;------------------------------------------------- define fusíveis ---------------------------------------------

        __config _xt_osc & _cp_off & _wdt_off & _pwrte_on

;------------------------------------------ definições de espaços na RAM ----------------------------------------
-
TM1:    EQU     12      ; RESERVA A POSIÇÃO NA RAM (UTILIZADO NO DELAY)
TM2:    EQU     13      ; RESERVA A POSIÇÃO NA RAM (UTILIZADO NO DELAY)
CONT:   EQU     14      ; RESERVA A POSIÇÃO NA RAM (CONTADOR)
TEMP_1: EQU     15      ; RESERVA A POSIÇÃO NA RAM (LIMITE DE CONTAGEM)
TEMP_2: EQU     16      ; RESERVA A POSIÇÃO NA RAM (QTD DE DELAY)

; ------------------------------------------ início do programa principal ----------------------------------------

        ORG     00H             ; INÍCIO DA GRAVAÇÃO

        CLRF    PORTB           ; ZERA A SAÍDA PORTB
        BSF     STATUS,RP0      ; VAI PARA O BANCO1
        CLRF    TRISB           ; HABILITA O PORTB COMO SAÍDA
        BCF     STATUS,RP0      ; RETORNA PARA O BANCO0

GET_CONV:

        MOVLW   16              ; CARREGA W COM 16 (QTD LINHAS PARA CONVERSÃO)
        MOVWF   TEMP_1          ; COPIA W EM TEMP_1
        CLRF    CONT            ; INICIA ZERANDO O CONTADOR

LOOP:   MOVLW   04              ; CARREGA W COM 04 (QTD DE DELAY)
        MOVWF   TEMP_2          ; COPIA W EM TEMP_2
        CALL    CONVERTE        ; FAZ A CONVERSÃO PARA 7 SEGUIMENTO
        MOVWF   PORTB           ; ENVIA CARACTER PARA O PORTB (DISPLAY)
        CALL    DELAY           ; DELAY DE 1SEGUNDO
        DECFSZ  TEMP_2          ; DECREMENTA QTD DE DELAY
        GOTO    $-2             ; REPETE DELAY (250ms x 4 = 1,0 S)

        INCF    CONT,F          ; INCREMENTA O CONTADOR
        DECFSZ  TEMP_1          ; DECREMENTA QTD DE LINHAS PARA CONVERSÃO
        GOTO    LOOP            ; REPETE EM LOOP
        GOTO    GET_CONV        ; INÍCIO DA CONVERSÃO



;------------------------------------------- tabela de conversão hexa/7seguimentos ------------------------------

CONVERTE:

        MOVF    CONT,W          ; CARREGA W COM O VALOR DO CONTADOR
        ANDLW   B'00001111'     ; ANULA OS BITS DO LADO ESQUERDO (ATÉ 15)


        ADDWF   PCL,F

;                   'PGFEDCBA'  ; POSIÇÃO DO DISPLAY NA PLACA EXPERIMENTAL
        RETLW   B'00111111'     ; 00 - CONVERTIDO FICA : 00
        RETLW   B'00000110'     ; 01 - CONVERTIDO FICA : 01
        RETLW   B'01011011'     ; 02 - CONVERTIDO FICA : 02
        RETLW   B'01001111'     ; 03 - CONVERTIDO FICA : 03
        RETLW   B'01100110'     ; 04 - CONVERTIDO FICA : 04
        RETLW   B'01101101'     ; 05 - CONVERTIDO FICA : 05
        RETLW   B'01111100'     ; 06 - CONVERTIDO FICA : 06
        RETLW   B'00000111'     ; 07 - CONVERTIDO FICA : 07
        RETLW   B'01111111'     ; 08 - CONVERTIDO FICA : 08
        RETLW   B'01100111'     ; 09 - CONVERTIDO FICA : 09
        RETLW   B'01110111'     ; 10 - CONVERTIDO FICA : 0A
        RETLW   B'01111100'     ; 11 - CONVERTIDO FICA : 0B
        RETLW   B'00111001'     ; 12 - CONVERTIDO FICA : 0C
        RETLW   B'01011110'     ; 13 - CONVERTIDO FICA : 0D
        RETLW   B'01111001'     ; 14 - CONVERTIDO FICA : 0E
        RETLW   B'01110001'     ; 15 - CONVERTIDO FICA : 0F

;-------------------- subrotina delay de 250 ms ----------------------
```

```
DELAY:                          ; NOME DO LABEL DESTA SUBROTINA DELAY

        MOVLW   251             ; CARREGA W COM O VALOR 51
        MOVWF   TM1             ; COPIA O W NA VARIÁVEL TM1

LOOP1:  MOVLW   248             ; CARREGA W COM O VALOR 248
        MOVWF   TM2             ; COPIA O W NA VARIÁVEL TM2

LOOP2:  NOP                     ; PERDE UM CICLO DE MÁQUINA
        DECFSZ  TM2,f           ; DECREMENTA TM2, SE Z=1, PULA UMA LINHA
        GOTO    LOOP2           ; CONTINUA A DECREMENTAR TM2 EM LOOP2
        DECFSZ  TM1,f           ; DECREMENTA TM1, SE Z=1, PULA UMA LINHA
        GOTO    LOOP1           ; RETORNA PARA LOOP1
        RETURN                  ; VOLTA PARA A ROTINA PRINCIPAL

        END                     ; FIM DESTA PROGRAMAÇÃO
```

49 – OPERAÇÕES ARITMÉTICAS BINÁRIAS

49.1 – SOMANDO DO1S NÚMEROS DE 2 BYTES CADA

A soma de duas parcelas de dois bytes cada, requer o cuidado de iniciar a adição com os bytes menos significativos (LSB_1 + LSB_2), e depois os bytes mais significativos (MSB)_1 + MSB_2).

Se na adição das parcelas menos significativos (LSB_1 + LSB_2) ocasionar um “carry” (“vai um”), antes de somar as duas parcelas mais significativos (MSB_1 + MSB_2), é importante incrementar o MSB_1.

```
;================================================================================================
; ESTE PROGRAMA EFETUA A SOMA BINÁRIA DE DUAS  PARCELAS DE DOIS BYTES  CADA
; A PRIMEIRA  PARCELA  FICA  EM : "PARCELA_1_H"  E  "PARCELA_1_L" (420FH).
; A SEGUNDA  PARCELA  FICA  EM : "PARCELA_2_H"  E  "PARCELA_2_L"  (400FH).
; APÓS A OPERAÇÃO SOMA, O RESULTADO FICA NO LUGAR DA SEGUNDA PARCELA (MSB+LSB)
; COM  A  CALCULADORA  D0  WINDOWS (HEXA),  PODEREMOS  VERIFICAR  OS  RESULTADOS
;================================================================================================

        list p = 16F84
        Radix dec
        Include <P16F84A.INC>
;---------------------------------------- define fusíveis-----------------------------------

        __config _xt_osc  &  _cp_off  &  _wdt_off  &  _pwrte_on

;---------------------------------- espaço na RAM ---------------------------------------

PARCELA_1_H:    EQU     012     ; RESERVA A POSIÇÃO  NA RAM (MSB DA PARCELA_1)
PARCELA_1_L:    EQU     013     ; RESERVA A POSIÇÃO  NA RAM (LSB DA PARCELA_1)
PARCELA_2_H:    EQU     014     ; RESERVA A POSIÇÃO  NA RAM (MSB DA PARCELA_2)
PARCELA_2_L:    EQU     015     ; RESERVA A POSIÇÃO  NA RAM (LSB DA PARCELA_2)


        ORG     00H             ; INÍCIO DA GRAVAÇÃO

        MOVLW   42H             ; CARREGA O MSB DA PRIMEIRA PARCELA EM W
        MOVWF   PARCELA_1_H     ; COPIA W EM PARCELA_1_H
        MOVLW   0FH             ; CARREGA O LSB DA PRIMEIRA PARCELA EM W
        MOVWF   PARCELA_1_L     ; COPIA W EM PARCELA_1_L

        MOVLW   40H             ; CARREGA O MSB DA SEGUNDA PARCELA EM W
        MOVWF   PARCELA_2_H     ; COPIA W EM PARCELA_2_H
        MOVLW   0FH             ; CARREGA O LSB DA PRIMEIRA PARCELA EM W
        MOVWF   PARCELA_2_L     ; COPIA W EM PARCELA_2_L
```

```
        CALL    SOMA            ; EXECUTA A OPERAÇÃO SOMA
        GOTO    FIM             ; FIM DESTE PROGRAMA

SOMA:                           ; SUBROTINA DA OPERAÇÃO SOMA

        MOVF    PARCELA_1_L,W   ; COPIA O LSB DA PRIMEIRA PARCELA EM W
        ADDWF   PARCELA_2_L,F   ; ADICIONA W COM LSB DA SEGUNDA PARCELA
        BTFSC   STATUS,C        ; SE OCORREU UM CARRY, PULA UMA LINHA
        INCF    PARCELA_2_H,F   ; INCREMENTA MSB DA PRIMEIRA PARCELA
        MOVF    PARCELA_1_H,W   ; COPIA P MSB DA PRIMEIRA PARCELA EM W
        ADDWF   PARCELA_2_H,F   ; ADICIONA W COM MSB DA SEGUNDA PARCELA
        RETURN                  ; RETORNA
FIM:

        END                     ; FIM DESTA PROGRAMAÇÃO
```

49.2 – SUBTRAINDO DUAS PARCELAS DE 2 BYTES CADA

A subtração de duas parcela (Parcela_1 - Parcela_2) de dois bytes cada, necessita o cuidado de verificar se a subtração entre os dois bytes menos significativos (LSB_1 – LSB_2) não ocorrer o “Carry” (resultado deu negativo!), é necessário decrementar o primeiro byte mais significativo (MSB_1) antes de subtraí-lo da segunda parcela (MLS_2).

```
;*******************************************************************************
; ESTE PROGRAMA EFETUA A SUBTRAÇÃO BINÁRIA DE DUAS PARCELAS DE DOIS BYTES CADA
; A PRIMEIRA PARCELA FICA EM : "PARCELA_1_H" E "PARCELA_1_L" (520FH).
; A SEGUNDA PARCELA FICA EM : "PARCELA_2_H" E "PARCELA_2_L"  (410EH).
; APÓS A SUBTRAÇÃO, O RESULTADO FICA NO LUGAR DA PRIMEIRA PARCELA (MSB+LSB)
; COM A CALCULADORA D0 WINDOWS (HEXA), PODEREMOS VERIFICAR OS RESULTADOS
;*******************************************************************************************
        list p = 16F84
        Radix dec
        Include <P16F84A.INC>
;------------------------------------------------ define fusíveis-----------------------------------------
        __config _xt_osc  &  _cp_off  &  _wdt_off  &  _pwrte_on

;------------------------------------------- espaço na RAM ------------------------------------------

PARCELA_1_H:    EQU     012     ; RESERVA A POSIÇÃO  NA RAM (MSB DA PARCELA_1)
PARCELA_1_L:    EQU     013     ; RESERVA A POSIÇÃO  NA RAM (LSB DA PARCELA_2)
PARCELA_2_H:    EQU     014     ; RESERVA A POSIÇÃO  NA RAM (MSB DA PARCELA_2)
PARCELA_2_L:    EQU     015     ; RESERVA A POSIÇÃO  NA RAM (LSB DA PARCELA_2)


        ORG     00H             ; INÍCIO DA GRAVAÇÃO

        MOVLW   52H             ; CARREGA O MSB DA PRIMEIRA PARCELA EM W
        MOVWF   PARCELA_1_H     ; COPIA W EM PARCELA_1_H
        MOVLW   0FH             ; CARREGA O LSB DA PRIMEIRA PARCELA EM W
        MOVWF   PARCELA_1_L     ; COPIA W EM PARCELA_1_L

        MOVLW   41H             ; CARREGA O MSL DA SEGUNDA PARCELA EM W
        MOVWF   PARCELA_2_H     ; COPIA W EM PARCELA_2_H
        MOVLW   0EH             ; CARREGA O LSB DA PRIMEIRA PARCELA EM W
        MOVWF   PARCELA_2_L     ; COPIA W EM PARCELA_2_L

        CALL    SUBTRAÇÃO       ; EXECUTA A OPERAÇÃO DE SUBTRAÇÃO
        GOTO    FIM             ; FIM DESTE PROGRAMA

SUBTRAÇÃO:                      ; SUBROTINA DA OPERAÇÃO DE SUBTRAÇÃO

        MOVF    PARCELA_2_L,W   ; COPIA O LSB DA PRIMEIRA PARCELA EM W
        SUBWF   PARCELA_1_L,F   ; ADICIONA W COM LSB DA SEGUNDA PARCELA
        BTFSS   STATUS,C        ; SE OCORREU CARRY (POSITIVO), PULA UMA LINHA
        DECF    PARCELA_1_H     ; DECREMENTA MSB DA PARCELA_1
```

```
        MOVF    PARCELA_2_H,W   ; COPIA P MSB DA PRIMEIRA PARCELA EM W
        SUBWF   PARCELA_1_H,F   ; ADICIONA W COM MSB DA SEGUNDA PARCELA
        RETURN                  ; RETORNA
FIM:

        END                     ; FIM DESTA PROGRAMAÇÃO
```

49.3 – MULTIPLICANDO DUAS PARCELAS DE 2 BYTES CADA

```
;*****************************************************************************************
; ESTE PROGRAMA EFETUA A MULTIPLICAÇÃO BINÁRIA DE DUAS  PARCELAS DE DOIS BYTES CADA.
; O MULTUPLICADOR FICA EM : "MCADOR_H" E "MCADOR_L" (420FH) (16 BITS).
; O MULTIPLICANDO FICA EM : "MCANDO_H" E "MCANDO_L" (400FH) (16 BITS).
; O RESULTADO FICA NO LUGAR D0 MCANDO_HH,MCANDO_HL,MCANDO_LH,MCANDOLL (32 BITS)
; COM A CALCULADORA D0 WINDOWS (HEXA), PODEREMOS VERIFICAR OS RESULTADOS
; (16 BITS * 16 BITS = 32 BITS)
;*****************************************************************************************
        list p = 16F84
        Radix dec
        Include <P16F84A.INC>
;------------------------------------------------- define fusíveis-----------------------------------------------

        __config _xt_osc  &  _cp_off  &  _wdt_off  &  _pwrte_on

;------------------------------------------------- espaço na RAM ------------------------------------------------

MCADOR_H:       EQU     012     ; RESERVA A POSIÇÃO  NA RAM (MULTIPLICADOR H)
MCADOR_L:       EQU     013     ; RESERVA A POSIÇÃO  NA RAM (MULTIPLICADOR L)
MCANDO_HH:      EQU     014     ; RESERVA A POSIÇÃO  NA RAM (MULTIPLICANDO HH)
MCANDO_HL:      EQU     015     ; RESERVA A POSIÇÃO  NA RAM (MULTIPLICANDO HL)
MCANDO_LH:      EQU     016     ; RESERVA A POSIÇÃO  NA RAM (MULTIPLICANDO LH)
MCANDO_LL:      EQU     017     ; RESERVA A POSIÇÃO  NA RAM (MULTIPLICANDO LL)
AC_H:           EQU     018     ; RESERVA A POSIÇÃO  NA RAM (AUXILIAR CARRY H)
AC_L:           EQU     019     ; RESERVA A POSIÇÃO  NA RAM (AUXILIAR CARRY L)
TEMP:           EQU     020     ; RESERVA A POSIÇÃO  NA RAM (ARQUIVO TEMP)


        ORG     00H             ; INÍCIO DA GRAVAÇÃO

        MOVLW   42H             ; CARREGA O MSB DO MULTIPLICADOR EM W
        MOVWF   MCADOR_H        ; COPIA W EM MCADOR_H
        MOVLW   0FH             ; CARREGA O LSB DO MULTIPLICADOR EM W
        MOVWF   MCADOR_L        ; COPIA W EM MCADOR_L
        MOVLW   40H             ; CARREGA O MSB DO MULTIPLICANDO EM W
        MOVWF   MCANDO_HH       ; COPIA W EM MCANDO_HH
        MOVLW   0FH             ; CARREGA O LSB DO MULTIPLICANDO EM W
        MOVWF   MCANDO_HL       ; COPIA W EM MCANDO_HL

        CALL    MULTIPLICA      ; EXECUTA A MULTIPLICAÇÃO
        GOTO    FIM             ; FIM DESTE PROGRAMA

MULTIPLICA:                     ; SUBROTINA DE MULTIPLICAÇÃO

        MOVLW   16              ; CARREGA W COM 16
        MOVWF   TEMP            ; NECESSÁRIO PARA 16 DESLOCAMENTOS
        MOVF    MCANDO_HH,W     ; COPIA MCANDO_HH EM W
        MOVWF   AC_H            ; COPIA W EM AC_H
        MOVF    MCANDO_HL,W     ; COPIA MCANDO_HL EM W
        MOVWF   AC_L            ; COPIA W EM AC_L
        CLRF    MCANDO_HH       ; LIMPA MCANDO_HH
        CLRF    MCANDO_HL       ; LIMPA MCANDO_HL
        BCF     STATUS,C        ; LIMPA FLAG DE CARRY
LOOP1:
        RRF     AC_H,F          ; RODA A DIREITA O AC_H
        RRF     AC_L,F          ; RODA A DIREITA O AC_L
        BTFSS   STATUS,C        ; PULA UMA LINHA SE HÁ CARRY
        GOTO    NO_SOMA         ; VAI PARA NO_SOMA
        MOVF    MCADOR_L,W      ; COPIA EM W O LSB DO MULTIPLICADOR
        ADDWF   MCANDO_HL,F     ; MCADOR_L + MCANDO_HL -> MCANDO_HL
```

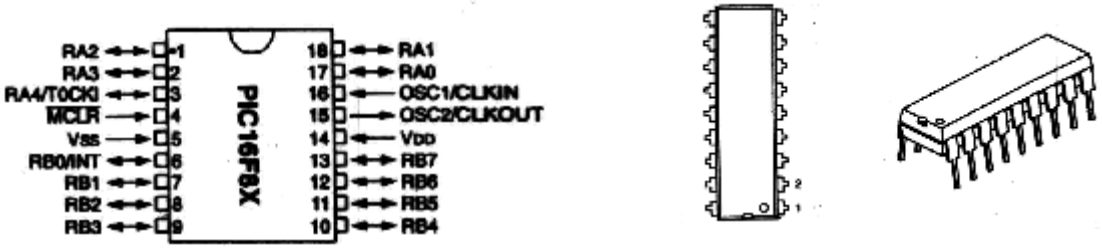
```
        BTFSC   STATUS,C        ; PULA UMA LINHA SE NÃO HÁ CARRY
        INCF    MCANDO_HH,F     ; INCREMENTA MCANDO_HH
        MOVF    MCADOR_H,W      ; COPIA EM W O MSB DO MULTIPLICADOR
        ADDWF   MCANDO_HH,F     ; MCADOR_H + MCANDO_HH -> MCANDO_HH
NO_SOMA:
        RRF     MCANDO_HH,F     ; RODA À DIREITA MCANDO_HH
        RRF     MCANDO_HL,F     ; RODA À DIREITA MCANDO_HL
        RRF     MCANDO_LH,F     ; RODA À DIREITA MCANDO_LH
        RRF     MCANDO_LL,F     ; RODA À DIREITA MCANDO_LL
        DECFSZ  TEMP,F          ; DECREMENTA TEMP
        GOTO    LOOP1           ; SE NÃO ZEROU,VAI PARA LOOP1

        RETURN                  ; RETORNA
FIM:

        END                     ; FIM DESTA PROGRAMAÇÃO
```

49.4 – DIVIDINDO DOIS NÚMEROS DE 3 BYTES CADA

```
;=========================================================================
; ESTE PROGRAMA EFETUA UMA DIVISÃO DE UM DIVIDENDO (24 BITS)/DIIVISOR (24 BITS)
; O DIVISOR É CARREGADO EM "DIVISOR_HH,DIVISOR_HL & DIVISOR_LL (24 BITS)
; O DIIVIDENDO É CARREGADO EM "DIVIDENDO_HH, DIVIDENDO_HL & DIVIDENDO_LL (24 BITS)
; O RESULTADO É GUARDADO EM DIVIDENDO_HH, DIVIDENDO_HL & DIVIDENDO_LL
; E O RESTO EM RESTO_HH, RESTO_HL & RESTO_LL
; COM A CALCULADORA D0 WINDOWS (HEXA), PODEREMOS VERIFICAR OS RESULTADOS
; (24 BITS / 24 BITS = 24 BITS) E RESTO (24 BITS)   -  (FFEF0AH) / (000105H)
;=========================================================================
        list p = 16F84
        Radix dec
        Include <P16F84A.INC>

;----------------------------- define fusíveis----------------------------

        __config _xt_osc  &  _cp_off  &  _wdt_off  &  _pwrte_on

;------------------------- espaço na RAM --------------------------------

DIVISOR_HH:     EQU     012     ; RESERVA A POSIÇÃO  NA RAM
DIVISOR_HL:     EQU     013     ; RESERVA A POSIÇÃO  NA RAM
DIVISOR_LL:     EQU     014     ; RESERVA A POSIÇÃO  NA RAM
DIVIDENDO_HH:   EQU     015     ; RESERVA A POSIÇÃO  NA RAM
DIVIDENDO_HL:   EQU     016     ; RESERVA A POSIÇÃO  NA RAM
DIVIDENDO_LL:   EQU     017     ; RESERVA A POSIÇÃO  NA RAM
RESTO_HH:       EQU     018     ; RESERVA A POSIÇÃO  NA RAM
RESTO_HL:       EQU     019     ; RESERVA A POSIÇÃO  NA RAM
RESTO_LL:       EQU     020     ; RESERVA A POSIÇÃO  NA RAM
AC_HH:          EQU     021     ; RESERVA A POSIÇÃO  NA RAM
AC_HL:          EQU     022     ; RESERVA A POSIÇÃO  NA RAM
AC_LL:          EQU     023     ; RESERVA A POSIÇÃO  NA RAM
TEMP:           EQU     024     ; RESERVA A POSIÇÃO  NA RAM

        ORG     00H             ; INÍCIO DA GRAVAÇÃO

; Carregando DIVIDENDO e DIVISOR

        MOVLW   0FFH            ; CARREGA FFH EM W
        MOVWF   DIVIDENDO_HH    ; COPIA W EM DIVIDENDO_HH
        MOVLW   0EFH            ; CARREGA 0EFH EM W
        MOVWF   DIVIDENDO_HL    ; COPIA W EM DIVIDENDO_HL
        MOVLW   0AH             ; CARREGA 0AH EM W
        MOVWF   DIVIDENDO_LL    ; COPIA W EM DIVIDENDO_LL

        MOVLW   00H             ; CARREGA 00H EM W
        MOVWF   DIVISOR_HH      ; COPIA W EM DIVISOR_HH
        MOVLW   01H             ; CARREGA O 01H EM W
        MOVWF   DIVISOR_HL      ; COPIA W EM DIVISOR_HL
        MOVLW   05H             ; CARREGA 05H EM W
```

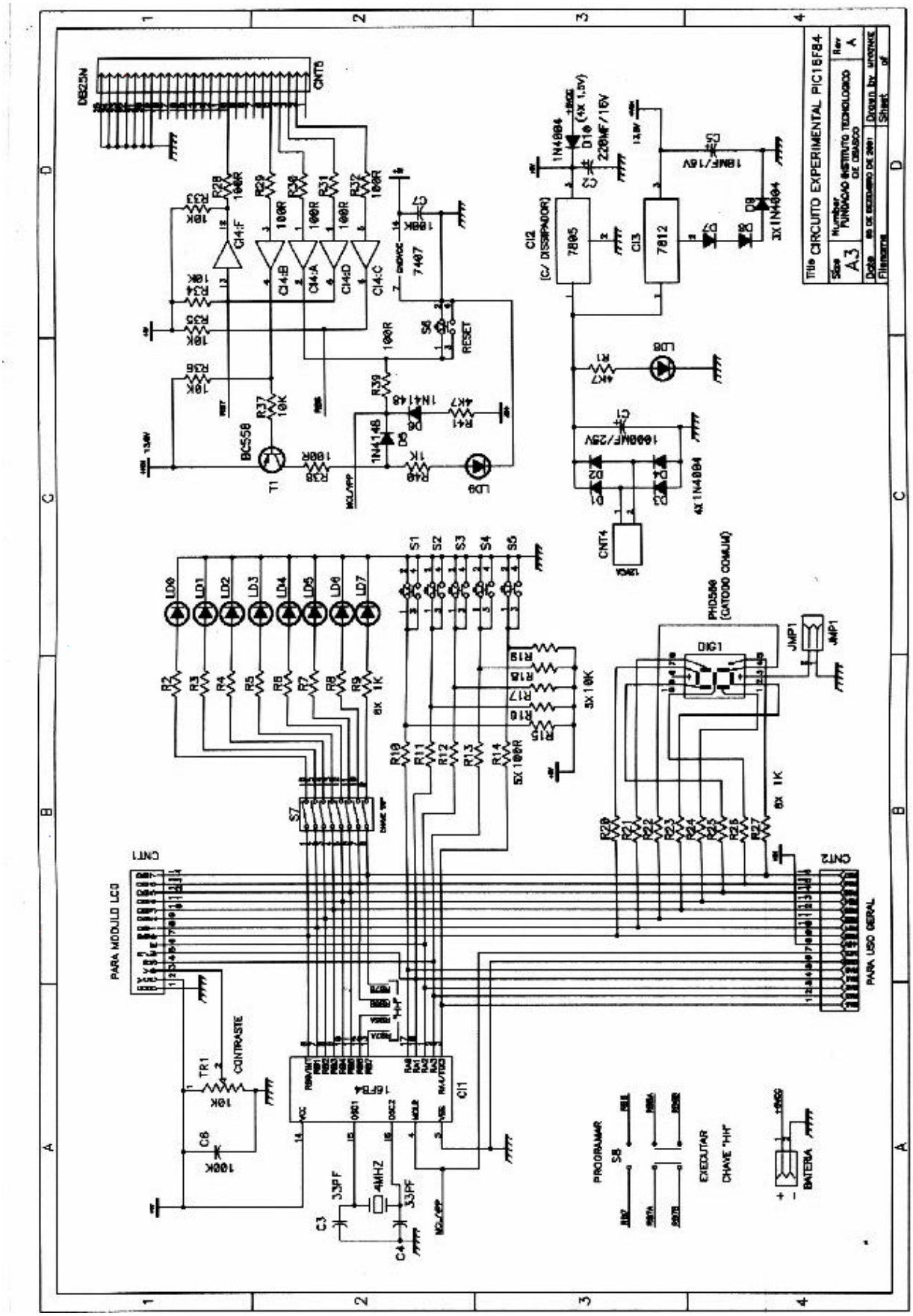
```
        MOVWF  DIVISOR_LL       ; COPIA W EM DIVISOR_LL

        CALL   DIVISÃO          ; EXECUTA A DIVISÃO
        GOTO   FIM
DIVISÃO:
        MOVLW  24               ; CARREGA W COM 24
        MOVWF  TEMP             ; NECESSÁRIO PARA DESLOCAMENTO

        MOVF   DIVIDENDO_HH,W ; CARREGA DIVIDENDO_HH EM W
        MOVWF  AC_HH            ; COPIA W EM AC_HH
        MOVF   DIVIDENDO_HL,W ; CARREGA DIVIDENDO_HL EM W
        MOVWF  AC_HL            ; COPIA W EM AC_HL
        MOVF   DIVIDENDO_LL,W ; CARREGA DIVIDENDO EM W
        MOVWF  AC_LL            ; COPIA W EM AC_LL

        CLRF   DIVIDENDO_HH     ; LIMPA DIVIDENDO-HH
        CLRF   DIVIDENDO_HL     ; LIMPA_DIVIDENDO_HL
        CLRF   DIVIDENDO_LL     ; LIMPA DIVIDENDO_LL
        CLRF   RESTO_HH         ; LIMPA RESTO_HH
        CLRF   RESTO_HL         ; LIMPA RESTO_HL
        CLRF   RESTO_LL         ; LIMPA RESTO_LL
LOOP1:
        BCF    STATUS,C         ; RESETA O CARRY
        RLF    AC_LL,F          ; RODA À ESQUERDA AC_LL
        RLF    AC_HL,F          ; RODA À ESQUERDA AC_HL
        RLF    AC_HH,F          ; RODA À ESQUERDA AC_HH
        RLF    RESTO_LL,F       ; RODA À ESQUERDA RESTO_LL
        RLF    RESTO_HL,F       ; RODA À ESQUERDA RESTO_HL
        RLF    RESTO_HH,F       ; RODA À ESQUERDA RESTO_HH

        MOVF   DIVISOR_HH,W     ; VERIFICA SE DIVISOR_HH > RESTO_HH
        SUBWF  RESTO_HH,W       ; CHECA A DIFERENÇA ENTRE AMBOS
        BTFSS  STATUS,Z         ; SE DIVISOR_HH = RESTO_HH, PULA 1 LINHA
        GOTO   LOOP2            ; SE NÃO, VAI PARA LOOP2

        MOVF   DIVISOR_HL,W     ; VERIFICA SE DIVISOR_HL > RESTO_HL
        SUBWF  RESTO_HL,W       ; CHECA A DIFERENÇA ENTRE AMBOS
        BTFSS  STATUS,Z         ; SE DIVISOR_HL = RESTO_HL, PULA 1 LINHA
        GOTO   LOOP2            ; SE NÃO, VAI PARA LOOP2
        MOVF   DIVISOR_LL,W     ; VERIFICA SE DIVISOR_LL > RESTO_LL
        SUBWF  RESTO_LL,W       ; CHECA A DIFERENÇA ENTRE AMBOS
LOOP2:
        BTFSS  STATUS,C         ; SE CARRY = 1, RESTO_LL > DIVISOR_LL
        GOTO   LOOP3            ; SE NÃO, VAI PARA LOOP3

        MOVF   DIVISOR_LL,W     ; RESTO_LL - DIVISOR_LL -> RESTO_LL
        SUBWF  RESTO_LL,F       ; VERIFICA A DIFERENCÁ ENTRE AMBOS
        BTFSS  STATUS,C         ; SE CARRY = 1, RESTO_LL > DIVISOR_LL
        DECF   RESTO_HL,F       ; DECREMENTA RESTO_HL

        MOVF   DIVISOR_HL,W     ; VERIFICA SE DIVISOR_HL > RESTO_HL
        SUBWF  RESTO_HL,F       ; CHECA A DIFERENÇA ENTRE AMBOS
        BTFSS  STATUS,C         ; SE CARRY =1, RESTO_HL > RESTO_HL
        DECF   RESTO_HH,F       ; DECREMENTA RESTO_HH

        MOVF   DIVISOR_HH,W     ; VERIFICA SE DIVISOR_HH > RESTO_HH
        SUBWF  RESTO_HH,F       ; SUBTRAI RESTO_HH - RESTO_HH ->RESTO_HH
        BSF    STATUS,C         ; SETA O CARRY
LOOP3:
        RLF    DIVIDENDO_LL,F   ; RODA DIVIDENDO_LL À ESQUERDA
        RLF    DIVIDENDO_HL,F   ; RODA DIVIDENDO_LL À ESQUERDA
        RLF    DIVIDENDO_HH,F   ; RODA DIVIDENDO_LL À ESQUERDA
        DECFSZ TEMP,F           ; DECREMENTA TEMP
        GOTO   LOOP1            ; SE NÃO ZEROU O TEMP, CONTINUA EM LOOP1

        RETURN                  ; RETORNA
FIM:

        END                     ; FIM DESTA PROGRAMAÇÃO
```

50 – FIGURA DO PIC 16F84

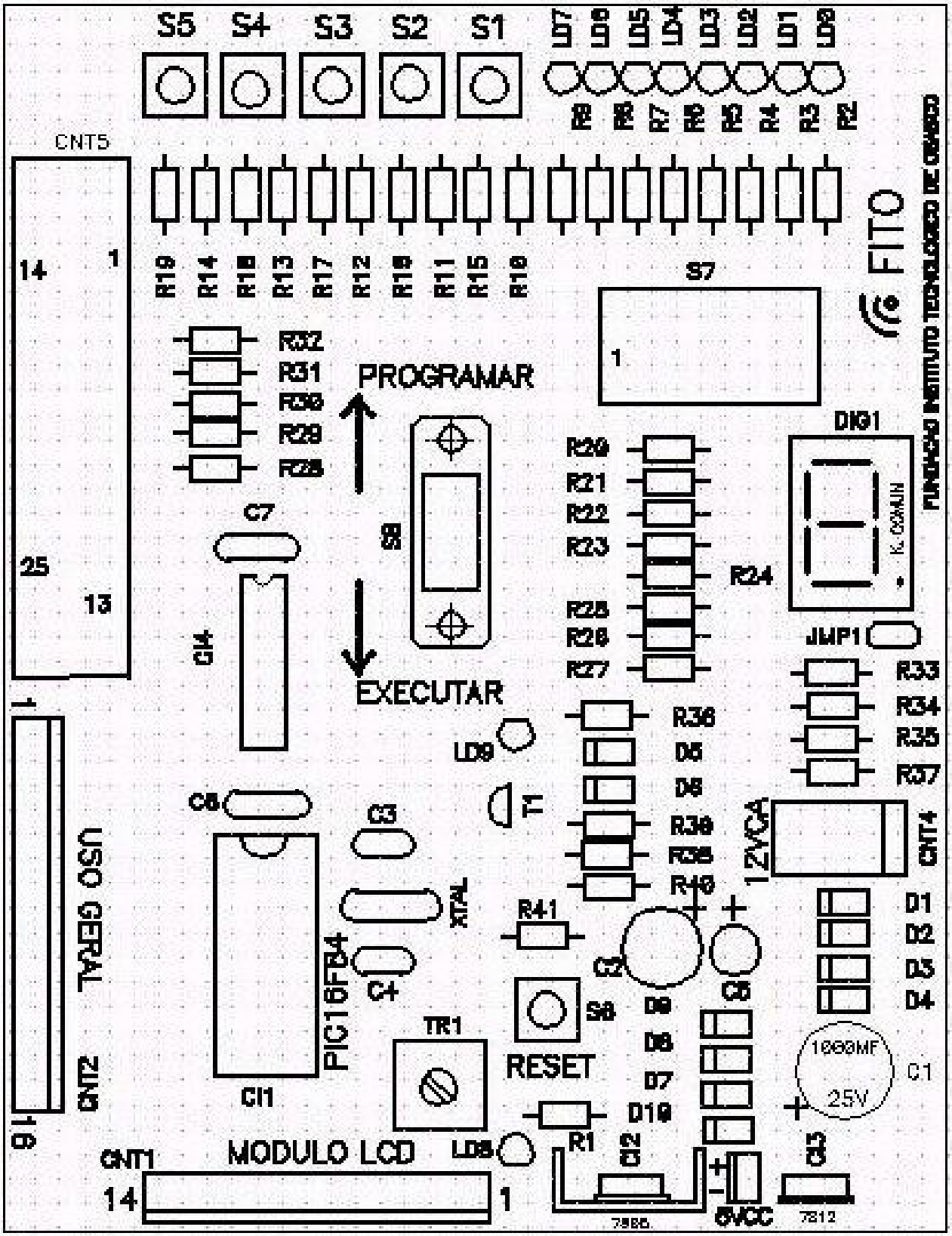


51 – LISTA DE MATERIAIS (CIRCUITO EXPERIMENTAL)

QTD	Valor	DESCRIÇÃO	REFERÊNCIA
1	7407	CI Buffer Hexa de 14 pinos	CI 4
2	100Kp	Capacitor Cerâmico	C6 e C7
2	33pF	Capacitor Cerâmico	C3 e C4
1	HH	Chave deslizante miniatura de 2 polos	S8
2	2 Vias	Conector "Burndy" para PCI	JMP1 e 6VCC
1	16 Vias	Conector "Burndy" para PCI	CNT2
1	Plug	Plug Fêmea para entrada de Fonte DC	CNT4
1	14Vias	Conector "Burndy" para PCI	CNT1
1	25 Vias	Conector macho DB25 para PCI	CNT5
8	1N4004	Diodo Retificador 1N4004	D1, D2, D3, D4, D7, D8, D9 e D10
2	1N4148	Diodo Retificador Rápido	D5 e D6
1	PHD560	Display 7 Segmentos Catodo Comum	DIG1
8	LED	Led Cilíndrico 2,5 mm Amarelo	LD0, LD1, LD2; LD3, LD4, LD5, LD6 e LD7
1	LED	Led Cilíndrico 2,5 mm Verde	LD8
1	LED	Led Cilíndrico 2,5 mm Vermelho	LD9
1	CI	Microprocessador PIC16F84	CI 1
1	PNP	Transistor BC558	T1
1	1000uF	Capacitor Eletrolíitico de **25V**	C1 (mínimo de 25 Volts)
1	10uF	Capacitor Eletrolítico de 16V	C5
1	220uF	Capacitor Eletrolítico de 16V	C2
1	7805	Regulador de tensão de 5V	CI 2
1	7812	Regulador de tensão de 12V	CI 3
12	100R	Resistor de 0,33W	R10, R11, R12, R13, R14, R28, R29, R30,R31
			R32, R38 e R39
10	10K	Resistor de 0,33W	R15, R16, R17, R18, R19, R33, R34, R35,R36
			e R37
17	1K	Resistor de 0,33W	R2, R3, R4, R5, R6, R7, R8, R9, R20, R21,
			R22, R23, R24, R25, R26, R27e R40
2	4K7	Resistor de 0,33W	R1 e R41
1	8 Vias	Chave "DIP"	S7
6	TACT	Chave "TACT"	S1, S2, S3, S4, S5, S6 e S7
1	10K	Trimpot Vertical 1 volta	TR1
1	4 MHz	Cristal Oscilador de 4 MHz	XTAL
1	JUMP	Jump	(para habilitar o Catodo Comum no "JMP1")
1	18 Pinos	Soquete para C. Integrado de 18 Pinos	Para o Microprocessador
1	T-220	Dissipador autoportante estanhado	Para o Regulador de Tensão de 5V



52 – DIAGRAMA DO CIRCUITO EXPERIMENTAL



53– FIGURA DO PCB DO CIRCUITO EXPERIMENTAL

54 – APÊNDICE I

54.1 - MÓDULO DE MATRIZ DE PONTO – MLCD

1- INTRODUÇÃO

Neste apêndice, referenciaremos apenas sobre o Módulo LCD, conhecidos por "DOT MATRIX", em particular, ao do tipo : "16x2" (16 caracteres x 2 linhas).

Estes módulos podem ser encontrados com LED backlight, que é uma iluminação de fundo, que facilita leitura em ambiente escuro. No seu verso encontra controladores que se comunicam por código do tipo ASCII.

2 – ROTEIRO PARA USO :

1) Ao energizar o módulo LCD, ajuste o trimpot de tal forma que possibilite a visualização da matriciação da primeira linha (para módulos de 2 linhas) ou a meia linha (para módulos de 1 linha).

2) O item acima deverá ser rigorosamente respeitado para que não ocorra a falta ou excesso de contraste, que possa impossibilitar a leitura do display.

3) Os módulos de apenas uma linha, também, deverão ser considerados eletricamente como se duas linhas, portanto para a FIXAÇÃO DAS CONDIÇÕES DE UTILIZAÇÃO, utilize o código 38H.

4) O sinal ENABLE (pino 6) deverá ser gerado conforme a descrição do "timing" adiante.

5) Para compatibilizar a velocidade do módulo com o CPU em que estiver conectado, existem duas possibilidades :

1- Intercalar uma rotina de atraso (delay) de aproximadamente 10 ms entre as instruções.
2- Fazer a leitura do "BUSY FLAG" após cada instrução.

6) Para utilizar um módulo LCD, **obrigatoriamente deveremos executar os 3 itens** de INICIALIZAÇÃO :

- FIXAÇÃO DAS CONDIÇÕES DE UTILIZAÇÃO
- CONTROLE ATIVO/INATIVO DO DISPLAY
- FIXAÇÃO DO MODO DE OPERAÇÃO

7) Para programar caracteres especiais, faça inicialmente, o ENDEREÇAMENTO DA CG RAM, cujo o endereço inicia em 40H

.

- Após executar o item acima, inicie a escrita da configuração do caracter especial, linha à linha, escrevendo esses dados na CG RAM.

- A incrementação do endereço da CGRAM será automática, se na INICIALIZAÇÃO assim o tiver configurado. Portanto, bastará escrever seqüencialmente as linhas correspondentes a todos os 8 caracteres especiais, não esquecendo também da linha do cursor.

8) Para escrever no display caracteres especiais, previamente programados, utilize como DADO, os códigos : 00H, como o primeiro caracter especial, 01H, como o segundo, e assim sucessivamente até o oitavo caracter especial.

3) MENU DE INICIALIZAÇÃO :

1) FIXAÇÃO DAS CONDIÇÕES DE UTILIZAÇÃO:

1 Linha 5x7 (8 bits) 30H
2 Linhas 5x7 (8 bits) 38H
1 Linha 5x10 (8 bits) 34H

1 Linha 5x7 (4 bits) 20H
2 Linhas 5x7 (4 bits) 28H
1 Linha 5x10 (4 bits) 24H

2) CONTROLE ATIVO/INATIVO DO DISPLAY:

Display aceso com cursor fixo 0EH
Display aceso com cursor intermitente 0FH
Display sem o cursor ..... CH
Display apagado 08H

3) FIXAÇÃO DO MODO DE OPERACÃO :

Escreve deslocando a mensagem à esquerda 07H
Escreve deslocando a mensagem à direita 05H
Escreve deslocando o cursor à direita 06H
Escreve deslocando o cursor à esquerda 04H

4) COMANDOS ÚTEIS :

Limpa o display e retorna o cursor 01H
Retorna o cursor sem alterar DD RAM 02H
Desloca somente o cursor à direita 14H
Desloca somente o cursor à esquerda 10H
Desloca o cursor + mensagem à direita 1CH
Desloca o cursor + a mensagem à esquerda 18H
Desloca o cursor para a segunda linha, na primeira posição C0H
Desloca o cursor para a primeira linha, na posição inicial 80H

5) ENDEREÇO DA CG RAM (CARACTER ESPECIAL) :

Para configurar o endereço inicial 40H
Para escrever primeiro caracter 00H
Para escrever último caracter 07H

OBS: Após o ENDEREÇAMENTO DA CG RAM, o cursor se desloca para a primeira posição da segunda linha (ou metade).

6 – EXEMPLO DE UTILIZAÇÃO DO MÓDULO LCD:

Os exemplos aqui apresentados são aplicáveis e referenciados no circuito exemplo exposto nesta publicação,

6.1 – GERANDO OS "DEFINE" E OS "EQU" :

```
#DEFINE     RS          PORTA,3          ; "RS" - REFERE-SE AO BIT3 DO PORTA
#DEFINE     RW          PORTA,1          ; "RW" - REFERE-SE AO BIT1 DO PORTA
#DEFINE     E           PORTA,2          ; "E"  - REFERE-SE AO BIT2 DO PORTA
#DEFINE     DATA_BUS    PORTB            ; "DATA_BUS  -  REFERE-SE AO PORTB

OFFSET:     EQU         12               ; "OFFSET"- RESERVA ESPAÇO NA RAM
```

Nos "DEFINE" do exemplo acima, o compilador respectivamente faz:

- PORTB : a substituição pelos 8 bits : B0 à B7
- PORTA : a substituição pelos 5 bits : A0 à A4
- RW : a substituição pelo bit : 1 do PORTA
- E : a substituição pelo bit 2 do PORTA
- RS : a substituição pelo bit 3 do PORTA

No "EQU" do exemplo acima, o compilador reserva o espaço de endereço 12 (0CH) na RAM batizando-o de "OFFSET" que será utilizado na programação como uma variável.

6.2 – GERANDO STATUS DE DADOS :

Para gerar os sinais de controle para a escrita de DADOS, siga a seqüência abaixo:

- Conhecido por "STATUS DE DADOS"

```
WR_DATA:                    ; STATUS DE ESCRITA (PORTA)

        BCF     RW          ; RESETA O BIT "RW" DO MLCD
        BSF     RS          ; SETA O BIT "RS" DO MLCD
        BSF     E           ; SETA O BIT "E" DO MLCD
        BCF     E           ; RESETA O BIT "E" DO MLCD

        RETURN
```

6.3 – GERANDO STATUS DE CONTROLE :

Para gerar os sinais de controle para a escrita de CONTROLE, siga a seqüência abaixo:

- Conhecido por “STATUS DE CONTROLE”

```
WR_CONTROL:             ; STATUS DE CONTROLE (PORTA)

        BCF     RW      ; RESETA O BIT "RW" DO MLCD
        BCF     RS      ; RESETA O BIT "RS" DO MLCD
        BSF     E       ; SETA O BIT "E" DO MLCD
        BCF     E       ; RESETA O BIT "E" DO MLCD

        RETURN
```

6.4 – VERIFICANDO O BUSY FLAG :

```
BUSY_CHECK:                     ; NOME DESTA SUBROTINA

        CLRF    PORTB           ; LIMPA O PORTB (NECESSÁRIO EM SIMULAÇÃO)
        MOVLW   B'10000000'     ; CARREGA W COM 80H
        BSF     STATUS,RP0      ; ACESSA O BANCO1 (TRISB)
        MOVWF   TRISB           ; HABILITA O BIT7 DO PORTB = ENTRADA
        BCF     STATUS,RP0      ; RETORNA PARA O BANCO0

        BCF     RS              ; RESETA O BIT "RS"
        BSF     RW              ; SETA O BIT "RW"

LOOP_C:

        BSF     E               ; SETA O BIT "E"
        BCF     E               ; RESETA O BIT "E"
        BTFSC   PORTB,7         ; VERIFICA O BUSY FLAG = 0,
        GOTO    LOOP_C          ; AGUARDA O BUSY FLAF (BIT7) ZERAR
        BCF     RW              ; SE BUSY FLAG=0, RESETA O  BIT "RW"

        BSF     STATUS,RP0      ; ACESSA O BANCO1 (TRISB)
        CLRF    TRISB           ; CONVERTE PORTB PARA SAÍDA
        BCF     STATUS,RP0      ; RETORNA PARA O BANCO0

        RETURN
```

7 – INICIALIZANDO UM MÓDULO (MLCD) :

Ao energizar o módulo LCD, é importante que a CPU faça a seqüência da operação abaixo :

```
LINHA:                          ; MODO  2 LINHAS (5x7)

        CALL    BUSY_CHECK      ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVLW   38H             ; ESPECIFICAÇÃO DE 2 LINHAS (5X7)
        MOVWF   DATA_BUS        ; ENVIA OS DADOS DO W PARA MLCD
        CALL    WR_CONTROL      ; GERA STATUS DE CONTROLE (PORTA)
```

```
DISPLAY:                        ; SEM CURSOR

        CALL    BUSY_CHECK  ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVLW   0CH         ; DISPLAY ACESO SEM CURSOR
        MOVWF   DATA_BUS    ; ENVIA O DADO PARA MLCD
        CALL    WR_CONTROL  ; GERA STATUS DE CONTROLE (PORTA0

OPERAÇÃO:                       ; ESCREVE DESLOCANDO À DIREITA

        CALL    BUSY_CHECK  ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVLW   06H         ; ESCREVE O CURSOR DESLOCANDO À DIREITA
        MOVWF   DATA_BUS    ; ENVIA DADOS PARA MLCD
        CALL    WR_CONTROL  ; GERA STATUS DE CONTROLE (PORTA)
```

7.1 – ESCREVENDO NO MÓDULO :

Estando o módulo já configurado os três itens anteriores, vamos agora, programá-lo para escrever a palavra : "FITO – Elo Digital" .
"+++++ 2006 +++++"

Portanto, iremos montar uma tabela de caracter.

```
TELA:

        MOVF    OFFSET,W    ; CARREGA EM W O OFFSET
        ADDWF   PCL,F       ; RETORNA COM O PROXIMO CARACTER EM W

        DT      "FITO-Elo DIGITAL"  ; PRIMEIRA LINHA
        DT      "+++++ 2006 +++++"  ; SEGUNDA LINHA
```

Vamos elaborar um PROGRAMA PRINCIPAL, que envia os caracteres da tabela (TELA), para escrever no display .
Assim :

```
INÍCIO: CLRF    OFFSET      ; ZERA A REFERÊNCIA DE POS. DOS CARACT.
        CALL    BUSY_CHECK  ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVLW   01H         ; DESLOCA O CURSOR 1o. POSIÇÃO/1o. LINHA E APAGA
        MOVWF   DATA_BUS    ; ENVIA DADO PARA CONTROLE DO MLCD
        CALL    WR_CONTROL  ; GERA STATUS DE CONTROLE (PORTA)

        CALL    DELAY       ; AGUARDA 500MS
        CALL    DELAY
        MOVLW   02          ; CARREGA W COM 02 (DECIMAL)
        MOVWF   LINE        ; CARREGA W EM LINE (2 linhas)
CONT2:
        MOVLW               ; CARREGA W COM 16 (DECIMAL)
        MOVWF   CARACTER    ; COPIA W EM CARACTER
REPETE:
        CALL    BUSY_CHECK  ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVF    OFFSET,W    ; CARREGA W COM OFFSET
        CALL    TELA        ; VAI ATÉ TELA E VOLTA C/ CARACT. EM W
        MOVWF   DATA_BUS    ; ENVIA O CARACTER (W) PARA MLCD
        CALL    WR_DATA     ; GERA ESTATUS DE ESCRITA PARA MLCD
        INCF    OFFSET,F    ; INCREMENTA O OFFSET (PRÓXIMO CARACTER)
        DECFSZ  CARACTER,F  ; DECREMENTA A CONTAGEM No. CARACTER
        GOTO    REPETE
        DECFSZ  LINE,F      ; DECREMENTA O LINE (MUDA A LINHA)
```

```
        GOTO        CONT1       ; SE NÃO ZEROU, CONTINUA EM CONT1
        GOTO        FINAL       ; SE TELA ESTÁ CHEIA, VAI PARA FINAL
CONT1:
        CALL        BUSY_CHECK  ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVLW       0C0H        ; DESLOCA O CURSOR 1o. POSIÇÃO/2o. LINHA
        MOVWF       DATA_BUS    ; ENVIA DADO PARA CONTROLE DO MLCD
        CALL        WR_CONTROL  ; GERA STATUS DE CONTROLE (PORTA)
        GOTO        CONT2       ; ESCREVE A SEGUNDA LINHA
```

NOTA :

- No exemplo acima, não se esquecer de iniciar o programa configurando o PORTA e o PORTB como saídas.

8 – EXEMPLO COMPLETO DE UM PROGRAMA PARA MLCD

```
;*************************************************************************************
;* O PROGRAMA TEM A FINALIDADE DE ENSAIAR DO TIPO MATRIZ DE PONTOS - MLCD
;* O HARDWARE ESTÁ EXPOSTO NA APOSTILA DO "PIC" COM A LEITURA DO BUSY FLAG ATRAVÉS DO
;* BIT 7 DO PORTB QUE É CONVERTIDO EM ENTRADA APLICÁVEL NO CIRCUITO EXPERIMENTAL
;*************************************************************************************

        LIST P=16F84
        Radix DEC
        INCLUDE <P16F84.INC>

        ORG 000H        ; INÍCIO DA GRAVAÇÃO NA EEPROM

;---------- definições de espaços na RAM e Bits ----------

#DEFINE     RS          PORTA,3     ; "RS" - REFERE-SE AO BIT3 DO PORTA
#DEFINE     RW          PORTA,1     ; "RW" - REFERE-SE AO BIT1 DO PORTA
#DEFINE     E           PORTA,2     ; "E"  - REFERE-SE AO BIT2 DO PORTA
#DEFINE      DATA_BUS   PORTB       ; "DATA_BUS" - REFERE-SE AO PORTB

TM1:        EQU         0CH         ; RESERVA A POSIÇÃO 0CH NA RAM
TM2:        EQU         0DH         ; RESERVA A POSIÇÃO ODH NA RAM
LINE:       EQU         0EH         ; RESERVA A POSIÇÃO 0EH NA RAM
CARACTER:   EQU         0FH         ; RESERVA A POSIÇÃO 0FH NA RAM
OFFSET:     EQU         11H         ; RESERVA A POSIÇÀO 11H NA RAM

; ---------- definição de entradas/saídas ----------

        MOVLW       B'00000001'     ; CARREGA W COM 01H (O BIT 0 DO PORTA = ENTRADA)
        BSF         STATUS,RP0      ; BANCO 1
        CLRF        TRISB           ; HABILITA o PORTB = SAÍDA
        MOVWF       TRISA           ; HABILITA OS BITS 1,2 E 3 DO PORTA = SAÍDA
        BCF         STATUS,RP0      ; BANCO 0

; ---------- subrotina para inicializar o MLCD ----------

LINHA:                              ; MODO  2 LINHAS : (5x7)
        CALL        BUSY_CHECK      ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVLW       38H             ; ESPECIFICAÇÃO DE 2 LINHAS (5X7)
        MOVWF       DATA_BUS        ; ENVIA OS DADOS DO W PARA MLCD
        CALL        WR_CONTROL      ; GERA STATUS DE CONTROLE (PORTA)

DISPLAY:                            ; DISPLAY SEM CURSOR
        CALL        BUSY_CHECK      ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVLW       0CH             ; DISPLAY ACESO SEM CURSOR
        MOVWF       DATA_BUS        ; ENVIA O DADO PARA MLCD
```

```
        CALL    WR_CONTROL  ; GERA STATUS DE CONTROLE (PORTA0

OPERAÇÃO:                   ; ESCREVE DESLOCANDO À DIREITA
        CALL    BUSY_CHECK  ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVLW   06H         ; ESCREVE O CURSOR DESLOCANDO À DIREITA
        MOVWF   DATA_BUS    ; ENVIA DADOS PARA MLCD
        CALL    WR_CONTROL  ; GERA STATUS DE CONTROLE (PORTA)

; ---------------------- PROGRAMA PRINCIPAL ---------------------------

INÍCIO: CLRF    OFFSET      ; ZERA A REFERÊNCIA DE POS. DOS CARACT.
        CALL    BUSY_CHECK  ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVLW   01H         ; DESLOCA O CURSOR 1o. POSIÇÃO/1o. LINHA E APAGA
        MOVWF   DATA_BUS    ; ENVIA DADO PARA CONTROLE DO MLCD
        CALL    WR_CONTROL  ; GERA STATUS DE CONTROLE (PORTA)

        CALL    DELAY       ; AGUARDA 500MS
        CALL    DELAY
        MOVLW   02          ; CARREGA W COM 02 (DECIMAL)
        MOVWF   LINE        ; CARREGA W EM LINE (2 linhas)

CONT2:  MOVLW   16          ; CARREGA W COM 16 (DECIMAL)
        MOVWF   CARACTER    ; COPIA W EM CARACTER

REPETE:
        CALL    BUSY_CHECK  ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVF    OFFSET,W    ; CARREGA W COM OFFSET
        CALL    TELA        ; VAI ATÉ TELA E VOLTA C/ CARACT. EM W
        MOVWF   DATA_BUS    ; ENVIA O CARACTER (W) PARA MLCD
        CALL    WR_DATA     ; GERA ESTATUS DE ESCRITA PARA MLCD
        INCF    OFFSET,F    ; INCREMENTA O OFFSET (PRÓXIMO CARACTER)
        DECFSZ  CARACTER,F  ; DECREMENTA A CONTAGEM No. CARACTER
        GOTO    REPETE
        DECFSZ  LINE,F      ; DECREMENTA O LINE (MUDA A LINHA)
        GOTO    CONT1       ; SE NÃO ZEROU, CONTINUA EM CONT1
        GOTO    FINAL       ; SE TELA ESTÁ CHEIA, VAI PARA FINAL

CONT1:  CALL    BUSY_CHECK  ; AGUARDA O MLCD (BUSY FLAG = 0)
        MOVLW   0C0H        ; DESLOCA O CURSOR 1o. POSIÇÃO/2o. LINHA
        MOVWF   DATA_BUS    ; ENVIA DADO PARA CONTROLE DO MLCD
        CALL    WR_CONTROL  ; GERA STATUS DE CONTROLE (PORTA)
        GOTO    CONT2       ; ESCREVE A SEGUNDA LINHA

FINAL:  CALL    DELAY       ; AGUARDA 500 MS
        CALL    DELAY
        GOTO    INÍCIO      ; REINICIA A ESCRITA NOVAMENTE

; ----------------- STATUS para sinais de DADOS ----------------------

WR_DATA:                    ; STATUS DE ESCRITA (PORTA)
        BCF     RW          ; RESETA O BIT "RW" DO MLCD
        BSF     RS          ; SETA O BIT "RS" DO MLCD
        BSF     E           ; SETA O BIT "E" DO MLCD
        BCF     E           ; RESETA O BIT "E" DO MLCD
        RETURN

; ----------------- STATUS para sinais de CONTROLE --------------------

WR_CONTROL:                 ; STATUS DE CONTROLE (PORTA)
        BCF     RW          ; RESETA O BIT "RW" DO MLCD
        BCF     RS          ; RESETA O BIT "RS" DO MLCD
        BSF     E           ; SETA O BIT "E" DO MLCD
        BCF     E           ; RESETA O BIT "E" DO MLCD
        RETURN
; ----------------- subrotina para checar BUSY FLAG ----------------------

BUSY_CHECK:                 ; NOME DESTA SUBROTINA É "Busy_Check"
```

```
        CLRF    PORTB       ; LIMPA O PORTB (NECESSÁRIO EM SIMULAÇÃO)
        MOVLW   B'10000000' ; CARREGA W COM 80H
        BSF     STATUS,RP0  ; ACESSA O BANCO1 (TRISB)
        MOVWF   TRISB       ; HABILITA O BIT7 DO PORTB = ENTRADA
        BCF     STATUS,RP0  ; RETORNA PARA O BANCO0

        BCF     RS          ; RESETA O BIT "RS"
        BSF     RW          ; SETA O BIT "RW"
LOOPC:  BSF     E           ; SETA O BIT "E"
        BCF     E           ; RESETA O BIT "E"
        BTFSC   PORTB,7     ; VERIFICA SE O BUSY FLAG (BIT 7) = 0
        GOTO    LOOPC       ; SE NÃO, AGUARDA O BUSY FLAG (BIT7) ZERAR
        BCF     RW          ; SE BUSY FLAG=0, RESETA O  BIT "RW"

        BSF     STATUS,RP0  ; ACESSA O BANCO1 (TRISB)
        CLRF    TRISB       ; CONVERTE PORTB PARA SAÍDA
        BCF     STATUS,RP0  ; RETORNA PARA O BANCO0
        RETURN

;---------------------- subrotina delay 250 ms -------------------------

DELAY:                      ; NOME DESTA SUBROTINA É "Delay" (ATRASO)

        MOVLW    251        ; CARREGA W COM O VALOR 251
        MOVWF   TM1         ; COPIA O W NA VARIÁVEL TM1

LOOPA:  MOVLW   248         ; CARREGA W COM O VALOR 248
        MOVWF   TM2         ; COPIA O W NA VARIÁVEL TM2

LOOPB:  NOP                 ; PERDE UM CICLO DE MÁQUINA
        DECFSZ  TM2,F       ; DECREMENTA TM2, SE Z=1, PULA UMA LINHA
        GOTO    LOOPB       ; CONTINUA A DECREMENTAR TM2 EM LOOP2

        DECFSZ  TM1,f       ; DECREMENTA TM1, SE Z=1, PULA UMA LINHA
        GOTO    LOOPA       ; RETORNA PARA LOOP1
        RETURN

;----- Nesta subrotina contém as frases para serem escrita no MLCD --------

TELA:
        MOVF    OFFSET,W    ; CARREGA EM W O OFFSET
        ADDWF   PCL,F       ; RETORNA COM O PROXIMO CARACTER EM W

        DT      "FITO-Elo DIGITAL"      ; PRIMEIRA LINHA
        DT      "+++++ 2006 +++++"      ; SEGUNDA LINHA

        END                 ; FIM DESTA PROGRAMAÇÃO
```

9– ENDEREÇOS DOS CARACTERES NA DDRAM :

Corresponde ao lugar físico do caracter no display.

LINHA 1	**80**	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**8A**	**8B**	**8C**	**8D**	**8E**	**8F**	90	91	92	93
LINHA 2	**C0**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**CA**	**CB**	**CC**	**CD**	**CE**	**CF**	D0	D1	D2	D3
LINHA 3	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93
LINHA 4	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1	D2	D3
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

10 – ENDEREÇOS DOS CARACTERES ESPECIAIS :

Os caracteres especiais previamente programado, durante a inicialização, podem ser utilizados a qualquer tempo como se fossem caracteres normais, lembrando que cada caracter ocupa 8 endereços.
Assim, o primeiro caracter corresponde ao endereço 40H, e os demais, 48H, 50H, 58H, 60H, 68H, 70H e 78H respectivamente.

Para escrever no display um caracter especial, previamente configurado, basta substituí-lo na forma hexadecimal, na seguinte ordem:

00H - para o primeiro caracter especial
01H - para o segundo caracter especial
02H - para o terceiro caracter especial
03H – para o quarto caracter especial

11 – CONFIGURANDO CARACTERES ESPECIAIS :

```
CONFIG_C_ESP:

        CALL    BUSY_CHECK          ; AGUARDA BUSY
        MOVLW   40H                 ; CARREGA W COM 40H
        MOVWF   DATA_BUS            ; ESCREVE NA VIA DE DADOS DO MLCD
        CALL    WR_CONTROL          ; STATUS DE ESCRITA DE CONTROLE
        CLRF    DATA_BUS            ; LIMPA BUS DE CONTROLE
WR_CE:
        CALL    BUSY_CHECK          ; AGUARDA O BUSY
        MOVF    OFFSET,W            ; CARREGA O VALOR DE W COM O OFFSET
        CALL    C_ESPECIAIS         ; VAI PARA TABELA C. ESP. E VOLTA C/ W CARREGADO
        MOVWF   DATA_BUS            ; O VALOR OBTIDO EM W É COL0CADO DATA_BUS
        CALL    WR_DATA             ; STATUS DE ESCRITA DE DADOS
        CLRF    DATA_BUS            ; LIMPA BUS DE CONTROLE
        INCF    OFFSET,F            ; INCREMENTA OFFSET
        DECFSZ  NUM_C_ESP,F         ; DECREMENTA No. CARC.ESP., SE=0, PULA
        GOTO    WR_CE               ; COTINUA À EXECUTAR
        CLRF    OFFSET              ; LIMPA OFFSET


C_ESPECIAIS:

        ADDWF   PCL,F               ; RETORNA COM A INFORMAÇÃO EM W


;00H ------------------------------------------------------------------------
        ; "ç" - C com cedilha - minúsculo
        DT      000H,000H,00EH,010H,010H,014H,00EH,010H
;01H ------------------------------------------------------------------------
        ; "ã" - A com til - minúsculo
        DT      00EH,000H,00EH,001H,00FH,011H,00FH,000H
;02H ------------------------------------------------------------------------
        ; "ô" - O com acento circunflexo - minúsculo
        DT      04CH,00AH,011H,00EH,011H,011H,011H,00EH
;03H ------------------------------------------------------------------------
        ; "á" - A com acento agudo - minúsculo
        DT      002H,004H,000H,00EH,001H,00FH,011H,00FH
```

12 – UTILIZANDO CARACTERES ESPECIAIS :

Numa tabela de caracteres, as frases que utilizam caracteres especiais, previamente configurados, ficam assim:

- A palavra : “Eletr**ô**nica” e “f**á**cil”

```
DT      “Eletr”,02H, “nica”    ; O código 02H substitui o caracter “ô”

DT      “f”,03H“cil”           ; O código 03H substitui o caracter “á”
```

13 – TABELA DE CÓDIGOS ASCII

TABELA DE CÓDIGO ASCII

Valores em Hexa		Dígitos mais signifcativo							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Dígitos menos significativo	**0**			ESPAÇO	0	@	P	`	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	BELL	ETB	'	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	]
	D	CR	GS	-	=	M	]	m	}
	E	SQ	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

Por exemplo, se deseja escrever “FITO”, fica :

```
DT      46H,49H,54H,4FH        ; “FITO”
```

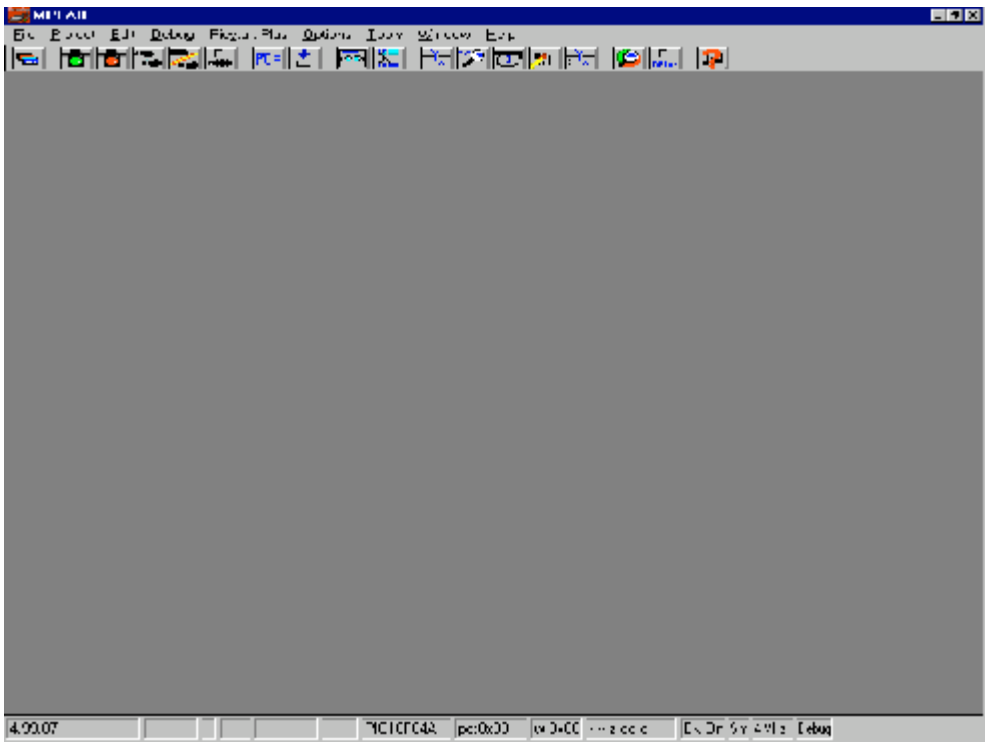
14 – PINAGEM DAS CONEXÕES PARA INTERFACE DO MLCD

No.	*SÍMBOLO*	*NÍVEL*	*FUNÇÃ0*
0	LED (-)	-	LED (0V)
1	Vss	-	GND (0V)
2	VDD	-	VCC (+5V + - 5%)
3	Vo	-	Ajuste de contraste
4	RS	H/L	Sinal do Seletor de Registro
5	R/W	H/L	Seleção de Leitura/Escrita
6	E	H,H->L	Sinal de Habilitação
7	DB0	H/L	DATA BIT 0
8	DB1	H/L	DATA BIT 1
9	DB2	H/L	DATA BIT 2
10	DB3	H/L	DATA BIT 3
11	DB4	H/L	DATA BIT 4
12	DB5	H/L	DATA BIT 5
13	DB6	H/L	DATA BIT 6
14	DB7	H/L	DATA BIT 7

– APÊNDICE II - SOBRE O USO DO MPLAB

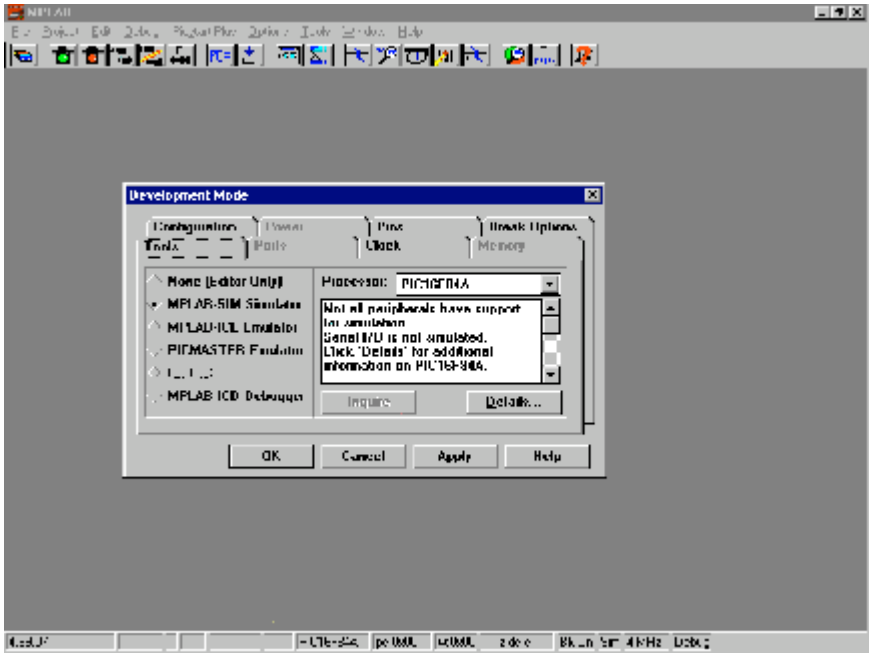
1- UTILIZANDO O MPLAB (PRIMEIRA VEZ)

Ao “abrir” o programa MPLAB pela primeira vez, a sua tela ficará parecida como esta:



Se o processador indicado no rodapé não for **16F84** (ou 16F84A), prossiga a seguinte seqüência:

Clique o menu **Options** e depois **Development Mode** , então teremos o seguinte quadro:



Marque a opção **MPLAB-SIM Simulator** e selecione **PIC16F84** (ou 16F84A) no menu à direita, deixando sua seleção conforme visto acima, e depois clique no botão **Reset**.
A sua tela deverá ficar como a tela inicial (primeira)

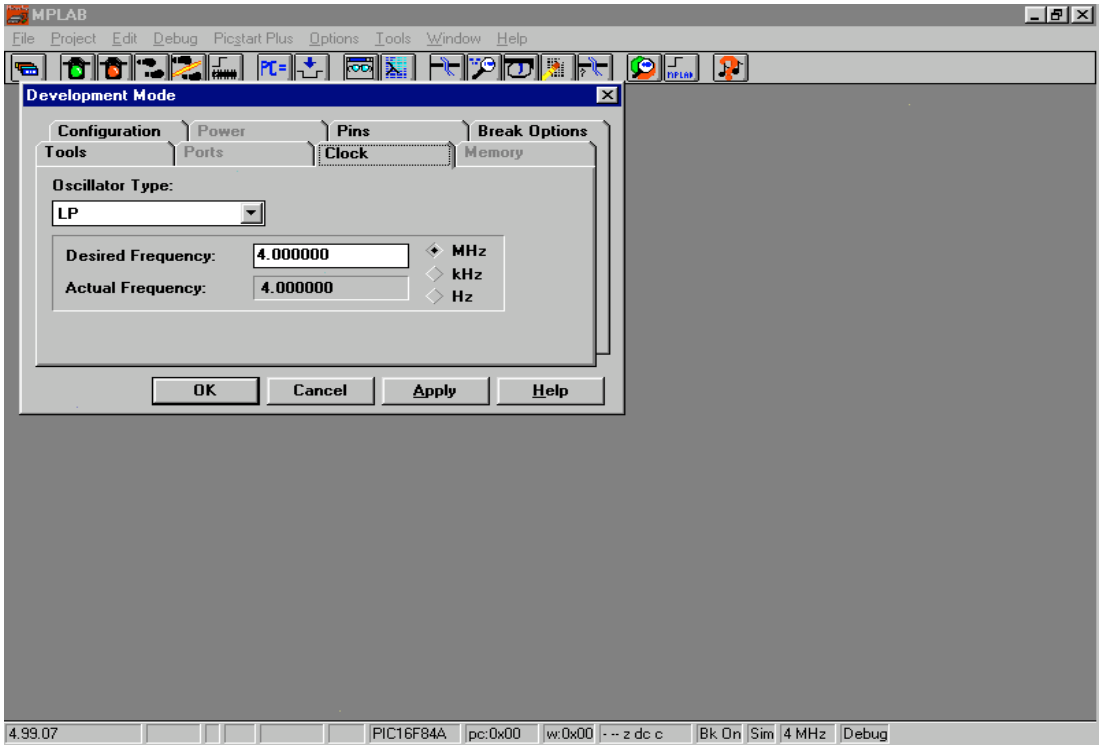
2 - AJUSTANDO A FREQÜÊNCIA (Clock) PARA SIMULAÇÃO:

Para realizarmos simulações no próprio compilador MPBLAB, deveremos ajustar a freqüência de clock com que o microcontrolador irá funcionar

Selecione o menu do MPLAB :

Options --- > **Processador Setup** ----> **Clock Frequêncy**

e obteremos a próxima figura :



No campo "**Desired Frequência**" digite a freqüência desejada e selecione uma das opções : **MHz**; **KHz** ou **Hz**

No campo "Actual Frequência" mostra a freqüência atual com que o simulador está realizando os eventos.

Antes de clicar em "Close", dê um clique em "Set Clock" para que o programa ajuste o valor internamente.

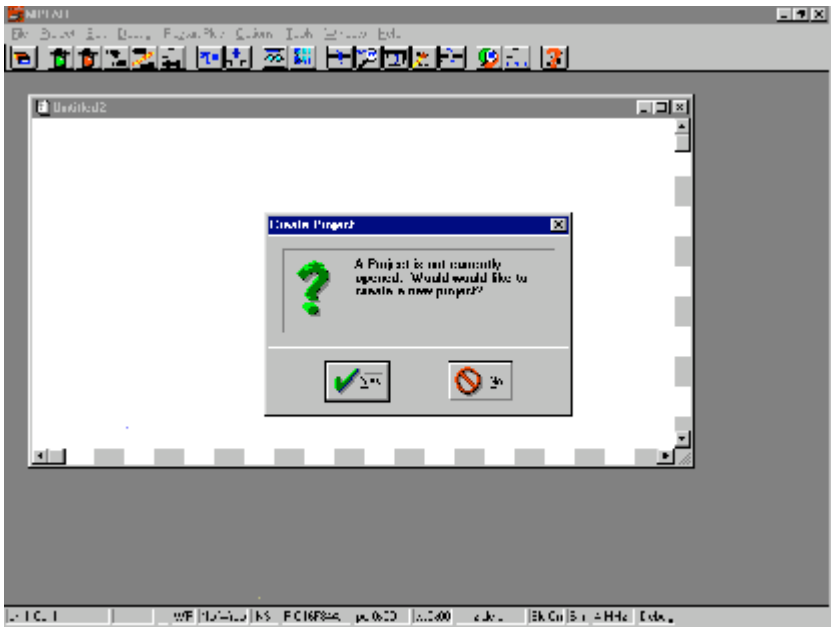
3 - TRABALHANDO COM "PROJETOS"

3.1 - CRIANDO UMA NOVA FONTE PARA TRABALHARMOS COM UM PROJETO.

Para iniciarmos um novo projeto (***New Project***), será necessário criar antes uma nova fonte (***New File*** que tem a extensão **.ASM**).
Para isso, siga a seguinte seqüência:

Selecione : File ---> **New**

Uma nova fonte de nome **Untile** (bloco vazio) surgirá, e o MPLAB lhe perguntará se deseja criar um novo projeto.



Responda NO (não) para prosseguir na criação da nova fonte

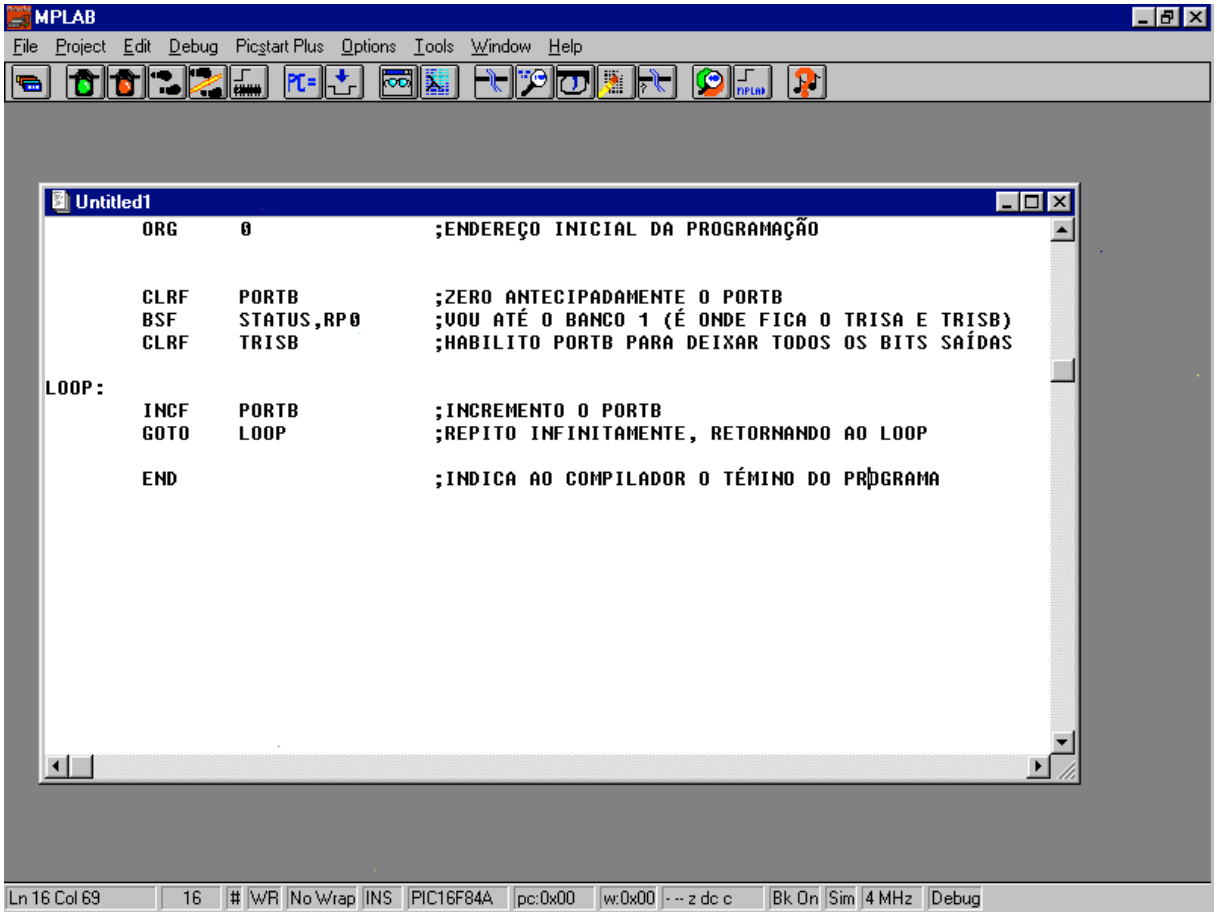
Nesta janela digite o texto conforme abaixo, e salve-o com o nome "**teste_1. asm**" .

de que modo ? Após digitar todo o programa, selecione :

File ----> Save as ...

Não esquecer de preencher na janela aberta um nome para a fonte no **drive A** (do disquete).

Então, dê um clique no botão **OK**, e a sua tela deverá estar conforme a próxima figura .

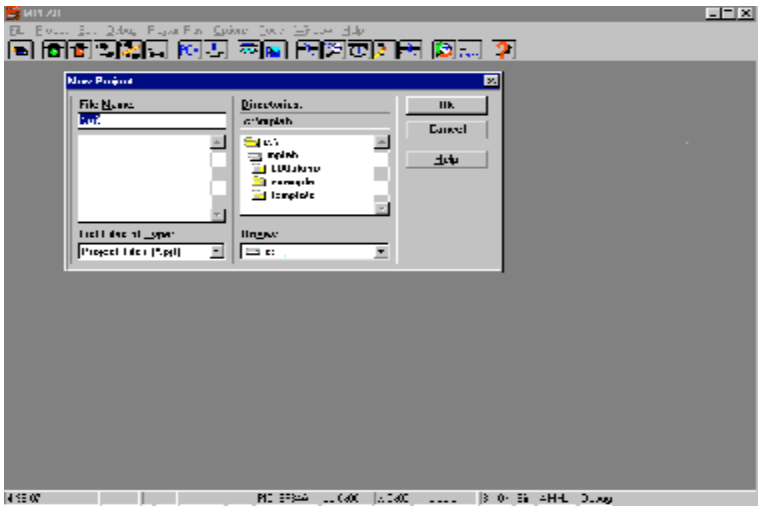


4 - CRIANDO UM PROJETO:

Após criarmos a **Fonte**, vamos criar o **Projeto**, pois é com o Projeto que iremos desenvolver o programa.

Selecione **Project** -----> **New Project**

A seguinte janela se abrirá :



Na janela "**File Name**" , digite o nome do projeto com extensão "**pjt**"

Não esquecer de selecionar o **Drive**, dê preferência para **A** (disquete)

IMPORTANTE *: A fonte e o projeto deverão estar no mesmo diretório !!!!*

Sempre que um ***novo projeto*** é criado, a janela "**Edit Project**" será exibida.

Obs: O nome da fonte não precisa ser o mesmo do projeto.

5 - EDITANDO O PROJETO (Novo ou Existente)

Entende-se por "***editar o projeto***" o processo de ***compilar*** ajustando os parâmetros em simulações.

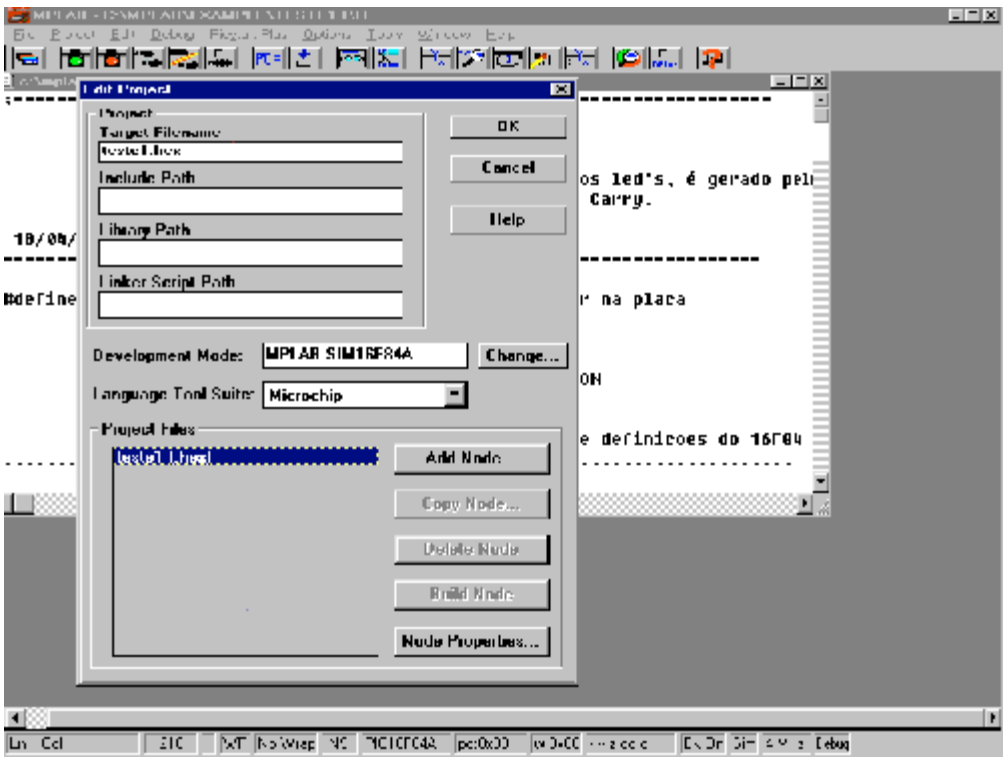
Quando o projeto é novo, esta opção aparecerá automaticamente.

Podemos utilizar ainda a edição do projeto para trocar a fonte usado no desenvolvimento.

Para indicarmos a fonte de um projeto, ou caso queira alterar os parâmetros do compilador, deveremos executar a seguinte seqüência:

Project ----------> Edit Project

Para o nosso exemplo, teremos a tela abaixo demonstrada.



Note que em nosso exemplo, o campo "**Taget filename**" indica o alvo , isto é, o objetivo final é a geração do arquivo **teste_1.hex**.

Ajuste o campo "**Development mode**" para exibir **MpLab-SIM, 16F84**.

Ajuste o campo "**Language tool suite**" para **Microchip**.

No campo "**Project file**"dê um click em cima do texto "**teste_1 [.hex**]" e observe que os botões **Add Node** e **Node Properties** ficarão realçados (caso ainda não estejam).

Sua janela deverá estar como a figura acima, caso não esteja refaça todos os ajustes.

6 - AJUSTANDO AS PROPRIEDADES DO "NÓ" (NODE):

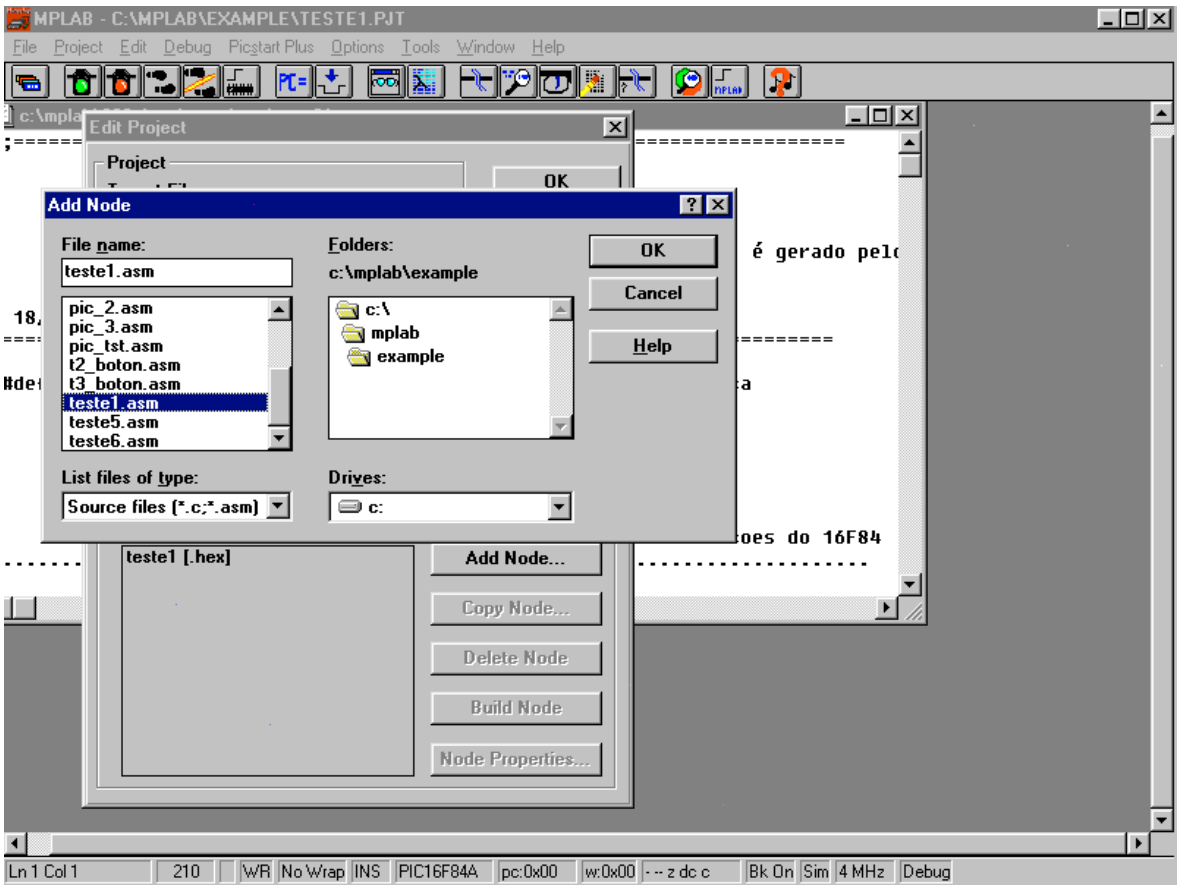
Dê um duplo clique no botão "**Node Properties**". surgirá a janela abaixo, com algumas opções pré-selecionadas, que deverão ser refeitas.

Description			
Define	☐ On		
Hex Format	☑ INHX8M	☐ INHX8S	☐ INHX32
Error File	☑ On	☐ Off	
List File	☑ On	☐ Off	
Cross-reference File	☐ On	☑ Off	
Warning level	☑ all	☐ warn+err	☐ err
Case sensitivity	☐ On	☑ Off	
Macro expansion	☐ On	☐ Off	
Default radix	☐ HEX	☐ DEC	☐ OCT
Tab size	☐ On		

Se as opções marcadas estiverem diferentes, com o auxílio do mouse, ajuste os campos de acordo com a figura anterior, e dê um clique no **OK** .
A sua tela retomará a da figura do "Edit Project".
Agora, podemos informar ao MpLab qual o compilador que deveremos utilizar, quais os arquivos para gerar, listar erros, ...etc, formatos HEX, entre outros.
Portanto, para cada projeto novo, você deverá ajustar desta forma;

7 - AJUSTANDO UMA FONTE (CHAMADO TAMBÉM DE NÓ):

Vamos informar agora ao compilador qual fonte será utilizada, dando um clique no botão **Add Node**, e a tela ficará assim :

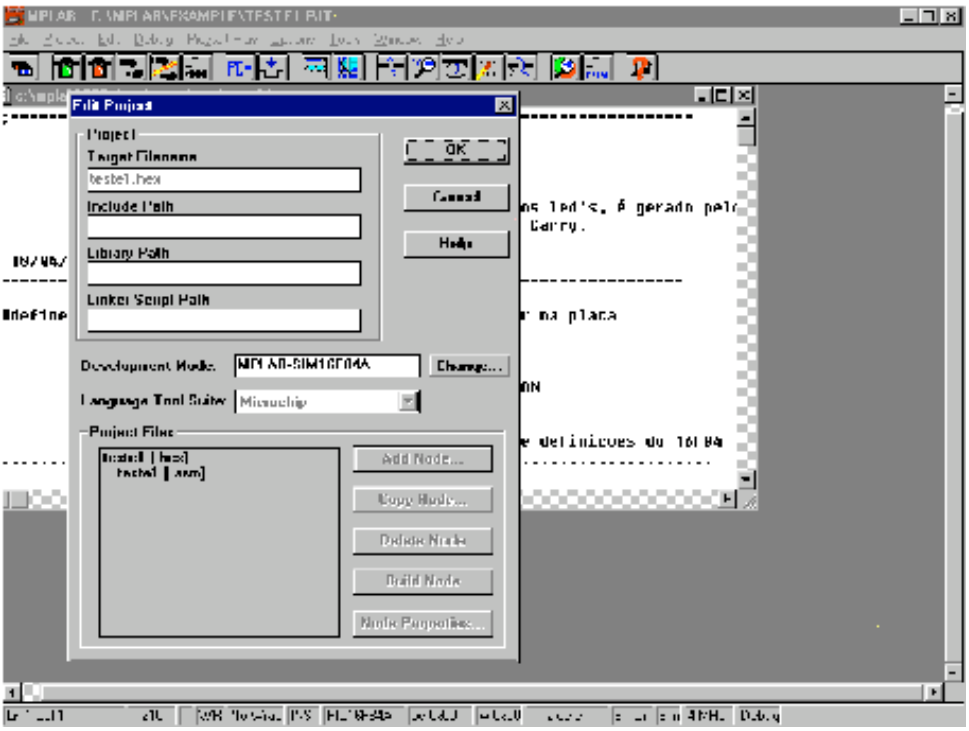


Nesta janela vamos escolher qual o diretório e dentro do mesmo, qual a fonte fará parte desse nosso projeto.

Observe que a janela já aparecerá com o diretório atual do projeto e como está demonstrada em nosso exemplo, a fonte criada anteriormente, **teste1.asm**.

Selecione a fonte e dê um clique no **OK**.

A sua janela "**Edit Project**" deverá estar conforme a próxima figura :



Agora, basta dar um clique no botão **OK**, que o processo de edição do projeto estará completo.

A partir deste ponto, poderemos compilar e simular este programa

OBSERVAÇÕES IMPORTANTES:

Evite que um "travamento"de programa obrigue-o à rescrever tudo que digitou, procure sempre salvar da seguinte forma:

- **Selecione Project ------- > Save Project**
- **Selecione File ---------- > Save All**

8 - COMPILANDO O PROGRAMA :

Para compilar a fonte, basta teclar: **F10**

Se não houver erros de sintaxe ou outros, haverá a indicação de tudo **OK**, pela janela Build Results, mensagens " **Build completed successfully**", conforme a figura abaixo:

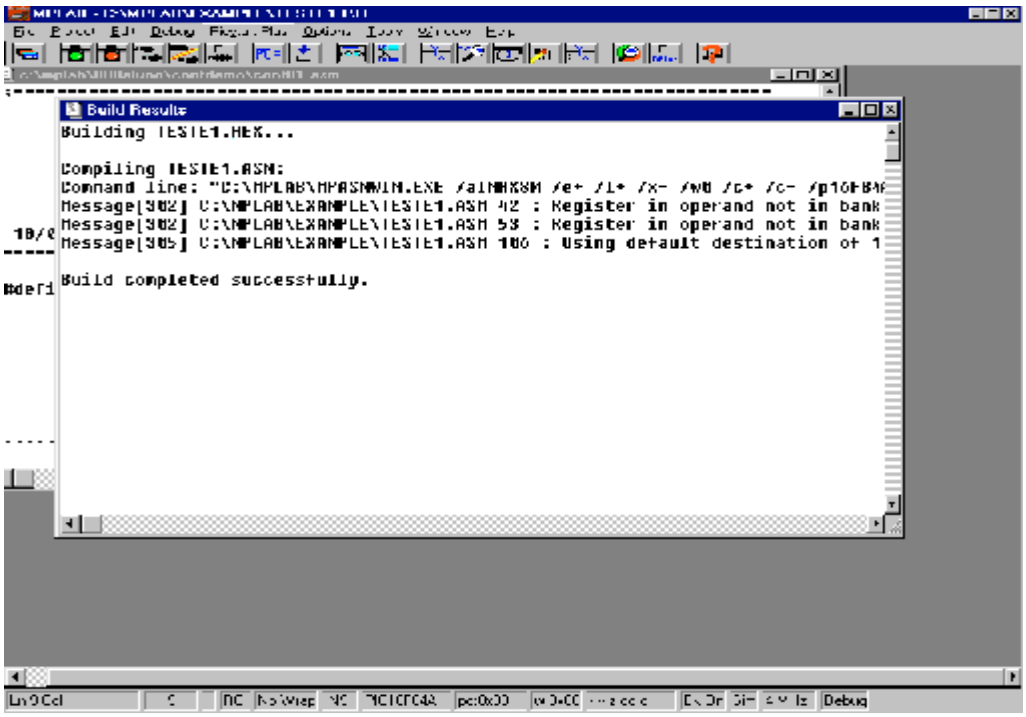
Feche a janela Build Results.

Importante: O compilador não é capaz de detectar erros de construção lógica, portanto, o fato de não constarem erros ou mensagens "**Warning**", não significa que tudo está OK !

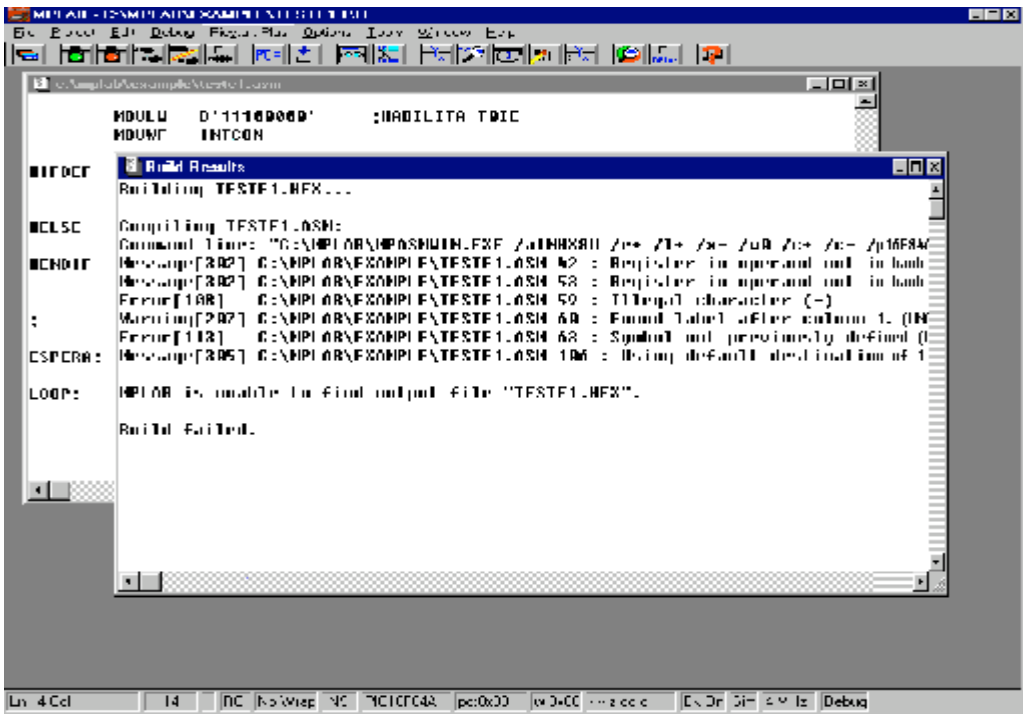
Se houver erros, uma janela indicará todos os erros e "**Warning**" e durante o processo, a barra de evolução, mudará da cor **verde** para **vermelha**.

Onde está o erro ou aviso de "Warning" que o compilador acusou ?

Basta com o auxílio do mouse, clicar sobre o aviso de erro ou "warning", e automaticamente, o cursor indicará a linha incorreta.



Abaixo, um exemplo de uma tela acusando erros e 'Warning".



9 - TRABALHANDO COM PROJETOS JÁ EXISTENTES :

A principal diferença que encontramos quando abrimos um projeto já existente, está no fato de não precisarmos selecionar novamente a fonte ou ajustar o compilador. Bastará selecionar nos diretórios existentes o projeto desejado.

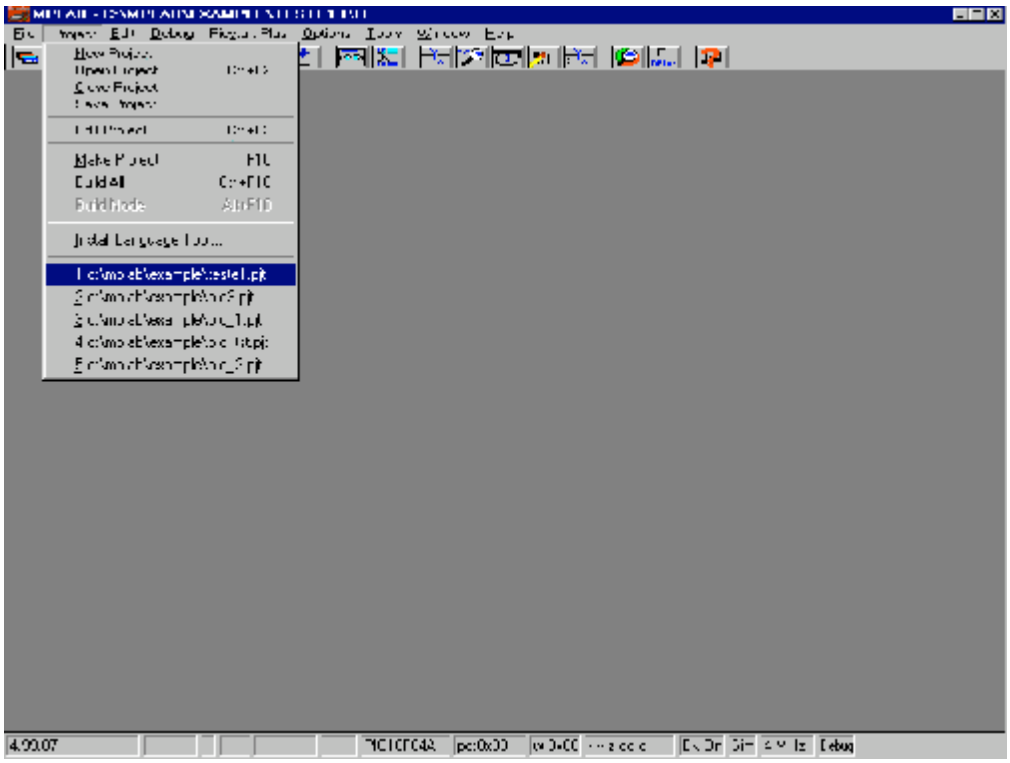
Para melhor entendimento, feche o MPLAB e volte para Windows, respondendo "**YES**" para todas as perguntas, e reinicie o MPLAB.

O MPLAB irá perguntar se você deseja abrir o ultimo projeto em uso. Se desejar, clique em "**YES**". No nosso caso, clique "**NO**" (não).

Selecione então o menu **Project**

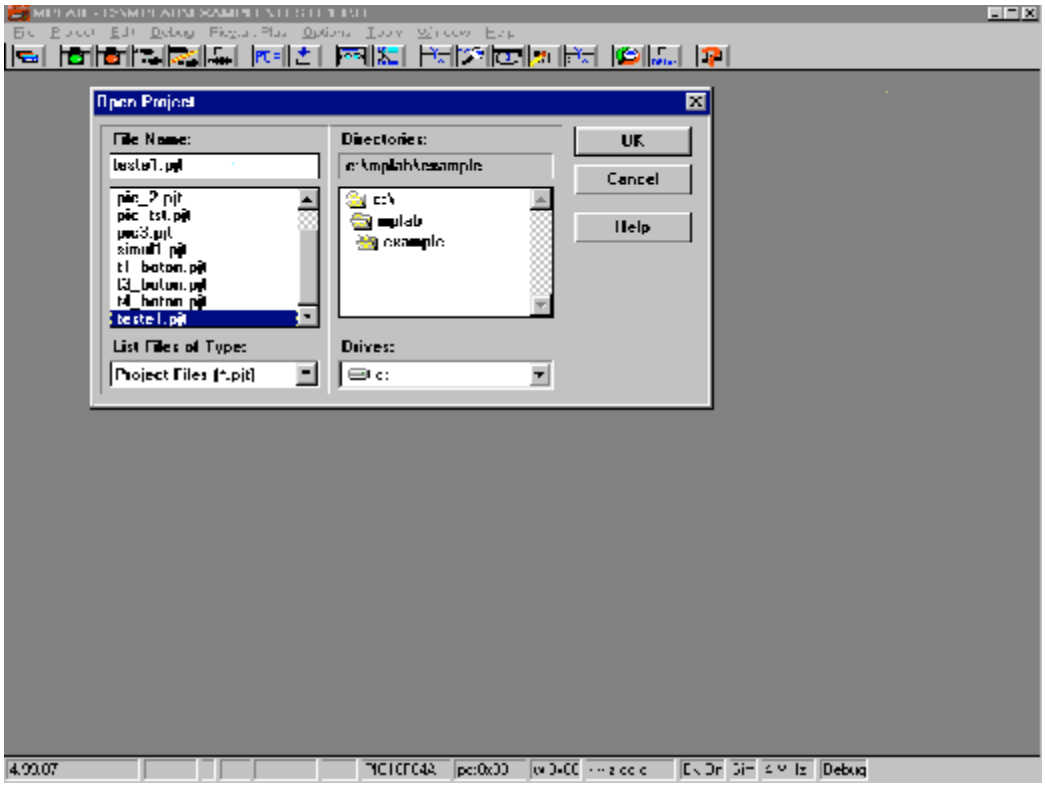
Veja que no rodapé do menu que se abriu temos o nome dos últimos projetos em que trabalhamos.

Caso o projeto indicado seja o interessado, basta clicar sobre o mesmo.
Veja como fica :



Note que estão "abertos" os últimos projetos, basta clicar sobre o interessado.

Se o projeto desejado não estiver indicado, dê um clique no item **Open Project**, e teremos uma nova janela conforme veremos na próxima figura .



Escolha o diretório onde está o projeto desejado e depois o nome do mesmo, em seguida dê um clique em **OK** para "abrir" o projeto.

Caso apareça uma janela com os dizeres "***i No hex file has been buit for this project***", apenas clique **OK** e prossiga normalmente.

O MPLAB está apenas informando que a sua fonte necessita ser compilada, pois não achou o hexadecimal da mesma. Isto pode ocorrer por termos fechado o projeto, antes de termos corrigido todos os erros, por exemplo.

10 - SIMULAÇÃO SIMPLES:

O MPLAB permite técnicas para simular o nosso programa, desde a execução passo a passo, até a animação.

As principais teclas para o controle de simulação:

F6 (Reset) :
Equivale ao reset da CPU, posiciona o contador de programa no endereço 000H, e coloca uma barra preta sobre a linha correspondente.
Esta barra indica " a próxima " instrução a ser simulada.

F7 (Step) :

A cada clique em F7, o MPLAB executa uma instrução do programa. Isso é muito útil em execuções do tipo passo a passo. Porém, se continuarmos a pressionar, a simulação se realizará de forma contínua.

CTRL + F9 :

Roda o programa dinamicamente, de tal modo que permite visualizar a seqüência do programa de forma animada, com a atualização constante dos valores dos registros.

F5 (Stop) :

Interrompe a simulação dinâmica, iniciada pelo Ctrl+F9.

F9 (Run) :

Realiza a simulação rápida, sem atualizar a tela (sem animação). É ideal para simular situações que tornam necessária a agilização do processo de simulação.

Apenas a janela do Stop Watch (cronômetro) é atualizada.

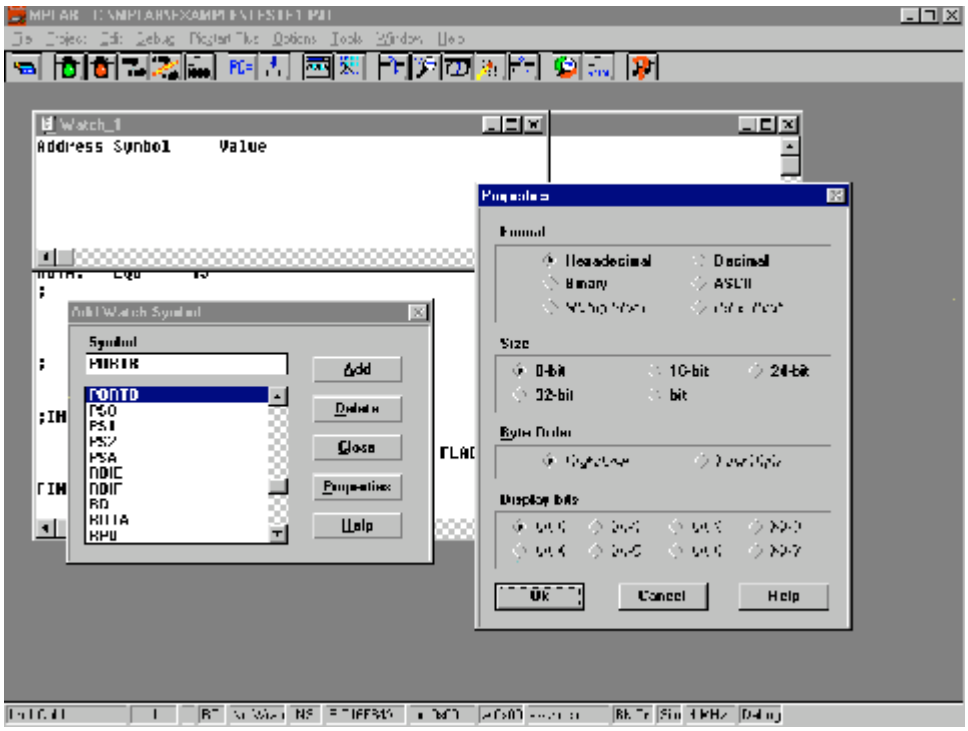
11 - OBSERVANDO REGISTROS DA CPU DURANTE A SIMULAÇÃO:

Além de podermos ver o tempo de execução, podemos ainda ver como os registros se comportam durante a execução do programa.

No menu, selecione a seqüência :

Window ---------- > New watch window

e a sua janela ficará assim :



Na janela ativa (**Add Watch Symbol**), no campo "**Symbol**": escreva **PORTB** e depois clique no botão **Properties**.

Sua tela deverá estar como a figura anterior, com uma janela onde se ajusta as propriedades do registro selecionado (no nosso caso**, PORTB** e a propriedade é **Decimal**).

Depois de um clique no **OK** , volta para a janela anterior, clique em **Add** e depois no **Close** da janela "**Add Watch Symbol**" e veja que a janela **Watch_1** contem agora os valores do **PORTB**.

Por exemplo :	**Adress**	**Symbol**	**Value**
	06	**PORTB**	**D'154'**

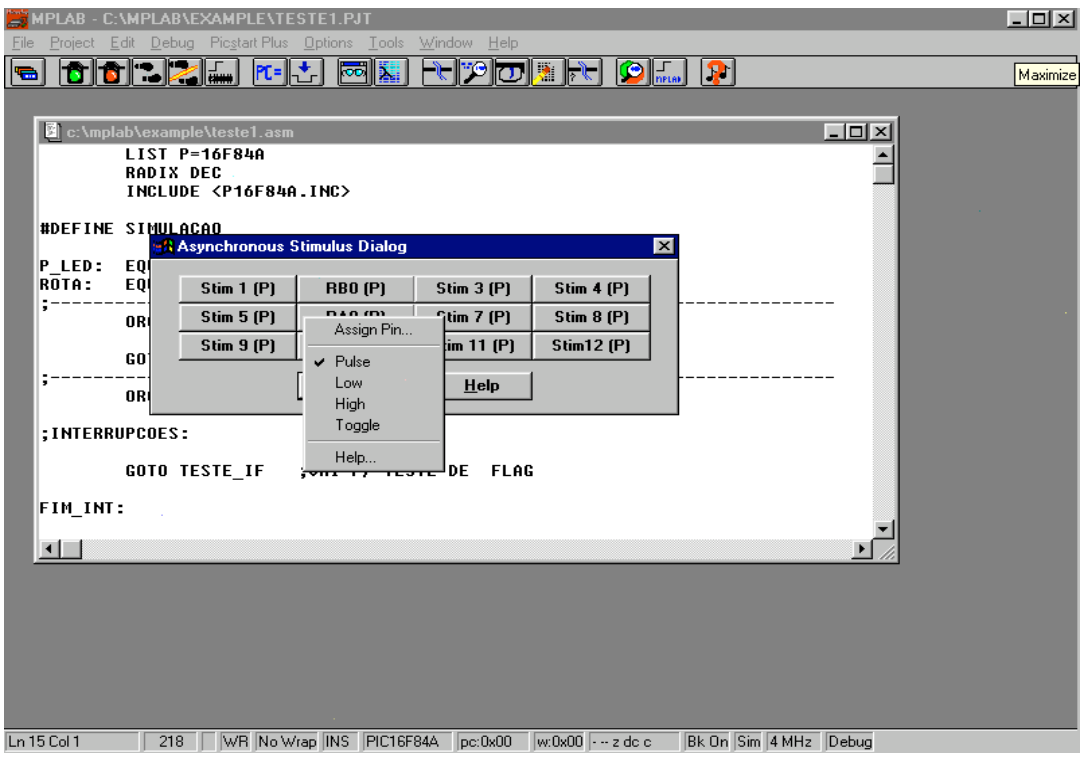
Para anexar mais registros, basta ativar com o mouse a barra da janela do **Watch_1** e pressionar a tecla "**Ins**"(inserir), e repetir os mesmos processos anteriores.

Para deletar registros inúteis, posicione o cursor sobre o registro, e depois pressione a tecla "**Del**'. (deletar).

12 - SIMULANDO SINAIS EXTERNOS :

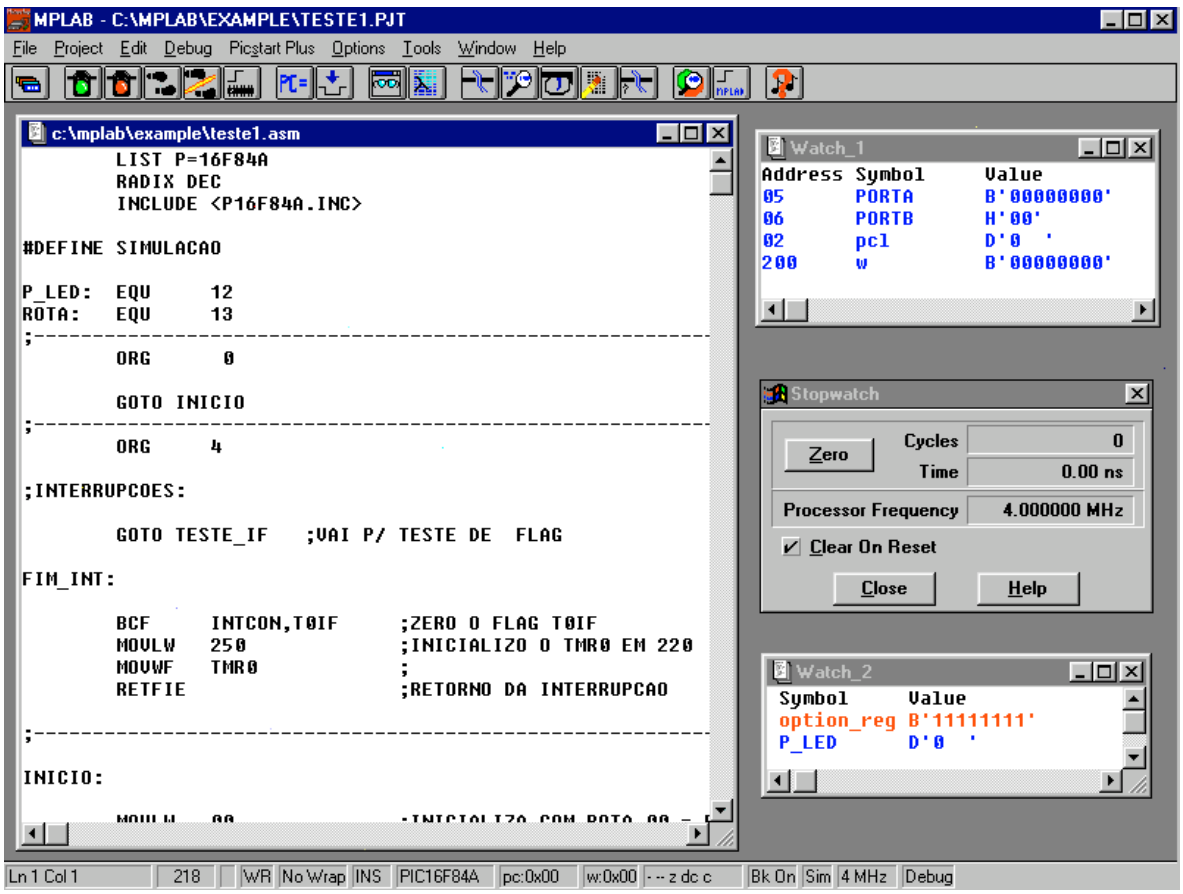
O MPLAB permite simular variações externas nos pinos do PIC.
Na tela inicial, selecione:

Debug --------> Simulator ----------> Stimulus --------- > Asynchronous Stimulus



Observe que existem 12 “botões” com nomes **Stim 1** à **Stim 12**, e dentro do parenteses a letra “ **P** ” .
Com o botão direito do mouse dê um clique no “botão” **Stim 1 (P)** e você obterá o seguinte opções do menu “**Assign Pin** ...”:

Pulse - (P) Da um pulso no pino selecionado (de 0 p/ 1 e retorna)
Low - (L) Coloca um nível “0” no pino
High - (H) Coloca um nível “1”
Toggle – (T) Inverte o estado no pino (inicia em “0”)



Exemplo de uma distribuição (lay-out) de janelas para executar uma simulação

13 - USANDO PONTOS DE PARADAS (BREAK POINTS) :

Durante uma simulação (**CTRL + F9**) ou **F9**, podemos prever uma interrupção do processo em andamento, sinalizando com “Break points”.

A técnica consiste em levar o cursor até a linha do programa, e clicar com o lado direito do mouse.
Uma pequena janela igual a próxima figura surgirá :

Selecione a opção " Break Point(s) " , e depois clique com o lado esquerdo.
A linha do programa que for selecionada, deverá mudar para cor vermelha.

Veja na figura abaixo, a linha "BCF INTCON,T0IF" está demarcada com um "Break Point ".

Ao "rodar " este programa numa simulação, fatalmente irá sofrer uma interrupção nesta linha demarcada.

```
        GOTO TESTE_IF    ;VAI P/ TESTE DE  FLAG

FIM_INT:

        BCF     INTCON,T0IF     ;ZERO O FLAG T0IF
        MOVLW   250             ;INICIALIZO O TMR0 EM 220
        MOVWF   TMR0            ;
        RETFIE                  ;RETORNO DA INTERRUPCAO
```

14 - RESUMO :

- Para iniciarmos um desenvolvimento de software, selecionamos inicialmente '***File***" na tela inicial do MPLAB. Em seguida ***:*** " ***New*** "
Após estas seleções, surgirá uma janela denominada " **Untiled1** "

Será nesta janela vazia (untiled1) que iniciaremos a digitação da **Fonte** (***File***)

- A **Fonte** (***File***) possui a extensão " **ASM** " . Sempre iniciamos a elaboração do programa através da Fonte que será salva com extensão "***.asm***"

Assim:

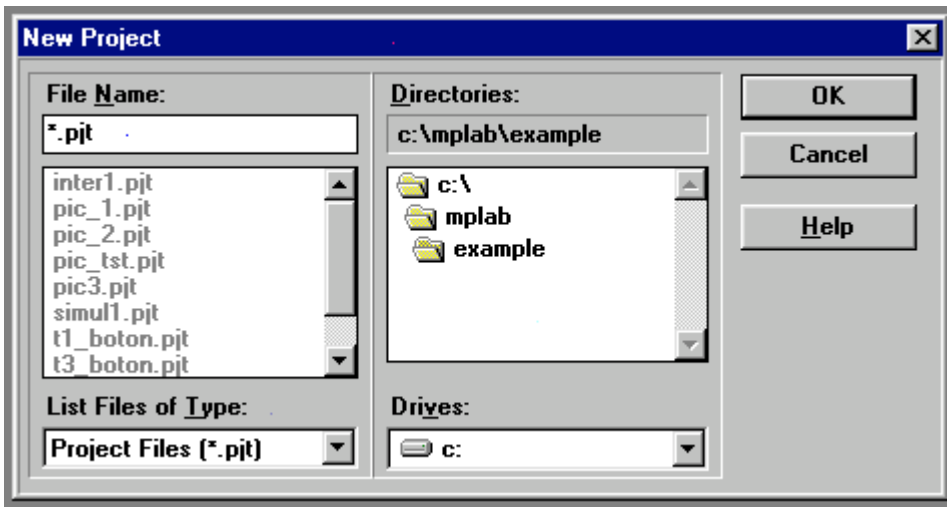
Selecione : **File** ------- > **Save as ....**

Veja a figura abaixo de uma janela ' **Untiled1**"

- O **Projeto** (***Project***) possui a extensão " **PJT** " .
Salve um projeto selecionando :

Project ---------- > **New Project**

E, na tela seguinte digite o nome no lugar do asterísco (**File Name :**), o mesmo que da **Fonte**, porém, com extensão "**pjt**".



15 - PARÂMETROS DO COMPILADOR

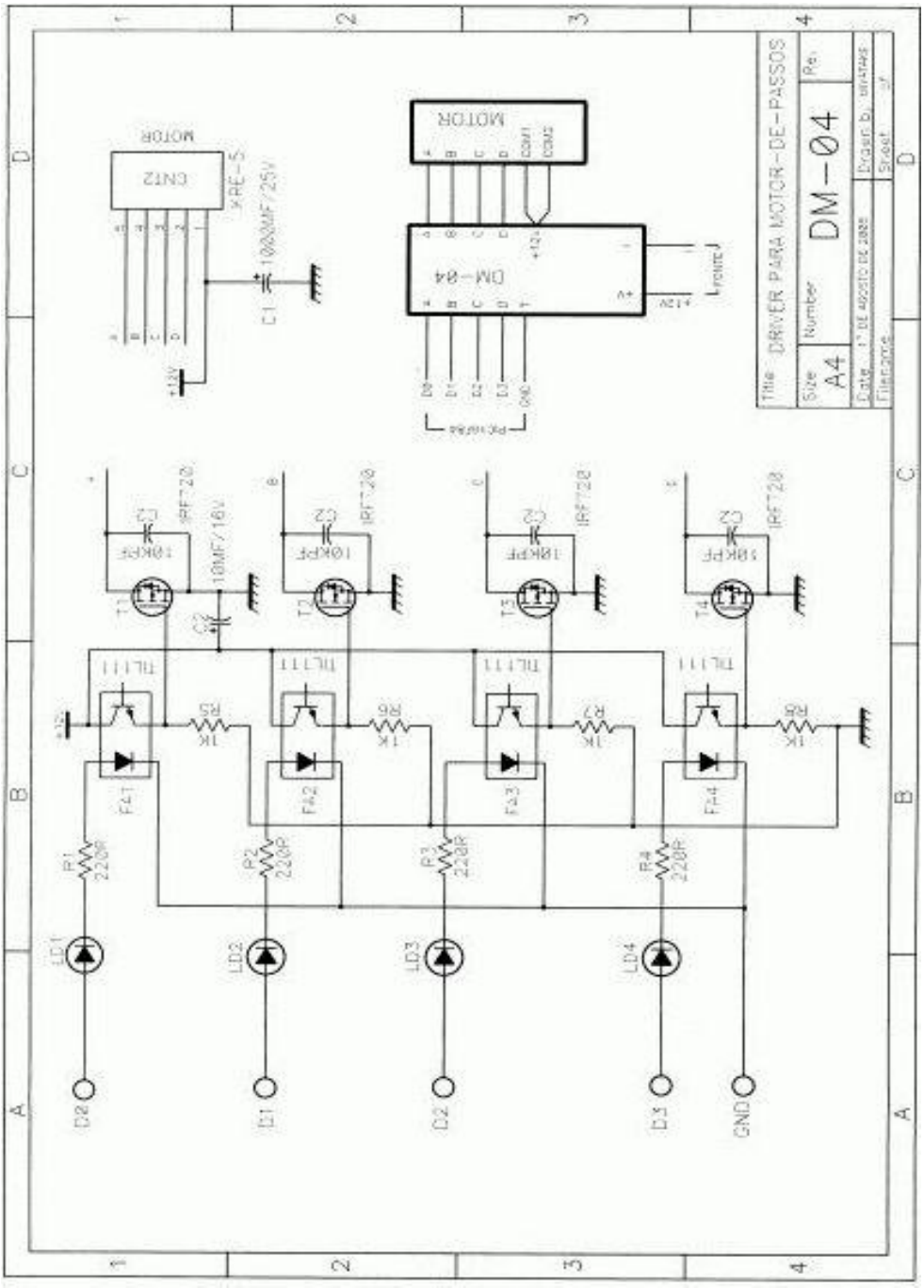
ATENÇÃO ! : Não esquecer de incluir, sempre que for iniciar uma **Fonte** (File) :

NOTA : Só utilize o sufixo "**A**" no "LIST P = 16F84 se o MPLAB, também tiver configurado como **PIC16F84A** (Vide o rodapé).

16 – CIRCUITO PARA INTERFACEAR UM MOTOR DE PASSOS

Este circuito permite que a PLACA EXPERIEMTAL comande os movimentos de um motor-de-passos, mantendo-se uma isolação galvânica de aproximadamente 1kVolts entre o circuito da Placa Experimental/Motor-de-passos.

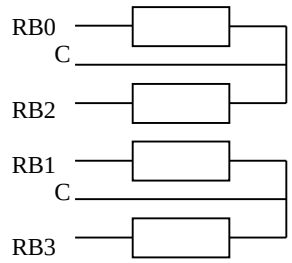
Esta separação, garante ao PIC16F84 a imunidade à qualquer ruído que o motor-de-passos possa a vir gerar, e que muitas vezes, causa o mal funcionamento ou destruição deste microcontrolador.



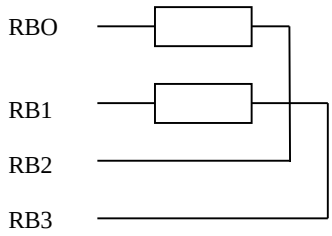
Circuito Driver para Motor de Passos de 4 Fases

17- PROGRAMA EXPERIMENTAL PARA MOVER UM MOTOR DE PASSOS.

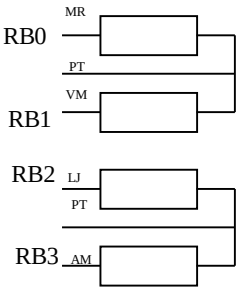
```
;===============================================================================
; Objetivo : É implementar um software que gera movimentos  em  um Motor  de  Passsos
; O ajuste da velocidade   e o sentido da rotação fica gravado  na EEPROM, de modo que
; mesmo desligado  da fonte, mantém memorizados os últimos  ajustes.
; RA0 = Tecla de para/move (S1), RA1 = Sensor de mudança de Sentido (S2)
; RA2 = Tecla de Aumento de Velocidade (S3), RA3 = Tecla de Redução de Velocidade (S4)
; RB0, RB1, RB2  e RB3 =  ( p/  Motor )        -  RB4, RB5, RB6 e RB7 =  não utilizados
;                                                    APLICÁVEL  NO CIRCUITO  DM-04

;===============================================================================
;
;          RB0
;            C
;
;          RB2
;
;          RB1
;            C
;
;          RB3
```



```
;          RBO
;
;          RB1
;
;          RB2
;
;          RB3
```



```
;          RB0
;
;          RB1
;
;          RB2
;
;          RB3
```



(Não funciona no DM-04)

```
          list p = 16F84                                                    PT = NEGATIVO
          Radix dec
          Include <P16F84A.INC>
;------------------------------------------------- define fusíveis-----------------------------------------------

          __config _xt_osc & _cp_off & _wdt_off & _pwrte_on

;---------------------------------------Reserva os espaços na RAM -----------------------------------------------

TEMP_1:             EQU     012         ; REGISTRO_1 TEMPORÁRIO VARIÁVEL (reserva)
TEMP_2:             EQU     013         ; REGISTRO_2 TEMPORÁRIO VARIÁVEL (Veloc.)
TEMP_3:             EQU     014         ; REGISTRO_3 TEMPORÁRIO VARIÁVEL (Horário)
TEMP_4:             EQU     015         ; RESERVA UM ESPAÇO NA RAM (Anti_horário)
TM1:                EQU     016         ; VARIÁVEL1 DO "TIMER"
TM2:                EQU     017         ; VARIÁVEL2 DO "TIMER"
TM3:                EQU     018         ; RESERVA A POSIÇÃO NA RAM
TM4:                EQU     019         ; RESERVA A POSIÇÃO NA RAM
STATUS2:            EQU     020         ; COPIA DO "STATUS"
W2:                 EQU     021         ; COPIA DO "W"
LISTA:              EQU     022         ; RESERVA UM ESPAÇO NA RAM
SENTIDO:            EQU     023         ; RESERVA UM ESPAÇO NA RAM
VELOCIDADE:         EQU     024         ; RESERVA UM ESPAÇO NA RAM

; ------------------------------------------------ endereço de RESET ------------------------------------------------

          ORG               00H

          GOTO    INICIO              ; VAI PARA O "INICIO"

;------------------------------------------------ endereço da INTERRUPÇÃO ----------------------------------------

          ORG               04H               ; À cada 512us, interrupção por TMR0 cai aqui

          MOVWF             W2                ; SALVA "W" ORIGINAL
          MOVF              STATUS,W          ; COPIA "STATUS" ORIGINAL E "W"
          MOVWF             STATUS2           ; SALVA "STATUS" ORIGINAL EM "STATUS2"

          CLRF              PCLATH            ; ZERA O "PCLATH"
```

```
; ------------------------------------ Executa  as TAREFAS abaixos ------------------------------------------

        INCF        LISTA,F         ; INICIA  CO M O END. DA LISTA JÁ INCREMENTADO
        MOVF        LISTA,W         ; COPIA A "LISTA" EM "W"
        ADDWF       PCL,F           ; EXECUTA A "TAREFA" DA "LISTA"

Lista de Tarefas:   ; À cada 1.024 us ( 512us x 2 = 1.024us )
        NOP                         ; PRIMEIRA "TAREFA" COMEÇA JÁ COM END. = + 01
        GOTO        MOV_MOTOR       ; O MOTOR DE PASSOS COMEÇA A FUNCIONAR
        GOTO        RD_TECLADOS     ; FAZ A LEITURA DE TECLADOS (SENSORES)
        CLRF        LISTA           ; ZERA A "LISTA" PARA RECOMEÇAR NOVAMENTE

; ------------------------------------------------ Fim das Interrupções (TMR0) ---------------------------------

FIM_INT:            ; Ao término das tarefas, retorna aqui

        MOVF        W2,W            ; RESTAURA O "W" ANTERIOR
        MOVF        STATUS2,W       ; COPIA EM "W" O "STATUS2"
        MOVWF       STATUS          ; RESTABELECE O VALOR ORIGINAL DE "STATUS"
        MOVLW       B'10100000'     ; CARREGA "W" COM B'10100000'
        MOVWF       INTCON          ; REABILITA A INTERRUPÇÃO
        RETFIE                      ; FIM DA INTERRUPÇÃO

; ====================== Aqui é o ponto de entrada deste programa ========================

INICIO:

; ------------------------------------------------------- condições iniciais do PIC ----------------------------------------

        CLRF        PORTB           ; LIMPA "PORTB" (pendência anterior)
        MOVLW       B'00001111'     ; CARREGA "W"  HABILITAÇÃO DO "PORTA" = ENTRADA
        BSF         STATUS,RP0      ; ACESSA O "BANCO1"
        CLRF        TRISB           ; HABILITA "PORTB" COMO SAÍDA
        MOVWF       TRISA           ; HABILITA AS TECLAS : S1, S2, S3 e S4
        BCF         STATUS,RP0      ; VOLTA PARA O "BANCO0"
        CLRF        LISTA           ; ZERA A REFERÊNCIA DA "LISTA" DE "TAREFAS"

; ------------------------------------------ Condições iniciais do Motor de Passos ------------------------------

        MOVLW       B'00000001'     ; SETA APENAS O PRIMEIRO BIT DO "PORTB"
        MOVWF       TEMP_3          ; PARA MOVER NO SENTIDO HORÁRIO

        MOVLW       B'00001000'     ; SETA APENAS O QUARTO BIT DO "PORTB"
        MOVWF       TEMP_4          ; PARA MOVER NO SENTIDO ANTI-HORÁRIO

; ------------------------------------------------------ Ajuste do Prescaler ----------------------------------------------

        CLRF        INTCON          ; DESABILITA TEMPORARIAMENTE AS INTERRUPÇÕES
        BSF         STATUS,RP0      ; BANCO1
        MOVLW       B'10000000'     ; 1:2 - TMR0 (2 x  256 x  1us = 512us)
        MOVWF       OPTION_REG      ; AJUSTA O "PRESCALER"
        BCF         STATUS,RP0      ; BANCO0

; ------------------------ Aqui é atualizada a última "VELOCIDADE" e " SENTIDO" ------------------------------------

        ; Atualiza o último "SENTIDO" e "VELOCIDADE" que foi salva na EEPROM

        MOVLW       00              ; CARREGA "W" COM 00 (PRIMEIRO END. NA EEPROM)
        MOVWF       EEADR           ; CARREGA "EEADR" COM O ENDEREÇO INICIAL EEPROM
        BSF         STATUS,RP0      ; BANCO1
        BSF         EECON1,RD       ; SETA A LEITURA EM "EECON1"
        BCF         STATUS,RP0      ; BANCO0
        MOVF        EEDATA,W        ; COPIA EM "W" O VALOR LIDO EM "EEDATA"
        MOVWF       VELOCIDADE      ; RESTAURA O ANTIGO VALOR QUE ESTAVA NA EEPROM

        MOVLW       01              ; COPIA "W" O SEGUNDO ENDEREÇO NA EEPROM
        MOVWF       EEADR           ; CARREGA "EEADR" COM O ENDEREÇO INICIAL EEPROM
```

```
        BSF     STATUS,RP0      ; BANCO1
        BSF     EECON1,RD       ; SETA A LEITURA EM "EECON1"
        BCF     STATUS,RP0      ; BANCO0
        MOVF    EEDATA,W        ; COPIA EM "W" O VALOR LIDO EM "EEDATA"
        MOVWF   SENTIDO         ; RESTAURA O ANTIGO VALOR QUE ESTAVA NA EEPROM

; ------------------------------------------------- Habilita a interrupção agora  ----------------------------------------------

        MOVLW   B'10100000'     ; CARREGA W COM B'10100000' (GIE = 1, T0IE = 1)
        MOVWF   INTCON          ; REABILITA A INTERRUPÇÃO POR TMR0

; ================================== LOOP ====================================

        NOP                     ; Fica neste LOOP aguardando uma interrupção (TMR0) que

        NOP                     ; ocorrerá à cada 512us. Vide  ajuste do PRESCALER
        GOTO    $-2             ; Volta duas linhas anteriores

; ============================================================================

; ---------------------- Inicia aqui os movimentos do motor, conforme o BIT 0 do SENTIDO ----------------------

HORÁRIO:        ;  Movimento Horário

        MOVF    TEMP_3,W        ; COPIA O "TEMP_3" EM W"
        MOVWF   PORTB           ; SETA O BIT NO "PORTB" CONFORME "W"
        CALL    DELAY_P         ; DEFINE A LARGURA DO PULSO
        CLRF    PORTB           ; RESETA O "PORTB"
        CALL    DELAY_V         ; DEFINE A VELOCIDADE
        RLF     TEMP_3,W        ; RODA UM BIT À ESQUERDA NO "TEMP_3"
        BTFSC   TEMP_3,3        ; NÃO CHEGOU NO QUARTO BIT, PULA UMA LINHA
        GOTO    BIT_0           ; SE CHEGOU, SALTA PARA "BIT_0"
        MOVWF   TEMP_3          ; COPIA A POSIÇÃO DO BIT EM "TEMP_3"
        GOTO    FIM_INT         ; FIM DE INTERRUPÇÃO
BIT_0:
        MOVLW   B'0000001'      ; SETA APENAS O PRIMEIRO BIT DO "PORTB"
        MOVWF   TEMP_3          ; DEIXA, ASSIM PREPARADO PARA UMA NOVA PARTIDA
        GOTO    FIM_INT         ; FIM DESTA INTERRUPÇÃO

ANTI_HORÁRIO:   ; Movimento Anti Horário

        MOVF    TEMP_4,W        ; COPIA O "TEMP_4" EM "W"
        MOVWF   PORTB           ; SETA O BIT NO "PORTB" CONFORME "W"
        CALL    DELAY_P         ; DEFINE A LARGURA DO PULSO
        CLRF    PORTB           ; RESETA O "PORTB"
        CALL    DELAY_V         ; DEFINE A VELOCIDADE
        RRF     TEMP_4,W        ; RODA UM BIT À DIREITA NO "TEMP_4"
        BTFSC   TEMP_4,0        ; NÃO CHEGOU NO PRIMEIRO BIT, PULA UMA LINHA
        GOTO    BIT_3           ; SE CHEGOU, SALTA PARA "BIT_3"
        MOVWF   TEMP_4          ; COPIA A POSIÇÃO DO BIT EM "TEMP_4"
        GOTO    FIM_INT         ; FIM DE INTERRUPÇÃO
BIT_3:
        MOVLW   B'00001000'     ; SETA APENAS O QUARTO BIT DO "PORTB"
        MOVWF   TEMP_4          ; DEIXA, ASSIM APREPARADO PARA UMA NOVA PARTIDA
        GOTO    FIM_INT         ; FIM DESTA  INTERRUPÇÃO

; ------------------------------------------------- Aqui se faz a leitura das teclas ----------------------------------------------

RD_TECLADOS:  ; As teclas acionadas assumem o valor "0"

        BTFSS   PORTA,0         ; SE A TECLA "S1" ESTÁ PRESSIONADA, AGUARDA
        GOTO    AGUARDA         ;
        BTFSS   PORTA,1         ; SE A TECLA "S2" ESTÁ PRESSIONADA, MUDA SENTIDO
        GOTO    MUDA_SENTIDO;
        BTFSS   PORTA,2         ; SE A TECLA "S3" ESTÁ PRESSIONADA, AUMENTA VELOC.
        GOTO    AUMENTA_VEL ;
        BTFSS   PORTA,3         ; SE A TECLA "S4" ESTÁ PRESSIONADA, DIMINUI VELOC.
```

```
        GOTO        DIMINUI_VEL    ;
        GOTO        FIM_INT        ; FIM DESTA INTERRUPÇÃO

; ------------------------------ Aqui, aguarda até a tecla S1 ser novamente pressionada ----------------------------------------

AGUARDA:    ; Nesta situação, somente o "S1" permite refazer o movimento do motor

        CALL        DELAY_500MS    ; (DELAY P/ NÃO CAUSAR O RETORNO IMEDIATO)
        BTFSC       PORTA,0        ; SE A TECLA "S1" = 0,  RESTARTA O MOTOR
        GOTO        $-1            ; FICA EM LOOP  (VOLTA UMA LINHA)
        CALL        DELAY_500MS    ; (DELAY P/ NÃO CAUSAR O RETORNO IMEDIATO)
        GOTO        MOV_MOTOR      ;  RESTARTA O MOTOR

; ------------------------------------------------ Aqui define o movimento do Motor ----------------------------------------

MOV_MOTOR:          ; Aqui define em que sentido deve girar o motor

        BTFSC       SENTIDO,0      ; VERIFICA O BIT 0 DO SENTIDO
        GOTO        ANTI_HORÁRIO   ; SE = 1, MOVE NO SENTIDO ANTI HORÁRIO
        GOTO        HORÁRIO        ; SE = 0, MOVE NO SENTIDO HORÁRIO

; ------------------------------------------------- Aqui muda o sentido da rotação ----------------------------------------

MUDA_SENTIDO:       ; Complementa o "BIT0"  do "SENTIDO"

        CALL        DELAY_500MS    ; (DELAY P/ NÃO CAUSAR O RETORNO IMEDIATO)
        COMF        SENTIDO,F      ; COMPLEMENTA O BIT DO "SENTIDO" (BIT0)
        GOTO        WR_SENTIDO     ; GRAVA NA EEPROM O NOVO "SENTIDO" DA ROTAÇÃO

; ------------------------------------- Aqui define a velocidade dos movimentos ----------------------------------------

AUMENTA_VEL:        ; AUMENTA A "VELOCIDADE"

        DECF        VELOCIDADE,W ; DECREMENTA A "VELOCIDADE" EM "W"
        MOVWF       TEMP_2       ; SALVO O VALOR EM "TEMP_2" (PROVISORIAMENTE)
        MOVLW       5            ; CARREGA "W" COM 5 (MÍNIMO)
        SUBWF       TEMP_2,W     ; COMPARA COM 5 (COM RESULTADO EM W)
        BTFSC       STATUS,Z     ; SE FOR DIFERENTE, PULA UMA LINHA
        GOTO        FIM_INT      ; É IGUAL, FIM DE DECREMENTAÇÃO
        MOVF        TEMP_2,W     ; COPIA "TEMP_2" EM "W"
        MOVWF       VELOCIDADE   ; ATUALIZA O VALOR DE "VELOCIDADE"
        GOTO        WR_VELOCIDADE ; GRAVA NA EEPROM A NOVA "VELOCIDADE"

DIMINUI_VEL:        ; DIMINUI A "VELOCIDADE"

        INCF        VELOCIDADE,W ; INCREMENTA A "VELOCIDADE" EM "W"
        MOVWF       TEMP_2       ; SALVO O VALOR EM "TEMP_2" (PROVISORIAMENTE)
        MOVLW       255          ; CARREGA "W" COM 250
        SUBWF       TEMP_2,W     ; COMPARA COM 5 (COM RESULTADO EM W)
        BTFSC       STATUS,Z     ; SE FOR DIFERENTE, PULA UMA LINHA
        GOTO        FIM_INT      ; É IGUAL, FIM DE DECREMENTAÇÃO
        MOVF        TEMP_2,W     ; COPIA "TEMP_2" EM "W"
        MOVWF       VELOCIDADE   ; ATUALIZA O VALOR DE "VELOCIDADE"
        GOTO        WR_VELOCIDADE ; GRAVA NA EEPROM A NOVA "VELOCIDADE"

; ------------------------------------------ Aqui define a velocidade do motor ----------------------------------------

DELAY_V:            ; Velocidade = Delay_v + Delay_p =

        MOVF        VELOCIDADE,W ; CARREGA EM "W" O VALOR DE "VELOCIDADE"
        CALL        DELAY        ; CHAMA "DELAY" COM "TM1" DEFINIDO EM "W"
        RETURN

; ------------------------------------------------- Aqui define a largura dos pulsos ----------------------------------------

DELAY_P:
```

```
        MOVLW   10              ; VALOR APROXIMADO, CONFORME A CONSTANTE  "W"
        CALL    DELAY           ; CHAMA "DELAY" COM O "TM1" DEFINIDO EM "W"
        RETURN

; ---------------------------------------------- Aqui define Delay de 250 milisegundos  ----------------------------------------

DELAY_500MS:

        MOVLW   02              ; CARREGA "W" COM NÚMERO DE VEZES (2) À EXECUTAR
        MOVWF   TM3             ; COPIA EM "TM3"
        MOVLW   250             ; VALOR APROXIMADO DE 250 MS
        CALL    DELAY           ; EXECUTA A ROTINA DELAY COM W = 250
        DECFSZ  TM3,F           ; DECREMENTA O "TM3"
        GOTO    $-3             ; SE NÃO ZEROU, VOLTA 3 LINHAS
        RETURN

; ------------------------------------------------- Subrotina de Delay Genérico ---------------------------------------------

DELAY:                  ; Aqui o valor do TM1 define conforme o que vem através do W

        MOVWF   TM1             ; COPIA O VALOR  DE "W" NA VARIÁVEL "TM1"

LOOP1:  MOVLW   250             ; CARREGA "W" COM O VALOR 248
        MOVWF   TM2             ; COPIA O "VELOCIDADE" NA VARIÁVEL "TM2"
        DECFSZ  TM2,F           ; DECREMENTA "TM2", SE Z=1, PULA UMA LINHA
        GOTO    $-1             ; CONTINUA A DECREMENTAR "TM2" EM "LOOP2"
        DECFSZ  TM1,F           ; DECREMENTA "TM1", SE Z=1, PULA  LINHA
        GOTO    LOOP1           ; RETORNA PARA "LOOP1"
        RETURN

; ---------------------------------------------- Aqui inicia a gravação na EEPROM ----------------------------------------

WR_VELOCIDADE:          ; Inicia a gravação da "VELOCIDADE" na EEPROM

        BCF     INTCON,GIE      ; DESABILITA QUALQUER INTERRUPÇÃO

        MOVF    VELOCIDADE,W ; COPIA O DADO DA "VELOCIDADE" EM "W"
        MOVWF EE    DATA        ; CARREGA O DADO EM "EEDATA"
        MOVLW   00              ; CARREGA "W" COM O ENDEREÇO DA EEPROM
        MOVWF   EEADR           ; COPIA EM "EEADR"
        CALL    WR_MEM          ; GERA O "STATUS" PARA A ESCRITA NA EEPROM
        GOTO    FIM_INT         ; FIM DESTA INTERRUPÇÃO

WR_SENTIDO:             ; Inicia a gravação do "SENTIDO" na EEPROM

        BCF     INTCON,GIE      ; DESABILITA QUALQUER INTERRUPÇÃO

        MOVF    SENTIDO,W       ; COPIA O DADO DA "SENTIDO" EM "W"
        MOVWF   EEDATA          ; CARREGA O DADO EM "EEDATA"
        MOVLW   01              ; CARREGA "W" COM O ENDEREÇO DA EEPROM
        MOVWF   EEADR           ; COPIA EM "EEADR"
        CALL    WR_MEM          ; GERA O "STATUS" PARA A ESCRITA NA EEPROM
        GOTO    FIM_INT         ; FIM DESTA INTERRUPÇÃO

; ---------------------------------------- Status de Escrita na Memória EEPROM ----------------------------------

WR_MEM:                 ; Para efetuar escrita na EEPROM, é necessário esta rotina

        BSF     STATUS,RP0      ; BANCO1
        BSF     EECON1,WREN     ; HABILITA A ESCRITA NA EEPROM
        MOVLW   55H             ; CARREGA "W" COM 55H
        MOVWF   EECON2          ; COPIA "W" EM "EECON2"
        MOVLW   0AAH            ; CARREGA AAH EM "W"
        MOVWF   EECON2          ; COPIA "W' EM "EECON2"
        BSF     EECON1,WR       ; INICIA A ESCRITA
        BCF     EECON1,WREN     ; DESABILITA A ESCRITA NA EEPROM
        BTFSC   EECON1,WR       ; VERIFICA SE ACABOU DE ESCREVER
```

```
        GOTO        $-1             ; SE NÃO, AGUARDA (EECON1,WR = 0)
        BCF         STATUS,RP0      ; BANCO0
        BSF         INTCON,GIE      ; REALIBILITA INTERRUPÇÕES
        RETURN

        END                         ; FIM DESTA PROGRAMAÇÃO
```

18 – SUBROTINAS CONVERSORES :

18.1 – CONVERSOR HEXA 1 BYTE –> DECIMAIS DE 3 VARIÁVEIS

```
;*******************************************************************************
; ESTE PROGRAMA CONVERTE DÍGITO HEXA DE 1 BYTE EM DECIMAIS DE 3 VARIÁVEIS
; SIMULE ATRAVÉS DO F7 OU F9 E GUARDE ATÉ OCORRER "STACK UNDERFLOW".
; VISUALIZE: DP1; DP2 e DP3 EM DECIMAIS. RESETE O PROGRAMA ANTES DE INICIAR A
; SIMULAÇÃO.
;*******************************************************************************


        LIST P=16F84
        RADIX DEC
        INCLUDE <P16F84.INC>

DP1:    EQU     0CH     ; UNIDADE
DP2:    EQU     0DH     ; DEZENA
DP3:    EQU     0EH     ; CENTENA

        ORG     00H     ; INÍCIO DA GRAVAÇÃO

        MOVLW           0FFH            ; VALOR UTILIZADO PARA CONVERSÃO : 0FFH

CONV_B_D:       ; SUBROTINA PARA CONVERTER BINÁRIO (1 BYTE) EM DECIMAL

        MOVWF           DP1             ; COPIA EM DP1 O VALOR HEXA A SER CONVERTIDO
        CLRF            DP2             ; LIMPA DP2
        CLRF            DP3             ; LIMPA DP3

BIN_1:
        MOVLW           100             ; CARREGA W COM 100
        SUBWF           DP1,W           ; VERIFICA SE DP1 É MENOR QUE 100
        BTFSS           STATUS,C        ; SE FOR MENOR, COPIA ESSE VALOR EM DP1
        GOTO            BIN_2           ; SE NÃO, VAI EM BIN_2
        MOVWF           DP1             ; DP1 = W = 100
        INCF            DP3,F           ; INCREMENTA DP3
        GOTO            BIN_1           ; VAI PARA BIN_1
BIN_2:
        MOVLW           10              ; CARREGA W COM 10
        SUBWF           DP1,W           ; COMPARA DP1 COM 10
        BTFSS           STATUS,C        ; SE FOR MENOR, COPIA EM DP1
        RETURN                          ; RETORNA
        MOVWF           DP1             ; DP1 = W = 10
        INCF            DP2,F           ; INCREMENTA DP2
        GOTO            BIN_2           ; VAI PARA BIN2


        END
```

18.2 - CONVERSOR HEXA 2 BYTES –> DECIMAIS DE 3 VARIÁVEIS

```
;================================================================================================
; ESTE PROGRAMA CONVERTE HEXADECIMAL (2 BYTES) EM DECIMAIS DE 3 VARIÁVEIS
; VISUALIZE O RESULTADO EM DP0; DP1; DP2 e DP3 EM DECIMAL. RESET O PROGRAMA ANTES DE ACIONAR O
; F9 (EXECUÇÃO RÁPIDA). NÃO ESQUECER DE DIGITAR O ZERO ANTES DAS LETRAS DO NÚMEROS HEXAS.
; AGUARDE ATÉ OCORRER "STACK UNDERFLOW"
;================================================================================================

        LIST P=16F84
        RADIX DEC
        INCLUDE <P16F84.INC>

D0:     EQU     0CH     ; UNIDADE
D1:     EQU     0DH     ; DEZENA
D2:     EQU     0EH     ; CENTENA
D3:     EQU     0FH     ; MILHAR
D4:     EQU     10H     ; DEZENA DE MILHAR
A0:     EQU     11H     ; LSB HEXA A SER CONVERTIDO EM DECIMAL
A1:     EQU     12H     ; MSB HEXA A SER CONVERTIDO EM DECIMAL
B0:     EQU     13H     ; LSB DO SUBTRAENDO
B1:     EQU     14H     ; MSB DO SUBTRAENDO
TEMP0:  EQU     15H     ; ARQUIVO TEMPORÁRIO
TEMP1:  EQU     16H     ; ARQUIVO TEMPORÁRIO

        ORG     00H

        MOVLW           0FFH            ; CARREGA AQUI O VALOR A SER CONVERTIDO
        MOVWF           A1              ; NESTE EXEMPLO : 0FFFFH
        MOVLW           0FFH            ; A1 = MSB HEXA
        MOVWF           A0              ; A0 = LSB HEXA


BIN16DEC:
        CLRF            D0              ; LIMPA OS VALORES INICIAIS DOS DÍGITOS DECIMAIS
        CLRF            D1              ;
        CLRF            D2              ;
        CLRF            D3              ;
        CLRF            D4              ;
BIN16DEC_1:
        MOVF            A0,W            ; COPIA O LSB BINÁRIO...
        MOVWF           TEMP0           ; PARA O TEMP0
        MOVF            A1,W            ; COPIA O MSB HEXA...
        MOVWF           TEMP1           ; PARA O TEMP1
        MOVLW           010H            ;
        MOVWF           B0              ;
        MOVLW           27H             ;
        MOVWF           B1              ; B=2710H (1.000 EM DECIMAL)
        CALL            SUB16B          ; SUBTRAI 10.000 DO DECIMAL DO HEXA
        BTFSS           STATUS,C        ; TESTA SE O RESULTADO É POSITIVO OU ZERO
        GOTO            BIN16DEC_2      ; NÃO ? ENTÃO VAI PARA A PRÓXIMA FASE
        INCF            D4,F            ; SIM ?  ENTÃO INCREMENTA DEZENA DE MILHAR
        GOTO            BIN16DEC_1      ; REINICIA O CICLO
BIN16DEC_2:
        MOVF            TEMP0,W         ; RETORNA O VALOR ANTERIOR...
        MOVWF           A0              ; ...DE A0
        MOVF            TEMP1,W         ; RETORNA O VALOR ANTERIOR...
        MOVWF           A1              ; ...DE A1
BIN16DEC_3:
        MOVF            A0,W            ; COPIA O LSB HEXA...
        MOVWF           TEMP0           ; .... PARA O TEMP0
        MOVF            A1,W            ; COPIA O MSB HEXA....
        MOVWF           TEMP1           ; ... PARA O TEMP1
        MOVLW           0E8H            ;
        MOVWF           B0              ;
        MOVLW           03H             ;
        MOVWF           B1              ; B=03E8H (1.000 EM DECIMAL)
```

```
        CALL    SUB16B      ; SUTRAI 1.000 DECIMAL DO VALOR BINÁRIO
        BTFSS   STATUS,C    ; TESTA SE O RESULTADO É POSITIVO OU ZERO
        GOTO    BIN16DEC_4  ; NÃO ? ENTÃO VAI PARA A PRÓXIMA FASE
        INCF    D3,F        ; SIM ? ENTÃO INCREMENTA MILHAR
        GOTO    BIN16DEC_3  ; REINICIA O CICLO
BIN16DEC_4:
        MOVF    TEMP0,W     ; RETORNA O VALOR ANTERIOR...
        MOVWF   A0          ; ... DE A0
        MOVF    TEMP1,W     ; RETORNA O VALOR ANTERIOR ...
        MOVWF   A1          ; ...DE A1
BIN16DEC_5:
        MOVF    A0,W        ; COPIA O LSB HEXA
        MOVWF   TEMP0       ; .... PARA O TEMP0
        MOVF    A1,W        ; COPIA O MSB HEXA
        MOVWF   TEMP1       ; ... PARA O TEMP1
        MOVLW   100         ;
        MOVWF   B0          ;
        CLRF    B1          ; B=100 DECIMAL
        CALL    SUB16B      ; SUBTRAI 100 DO DECIMAL HEXA (A)
        BTFSS   STATUS,C    ; TESTA SE O RESULTADO É POSITIVO OU ZERO
        GOTO    BIN16DEC_6  ; NÃO ? ENTÃO VA PARA PRÓXIMA FASE
        INCF    D2,F        ; SIM ? ENTÃO INCREMENTA A CENTENA
        GOTO    BIN16DEC_5  ; REINICIA O CICLO
BIN16DEC_6:
        MOVF    TEMP0,W     ; RETORNA O VALOR ANTERIOR ...
        MOVWF   A0          ; ....DE A0
        MOVF    TEMP1,W     ; RETORNA O VALOR ANTERIOR ....
        MOVWF   A1          ; ... DE A1
BIN16DEC_7:
        MOVF    A0,W        ; COPIA O LSB HEXA ....
        MOVWF   TEMP0       ; ..... PARA O TEMP0
        MOVF    A1,W        ; COPIA O MSB HEXA ...
        MOVWF   TEMP1       ; .... PAR O TEMP1 ...
        MOVLW   10          ;
        MOVWF   B0          ;
        CLRF    B1          ; B=10 DECIMAL
        CALL    SUB16B      ; SUBTRAI 10 DECIMAL DO VALOR HEXA (A)
        BTFSS   STATUS,C    ; TESTA SE O RESULTADO É POSITIVO OU ZERO
        GOTO    BIN16DEC_8  ; NÃO ? ENTÃO VAI PARA A FASE FINAL
        INCF    D1,F        ; SIM ? ENTÃO INCREMENTA AS DEZENAS
        GOTO    BIN16DEC_7  ; REINICIA O CICLO
BIN16DEC_8:
        MOVF    TEMP0,W     ; COPIA O VALOR TEMPORÁRIORESTANTE...
        MOVWF   D0          ; .... PARA AS UNIDADES
        RETURN              ; RETORNA
SUB16B:     ; Subrotina para subtrair operações de 2 bytes
        MOVF    B0,W        ;
        SUBWF   A0,F        ;
        MOVLW   01H         ;
        BTFSS   STATUS,C    ;
        SUBWF   A1,F        ;
        BTFSS   STATUS,C    ;
        GOTO    EMPRESTA    ;
        MOVF    B1,W        ;
        SUBWF   A1,F        ;
FIM_SUB16B:
        MOVF    A0,W        ;
        BTFSC   STATUS,Z    ;
        MOVF    A1,F        ;
        RETURN              ;
EMPRESTA:
        MOVF    B1,W        ;
        SUBWF   A1,W        ;
        BCF     STATUS,C    ;
        GOTO    FIM_SUB16B  ;

        END
```

18.3 – CONVERSOR DE 4 DECIMAIS -> HEXA DE 2 BYTES

```
; ==========================================================================
; ESTE PROGRAMA CONVERTE DECIMAIS DE 4 VARIÁVEIS EM HEXADECIMAL
; VISUALIZE OS RESULTADOS EM : A0 e A1 (HEXA). RESET O PROGRAMA ANTES DE
; FAZER A CONVERSÃO. UTILIZE F6 (RESET) E F9 (PARA A CONVERSÃO)
;==========================================================================


        LIST P=16F84
        RADIX DEC
        INCLUDE <P16F84.INC>

D0:     EQU     0CH     ; UNIDADE (DECIMAL) A SER CONVERTIDA
D1:     EQU     0DH     ; DEZENA (DECIMAL) A SER CONVERTIDA
D2:     EQU     0EH     ; CENTENA (DECIMAL) A SER CONVERTIDA
D3:     EQU     0FH     ; MILHAR (DECIMAL) A SER CONVERTIDA
D4:     EQU     10H     ; DEZENA MILHAR (DECIMAL) A SER CONVERTIDA
A0:     EQU     11H     ; LSB HEXA CONVERTIDO
A1:     EQU     12H     ; MSB HEXA  CONVERTIDO

        ORG     00H

                ; Carrega aqui os valores decimais a serem convertidos em hexa

        MOVLW           05              ; VALOR DA UNIDADE
        MOVWF           D0              ;

        MOVLW           05              ; VALOR DA DEZENA
        MOVWF           D1              ;

        MOVLW           02              ; VALOR DE CENTENA
        MOVWF           D2              ;

        MOVLW           00              ; VALOR DA MILHAR
        MOVWF           D3              ;

        MOVLW           00              ; VALOR DA DEZENA DE MILHAR
        MOVWF           D4              ;

DEC16BIN:
        CLRF            A1              ; LIMPA A1
        MOVF            D0,W            ; COPIA O VALOR DE D0 EM A0
        MOVWF           A0              ;
DEC16BIN_1:
        MOVLW           01              ; SUBTRAI 01...
        SUBWF           D4,F            ; DAS DEZENAS DE MILHARES
        BTFSS           STATUS,C        ; VERIFICA SE O RESULTADO É NEGATIVO
        GOTO            DEC16BIN_2      ; SE FOR NEGATIVO, VAI PARA DEC16BIN_2
        MOVLW           27H             ; SOMA....
        ADDWF           A1,F            ;
        MOVLW           10H             ; 2710H (10.000 EM DECIMAL)...
        ADDWF           A0,F            ; AO RESULTADO
        BTFSC           STATUS,C        ; VERIFICA SE O RESULTADO LSB TRANSBORDOU
        INCF            A1,F            ; SE TRANSBORDOU, INCREMENTA MSB
        GOTO            DEC16BIN_1      ;
DEC16BIN_2:
        MOVLW           01              ; SUBTRAI 1....
        SUBWF           D3,F            ; DAS MILHARES
        BTFSS           STATUS,C        ; VERIFICA SE O RESULTADO É NEGATIVO
        GOTO            DEC16BIN_3      ; SE FOR NEGATIVO VAI PARA DEC16BIN_3
        MOVLW           03H             ; SOMA....
        ADDWF           A1,F            ; .....
        MOVLW           0E8H            ;  ....
        ADDWF           A0,F            ; 3E8H (1.000 EM DECIMAL) AO RESULTADO
```

```
        BTFSC   STATUS,C        ; VERIFICA SE O RESULTADO LSB TRANSBORDOU
        INCF    A1,F            ; SE TRANSBORDOU, INCREMENTA MSB
        GOTO    DEC16BIN_2      ; CONTINUA EM DEC16BIN_2
DEC16BIN_3:
        MOVLW   01H             ; SUBTRAI 01 ....
        SUBWF   D2,F            ; DAS CENTENAS
        BTFSS   STATUS,C        ; VERIFICA SE O RESULTADO É NEGATIVO
        GOTO    DEC16BIN_4      ; SE FOR NEGATIVO, VAI PARA DEC16BIN_4
        MOVLW   100             ; SOMA....
        ADDWF   A0,F            ; 100 DECIMAL AO RESULTADO
        BTFSC   STATUS,C        ; VERIFICA SE O LSB TRANSBORDOU
        INCF    A1,F            ; SE TRANSBORDOU, INCREMENTA MSB
        GOTO    DEC16BIN_3      ; CONTINUA EM DEC16BIN_3
DEC16BIN_4:
        MOVLW   01H             ; SUBTRAI 1...
        SUBWF   D1,F            ; DAS DEZENAS
        BTFSS   STATUS,C        ; VERIFICA SE O RESULTADO É NEGATIVO
        RETURN                  ; FIM DESTA CONVERSÃO
        MOVLW   10              ; .....
        ADDWF   A0,F            ; SOMA 10 AO RESULTADO LSB
        BTFSC   STATUS,C        ; VERIFICA SE TRANSORDOU
        INCF    A1,F            ; SE TRANSBORDOU, INCREMENTA O RESULTADO MSB
        GOTO    DEC16BIN_4      ; CONTINUA EM DEC16BIN_4

        END             ; FIM DESTA PROGRAMAÇÃO
```

19 – **CONVERSOR de TABELAS -> DISPLAY e SAÍDA PARA D.A.**

Este programa é utilizado em aparelhos que tem uma saída com nível analógico de saída monitorado em Display digital de 3 dígitos.

```
;-------------------------------------------------------------------------------------------------------------------------------------------------------
; ESTE PROGRAMA SIMULA UM CIRCUITO QUE GERA UMA INFORMAÇÃO DECIMAL DE TRÊS DÍGITOS, CONFORME UM
; STEP PRÉ-GRAVADO NUMA TABELA, E, SIMULTANEAMENTE ENVIA UM VALOR HEXA DECIMAL  CORRESPONDENTE
; TAMBÉM CONTIDO NUMA TABELA PRÉ DEFINIDA (CURVA DE SAÍDA ANALÓGICA) PARA UM  CONVERSOR D.A.
; É UTILIZADO PARA PROJETOS, POR EXEMPLO: FONTE AJUSTÁVEL TIPO DC COM 3 DIGITOS. PARA CADA VALOR
; EXPOSTO NOS DÍGITOS, HAVERÁ UM VALOR DE NÍVEL DC NA SUA SAÍDA PARA CONVERSOR ANALÓGICO D.A.
; PARA SIMULAR  0  INCREMENTO, FOI  ANEXADA  A  SUBROTINA : "UP_DISPLAY".
;-------------------------------------------------------------------------------------------------------------------------------------------------------

        LIST P=16F84
        RADIX DEC
        INCLUDE <P16F84.INC>

        __CONFIG _CP_OFF & _PWRTE_ON & _WDT_OFF & _XT_OSC

DP1             EQU     12H     ; POSIÇÃO NA RAM P/ SALVAR O VALOR DO DÍGITO 01
DP2             EQU     13H     ; POSIÇÃO NA RAM P/ SALVAR O VALOR DO DÍGITO 02
DP3             EQU     14H     ; POSIÇÃO NA RAM P/ SALVAR O VALOR DO DÍGITO 03
W2              EQU     15H     ; POSIÇÃO NA RAM P/ SALVAR O VALOR DE W NUMA INTERRUPÇÃO
STATUS_2        EQU     16H     ; POSIÇÃO NA RAM P/ SALVAR O VALOR DE STATUS NUMA INTERRUPÇÃO
TAREFAS         EQU     17H     ; POSIÇÃO NA RAM P/ MARCAR POSIÇÃO DE ENDEREÇO DAS TAREFAS
OFFSET          EQU     18H     ; POSIÇÃO NA RAM P/ DEFINIR A POSIÇÃO DAS TABELAS DO DISPLAY
K_OUT           EQU     19H     ; POSIÇÃO NA RAM P/ SALVAR O VALOR DE SAÍDA ANALÓGICA
LSTP            EQU     20H     ; POSIÇÃO NA RAM P/ SALVAR O VALOR DE STEP (DISPLAY)


        ORG     000H            ; INÍCIO DA GRAVAÇÃO

        GOTO    INÍCIO          ; VAI PARA INÍCIO

;////////////////////////////////////////////////////// TRATAMENTO QUANDO OCORRE TRANSBORDO TMR0 ////////////////////////////////////////

;---------------- Cai aqui todas as vezes que ocorre uma interrupção (TMR0)----------------

        ORG     0004H           ; INTERRUPÇÃO VETORIZADA
```

```
; ---------- Salva os valores do W e do Status anteriores à interrupção ----------

        MOVWF   W2              ; SALVA O W ORIGINAL
        MOVF    STATUS,W        ; COPIA O STATUS ORIGINAL EM W
        MOVWF   STATUS_2        ; SALVA O STATUS EM STATUS_2

        GOTO    TMR0_GET        ; INICIA TMR0_GET (TAREFAS)

;---------- Restaura os valores anterior à Interrupção ----------

FIM_INT:

        MOVF    W2,W            ; RESTAURA O W ANTERIOR
        MOVF    STATUS_2,W      ; COPIA EM W O STATUS ANTERIOR
        MOVWF   STATUS          ; RESTABELECE OS VALORES DO STATUS, ANTERIOR À INTERRUPÇÃO
        BSF     INTCON,T0IE     ; HABILITA TMR0
        BCF     INTCON,T0IF     ; LIMPA O FLAG DE INTERRUPÇÃO POR TMR0
        MOVLW   250             ; RECARREGA TMR0
        MOVWF   TMR0            ;
        RETFIE
;-----------------------------------------------------------------------------------------------------------------------------------
; INÍCIO DA EXECUÇÃO DESTE  PROGRAMA - VALORES INICIAIS
;-----------------------------------------------------------------------------------------------------------------------------------

INÍCIO:

        BSF     STATUS,RP0      ; BANCO1
        CLRF    TRISB           ; CONVERTE EM SAÍDA AS PORTA : B
        BCF     STATUS,RP0      ; BANCO 0

        MOVLW   00H             ; INICIA COM OS PORTS : A e B ZERADOS
        MOVWF   PORTA           ; ZERA PORTA
        MOVWF   PORTB           ; ZERA PORTB

        CLRF    DP1             ; ZERA INICIALMENTE OS VALORES DOS DÍGITOS
        CLRF    DP2             ;
        CLRF    DP3             ;
        CLRF    K_OUT           ; INICIA COM AS SAÍDAS ANALÓGICA ZERADAS
        CLRF    OFFSET          ; ZERA OFFSET INICIALMENTE
        CLRF    LSTP            ; ZERA STEPS INICIALMENTE

        MOVLW   250             ; VALOR INICIAL DO RESET (TMR0)
        MOVWF   TMR0            ;

        CLRF    TAREFAS         ; ZERA A POSIÇÃO INICIAL DAS TAREFAS

;-----------------------------------------------------------------------------------------------------------------------------------
; CONDIÇÕES DO TIMER 0
;-----------------------------------------------------------------------------------------------------------------------------------

; PRESCALER DE 1 = 1.000.000Hz/1 = 1.000.000 Hz
; INICIA COM PRESET = 250, 1.000.000/1/6 =  À CADA 6,0 uS TMR0 CAUSA INTERRUPÇÃO

        BCF     INTCON,GIE      ; DESABILITA QUALQUER INTERRUPÇÃO
        BSF     STATUS,RP0      ; BANCO1
        MOVLW   B'00001000'     ; PRESCALER 1:1 WD
        MOVWF   OPTION_REG      ; CLK=4MHZ, CLK = 1000.000 Hz
        BCF     STATUS,RP0      ; BANCO0
        BSF     INTCON,T0IE     ; HABILITA TMR0
        BCF     INTCON,T0IF     ; LIMPA O FLAG DE INTERRUPÇÃO POR TMR0
        BSF     INTCON,GIE      ; HABILITA TODAS AS INTERRUPÇÕES

;////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;
;                               AGUARDA EM LOOP AQUI ATÉ OCORRER UMA INTERRUPÇÃO
;
;////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

MAIN    MOVF    TMR0,F          ; VERIFICA SE  O VALOR DO TMR0 = 0
LOOP_X  BTFSC   STATUS,Z        ;
        GOTO    MAIN            ; SENÃO, AGUARDA EM LOOP
        GOTO    LOOP_X          ;
;---------- Fim das Interrupções ----------

FIM_INT_TMR0:
```

```
        MOVLW  250              ; CAREGA EM W O VALOR DE TMR0
        MOVWF  TMR0             ; CAREGA O VALOR DO PRESET DO CONTADOR TMR0
        BCF    INTCON,T0IF      ; LIMPA O FLAG DO TMR0
        GOTO   FIM_INT          ; FIM DE INTERRUPÇÕES

;-------------------------------------------------------------------------------------------------------------------------------------
        ORG    0100H            ; Novo início da gravação na memória
;-------------------------------------------------------------------------------------------------------------------------------------

TMR0_GET:               ; INICIA AS SEQUÊNCIAS DE TAREFAS

        MOVLW  01H              ; PCLATH = 01H (Atualiza o PCLATH)
        MOVWF  PCLATH           ;

        INCF   TAREFAS,F        ; INCREMENTA POSIÇÃO DE TAREFAS
        MOVF   TAREFAS,W        ; COPIA POSIÇÃO DE TAREFAS EM W
        ADDWF  PCL,F            ; APONTA ENDEREÇO 2 LINHAS ABAIXO

        NOP                     ; ENDEREÇO VAGO
        GOTO   GET_PARAMETRO    ; ROTINA P/ ATUALIZAR OS PARAMETROS,  CONFORME VALOR DO OFFISET
        GOTO   CONV_HEX_DEC     ; ROTINA P/ CONVERTER HEX->DECIMAL (DISPLAY)
        GOTO   UP_DISPLAY       ; INCREMENTA DISPLAY (SIMULANDO TECLA : INCREMENTO DE DISPLAY)
        CLRF   TAREFAS          ; ZERA POSIÇÃO DAS TAREFAS P/ REINICIAR NO TOPO

        GOTO   FIM_INT_TMR0     ; FIM

;------------------------------------------------------------ ROTINA PARA GERAR PARAMETROS ----------------------------------------------

GET_PARAMETRO: ; RECEBE EM W O VALOR CORRESPONDENTE DO NÍVEL ANALÓGICO

        ; AQUI, CARREGA O VALOR DA TABELA GET_STEP E COPIA PARA LSTP
        ; E TAMBÉM O VALOR DO K_OUT, CONFORME O VALOR ESTABELECIDO POR OFFSET.

        MOVF   OFFSET,W         ; RETORNA COM O VALOR DO LSTP EM W
        CALL   GET_STEP         ;
        MOVWF  LSTP             ; CARREGA EM LSTP

        MOVF   OFFSET,W
        CALL   GET_K_OUT        ; RETORNA COM O VALOR DE K_OUT EM W
        MOVWF  K_OUT            ; CARREGA EM K_OUT
        GOTO   FIM_INT_TMR0     ;

GET_STEP:       ; CONTÉM 100 STEPS DE AJUSTE, MAS UTILIZA NESTE EXEMPLO APENAS: 60

        MOVLW  01H              ; PCLATH = 01H (Atualiza o PCLATH)
        MOVWF  PCLATH           ;

        MOVF   OFFSET,W
        ADDWF  PCL,F

        DT     001, 003, 004, 006, 008, 010, 015, 018, 020, 025
        DT     028, 030, 032, 035, 037, 039, 042, 045, 050, 055
        DT     058, 060, 065, 070, 072, 075, 080, 085, 090, 095
        DT     100, 110, 115, 117, 120, 125, 130, 135, 140, 145
        DT     150, 155, 160, 165, 170, 175, 180, 185, 190, 195
        DT     200, 205, 210, 215, 218, 119, 220, 225, 233, 255
        ;DT    000, 000, 000, 000, 000, 000, 000; 000, 000, 000
        ;DT    000, 000, 000, 000, 000, 000, 000, 000, 000, 000
        ;DT    000, 000, 000, 000, 000, 000, 000, 000, 000, 000
        ;DT    000, 000, 000, 000, 000, 000, 000, 000, 000, 000

;-------------------------------------------------------------------------------------------------------------------------------------
        ORG    0200H            ; Novo início da gravação na memória
;-------------------------------------------------------------------------------------------------------------------------------------

GET_K_OUT:  ; CONTÉM 100 STEPS DE AJUSTE, MAS UTILIZA NESTE EXEMPLO APENAS : 60


        MOVLW  02H              ; PCLATH = 02H  (Atualiza o PCLATH)
        MOVWF  PCLATH           ;

        MOVF   OFFSET,W         ; COPIA O ENDEREÇO OFFSET EM W
        ADDWF  PCL,F

        DT     010, 012, 013, 014, 015, 015, 018, 020, 022, 025
        DT     027, 030, 033, 035, 037, 040, 043, 045, 047, 050
```

```
        DT      053, 055, 057, 059, 062, 065, 058, 070, 073, 075
        DT      077, 080, 082, 085, 087, 090, 092, 095, 095, 097
        DT      100, 103, 105, 108, 110, 112, 115, 118, 120, 123
        DT      125, 135, 145, 155, 165, 175, 185, 195, 205, 215
        ;DT     000, 000, 000, 000, 000, 000, 000, 000, 000, 000
        ;DT     000, 000, 000, 000, 000, 000, 000, 000, 000, 000
        ;DT     000, 000, 000, 000, 000, 000, 000, 000, 000, 000
        ;DT     000, 000, 000, 000, 000, 000, 000, 000, 000, 000

;============ SUBROTINA PARA INCREMENTAR E DECREMENTAR DISPLAY ============================
;
; As Subrotinas abaixo: são ativados durante a execução das TAREFAS e serve para incrementar ou
; decrementar o valor do Display e o valor da saída Analógica correspondente.
; Na prática, elas devem ser incorporadas a uma leitura de teclados tipo: "UP" ou "DOWN".
; Neste exemplo, só é possível simular o incremento do valor do Display e da saída analógica  D.A.
;
; ------------------------------------------------------ ROTINA PARA INCREMENTAR DISPLAY ----------------------------------------------

UP_DISPLAY:             ; AQUI, INCREMENTA O VALOR DE OFFSET ATÉ O LIMITE ESTABELECIDO
                        ; E ATUALIZA O VALOR DE SAÍDA ANALÓGICA DE 60 VALORES (STEPS)

        MOVF    OFFSET,W        ; COPIA O VALOR DE OFFSET EM W
        SUBLW   59              ; COMPARA COM 59 (Poderia ser de 100 Steps)
        BTFSC   STATUS,Z        ; SE FOR = 60, NÃO INCREMENTA MAIS
        GOTO    FIM_INT_TMR0    ;
        INCF    OFFSET,F        ; INCREMENTA O OFFSET
        GOTO    FIM_INT_TMR0

; ------------------------------------------------------ ROTINA PARA DECREMENTAR DISPLAY ----------------------------------------------

DOWN_DISPLAY:           ; AQUI,  DECREMENTA O VALOR DE  OFFSET ATÉ O LIMITE DE ZERO
                        ; E ATUALIZA O VALOR DE SAÍDA ANALÓGICA. Neste exemplo = 60 Steps

        MOVF    OFFSET,F        ; VERIFICA SE JÁ NÃO ZEROU
        BTFSC   STATUS,Z        ; SE ZEROU, NÃO DECREMENTA MAIS
        GOTO    FIM_INT_TMR0    ;
        DECF    OFFSET,F        ; SE NÃO, CONTINUA A DECREMENTAR
        GOTO    FIM_INT_TMR0    ;

;================================ CONVERSOR HEXA-DECIMAL ======================================

CONV_HEX_DEC:           ; Os valores da Tabela GET_STEP podem estar expresso em Hexa ou Decimal

        MOVF    LSTP,W          ; VALOR UTILIZADO PARA CONVERSÃO

CONV_B_D:       ; SUBROTINA PARA CONVERTER BINÁRIO (1 BYTE) EM DECIMAL

        MOVWF   DP1             ; COPIA EM DP1 O VALOR HEXA A SER CONVERTIDO
        CLRF    DP2             ; LIMPA DP2
        CLRF    DP3             ; LIMPA DP3
BIN_1:
        MOVLW   100             ; CARREGA W COM 100
        SUBWF   DP1,W           ; VERIFICA SE DP1 É MENOR QUE 100
        BTFSS   STATUS,C        ; SE FOR MENOR, COPIA ESSE VALOR EM DP1
        GOTO    BIN_2           ; SE NÃO, VAI EM BIN_2
        MOVWF   DP1             ; DP1 = W = 100
        INCF    DP3,F           ; INCREMENTA DP3
        GOTO    BIN_1           ; VAI PARA BIN_1
BIN_2:
        MOVLW   10              ; CARREGA W COM 10
        SUBWF   DP1,W           ; COMPARA DP1 COM 10
        BTFSS   STATUS,C        ; SE FOR MENOR, COPIA EM DP1
        GOTO    FIM_INT_TMR0    ; FIM INT_TMR0
        MOVWF   DP1             ; DP1 = W = 10
        INCF    DP2,F           ; INCREMENTA DP2
        GOTO    BIN_2           ; VAI PARA BIN2

        END     ; FIM DESTA PROGRAMAÇÃO
```

AGRADECIMENTO :

Agradeço a relevante colaboração da professora Kimie Matsumoto pelo trabalho de revisão gramatical.

REFERÊNCIAS BIBLIOGRÁFICAS :

Microchip Technolog Inc. Home Page (internet) e Technical Library CD-ROM 2000.

Microcontroladores PIC : Teoria e Prática/ Vidal Pereira da Silva Júnior

Desbravando o PIC / David José de Souza - Editora Érica

EMBEDDED CONTROL HANDBOOK- Microchip/1995

MICROCONTROLADORES PIC -: Técnicas Avançadas / Fábio Pereira – Editora Érica

BANCO 0 (BCF STATUS,RP0)		
00H	**INDF**	Endereçamento indireto
01H	**TMR0**	Registro de contagem do timer 0
02H	**PCL**	Parte baixo do PC
03H	**STATUS**	Registro de STATUS
04H	**FSR**	Ponteiro para endereçamento indireto
05H	**PORTA**	Registro dos pinos do PORTA
06H	**PORTB**	Registro dos pinos do PORTB
07H	**----**	Não implementado
08H	**EEDATA**	Dado lido ou gravado na EEPROM
09H	**EEADR**	Endereço para ler ou gravar na EEPROM
0AH	**PCLATH**	Parte alta do PC
0BH	**INTCON**	Registro do INTCON
BANCO 1 (BSF STATUS,RP0)		
80H	**INDF**	Endereçamento indireto
81H	**OPTION_REG**	Registro OPTION_REG
82H	**PCL**	Parte baixo do PC
83H	**STATUS**	Registro de STATUS
84H	**FSR**	Ponteiro para endereçamento indireto
85H	**TRISA**	Se TRISA = 1 entrada; se TRISA = 0 saída
86H	**TRISB**	Se TRISB = 1 entrada; se TRISB = 0 saída
87H	**------**	Não implementado
88H	**EECON1**	Controle da EEPROM
89H	**EECON2**	Controle da EEPROM
8AH	**PCLATH**	Parte alta do PC
8BH	**INTCON**	Registro do INTCON

PS2	PS1	PS0	DIV. TIMER	DIV. W. DOG
0	**0**	**0**	**1: 2**	**1: 1**
0	**0**	**1**	**1: 4**	**1: 2**
0	**1**	**0**	**1: 8**	**1: 4**
0	**1**	**1**	**1: 16**	**1 : 8**
1	**0**	**0**	**1: 32**	**1: 16**
1	**0**	**1**	**1: 64**	**1: 32**
1	**1**	**0**	**1: 128**	**1: 64**
1	**1**	**1**	**1: 256**	**1: 128**

REGISTRO STATUS

Bit 7							Bit 0
IRP	**RP1**	**RP0**	**TO**	**PD**	**Z**	**DC**	**C**

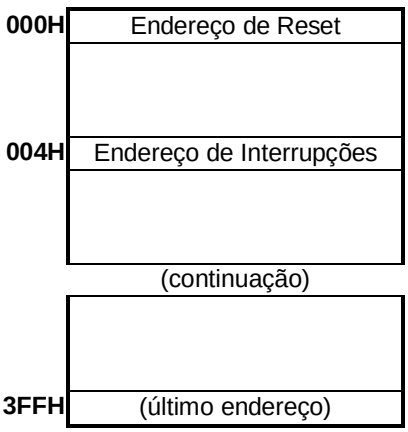
REGISTRO OPTION_REG

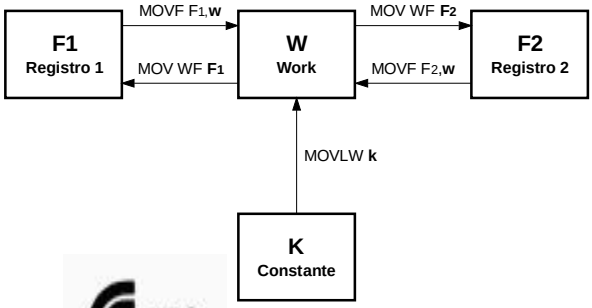
Bit 7							Bit 0
RBPU	**INTEDG**	**TOCS**	**TOSE**	**PSA**	**PS2**	**PS1**	**PS0**

REGISTRO INTCON

Bit 7							Bit 0
GIE	**EEIE**	**TOIE**	**INTE**	**RBIE**	**TOIF**	**INTF**	**RBIF**

INSTRUÇÕES DO PIC 16F84				
Operações de 1 Bit				
Operando		**Descrição**	**Ciclos**	**Afetados**
BCF	f,b	Zera o Bit b em f	1	
BSF	f,b	Seta o bit b em f	1	
BTFSC	f,b	Se b = 0, pula	1(2)	
BTFSS	f,b	Se b = 1, pula	1(2)	
Operações de Constantes e de Controles				
ADDLW	k	Soma W com k	1	C,DC,Z
ANDLW	k	AND entre W e k	1	Z
CALL	k	Chama a subrotina k	2	
CLRWDT		Zera o timer W.Dog	1	TO\,PD\
GOTO	k	Desvia para o endereço k	2	
IORLW	k	OR entre W e k	1	Z
MOVLW	k	Carrega W com k	1	
RETFIE		Retorna Interrupção (GIE=1)	2	
RETLW	k	Retorna com W=k	2	
RETURN	k	Retorna de Subrotina	2	
SLEEP		Entra modo SLEEP	1	TO\,PD\
SUBLW	k	Subtrai k de W	1	C,DC,Z
XORLW	k	XOR entre W e k	1	Z
Operações de 1 Byte				
ADDWF	f,d	Soma W e f	1	C,DC,Z
ANDWF	f,d	AND entre W e f	1	Z
CLRF	f	Zera f	1	Z
CLRW		Zera W	1	Z
COMF	f,d	Complementa f	1	Z
DECF	f,d	Decrementa f	1	Z
DECFSZ	f,d	Decrementa f, pula se f=0	1(2)	
INCF	f,d	Incrementa f	1	Z
INCFSZ	f,d	Incrementa f, pula se f=0	1(2)	
IORWF	f,d	OR entre W e f	1	Z
MOVF	f,d	Move o f para d	1	Z
MOVWF	f	Move o W para o f	1	
NOP		Não executa nada	1	
RLF	f,d	Roda à esquerda p/ Carry	1	C
RRF	f,d	Roda à direita p/ Carry	1	C
SUBWF	f,d	Subtrai W de f	1	C,DC,Z
SWAPF	f,d	Troca os NIBLES em f	1	
XORWF	f,d	XOR entre W e f	1	Z





FUNDAÇÃO INSTITUTO TECNOLÓGICO DE OSASCO - FITO

Prof. *Paulo Takashi Miyatake*