

FÁBIO PEREIRA

**Baseado no PIC 16F627 e 16F628**

**Conceitos básicos**

**Programação**

**Técnicas de hardware e de software**

**Projetos e exemplos**

**MICROCONTROLADORES**

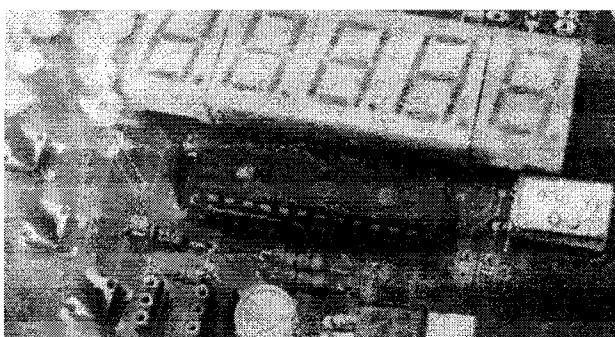
**PIC**

**TÉCNICAS AVANÇADAS**



**BRINDE**

Código-fonte dos  
programas e exemplos  
do livro disponíveis  
na INTERNET.



**MICROCONTROLADORES**

# PIC

**TÉCNICAS AVANÇADAS**



**DENTAL MÉDICA EDITORA E LIVRARIA LTDA.**

**LIVROS EM GERAL TÉCNICOS E CIENTÍFICOS**

AL. BARÃO DE LIMEIRA, 89 - S/ 407 - CEP 01202-001  
STA. EFIGÊNIA - SP - TEL.: (11) 221-1339 - FAX: (11) 221-5718  
E-MAIL: dentalmed@uol.com.br

## ***Seja Nossa Parceiro no Combate à Cópia Ilegal***

A cópia ilegal é crime. Ao efetuá-la, o infrator estará cometendo um grave erro, que é inibir a produção de obras literárias, prejudicando profissionais que serão atingidos pelo crime praticado.

Junte-se a nós nesta corrente contra a pirataria. Diga não à cópia ilegal.

## ***Seu Cadastro é muito Importante para Nós***

Ao preencher e remeter a ficha de cadastro constante no final desta publicação, cuja postagem será paga pela Editora Érica, bastando depositá-la em qualquer caixa de correio, você passará a receber, automaticamente, informações sobre nossos lançamentos em sua área de preferência.

Conhecendo melhor nossos leitores e suas preferências, vamos produzir títulos que atendam suas necessidades.

Obrigado pela sua escolha.

## ***Fale Conosco!***

Eventuais problemas referentes ao conteúdo deste livro serão encaminhados ao(s) respectivo(s) autor(es) para esclarecimento, excetuando-se as dúvidas que dizem respeito a pacotes de softwares, as quais sugerimos que sejam encaminhadas aos distribuidores e revendedores desses produtos, que estão habilitados a prestar todos os esclarecimentos.

Os problemas só podem ser enviados por:

- 1) E-mail: [producao@erica.com.br](mailto:producao@erica.com.br)
- 2) Fax: (11) 217.4060
- 3) Carta: Rua São Gil, 159 - Tatuapé - CEP 03401-030 - São Paulo - SP





Ano: 2005 2004 2003 **2002**

Edição: 9 8 7 6 5 4 3 2 1

**Editora Érica Ltda.**

**Conselho Editorial:**

Diretor Editorial:

*Antonio Marco Vicari Cipelli*

Diretor Comercial:

*Paulo Roberto Alves*

Diretor de Publicidade:

*Waldir João Sandrini*

Capa:

*Maurício S. de França*

Revisão Interna:

*Graziela Gonçalves De Filippis*

Revisão Gramatical:

*Marlene Teresa Santin Alves*

Editoração:

*Rosana Ap. A. Santos*

Desenhos:

*Pedro Paulo Vieira Herruzzo*

Avaliação Técnica:

*Flávio Eugenio de Lima*

Coordenação e Revisão:

*William Pereira Alves*

*Rosana Arruda da Silva*

Copyright © 2002 da Editora Érica Ltda.

Dados Internacionais de Catalogação na Publicação (CIP)  
(Câmara Brasileira do Livro, SP, Brasil)

Pereira, Fábio  
Microcontroladores PIC: Técnicas Avançadas / Fábio Pereira. - - São Paulo : Érica,  
2002.

Bibliografia.

ISBN: 85-7194-727-9

1. Microcontroladores I. Título.

02-1689

CDD-621.3916

### Índices para Catálogo Sistemático:

1. Microcontroladores PIC: Engenharia eletrônica 621.3916

**Todos os direitos reservados.** Proibida a reprodução total ou parcial, por qualquer meio ou processo, especialmente por sistemas gráficos, microfilmicos, fotográficos, reprográficos, fonográficos, video-gráficos, internet, e-books. Vedada a memorização e/ou recuperação total ou parcial em qualquer sistema de processamento de dados e a inclusão de qualquer parte da obra em qualquer programa juscibernético. Essas proibições aplicam-se também às características gráficas da obra e à sua editoração. A violação dos direitos autorais é punível como crime (art. 184 e parágrafos, do Código Penal, cf. Lei nº 6.895, de 17.12.80) com pena de prisão e multa, conjuntamente com busca e apreensão e indenizações diversas (artigos 102, 103 parágrafo único, 104, 105, 106 e 107 itens 1, 2 e 3 da Lei nº 9.610, de 19/06/98, Lei dos Direitos Autorais).

*O Autor e a Editora acreditam que todas as informações aqui apresentadas estão corretas e podem ser utilizadas para qualquer fim legal. Entretanto, não existe qualquer garantia, explícita ou implícita, de que o uso de tais informações conduzirá sempre ao resultado desejado. Os nomes de sites e empresas, porventura mencionados, foram utilizados apenas para ilustrar os exemplos, não tendo vínculo algum com o livro, não garantindo a sua existência nem divulgação. Eventuais erratas estarão disponíveis no site da Editora Érica para download.*

"Algumas imagens utilizadas neste livro foram obtidas a partir do CorelDRAW 7, 8 e 9 e da Coleção do MasterClips/ MasterPhotos© da IMSI, 1985 Francisco Blvd. East, San Rafael, CA 94901-5506, USA."



**Editora Érica Ltda.**

Rua São Gil, 159 - Tatuapé

CEP: 03401-030 - São Paulo - SP

Fone: (11) 295-3066 - Fax: (11) 217-4060

Site: [www.editoraerica.com.br](http://www.editoraerica.com.br)

## **Dedicatória**

À minha esposa Josi, aos meus pais Pedro e Cristina e à minha avó Juçá.

Se alguém julga saber alguma coisa, com efeito, não  
aprendeu ainda como convém saber.

1 Co 8:2

## Agradecimentos

Agradeço a todos os que acreditaram e me deram forças para continuar nos momentos de fraqueza e desânimo.

Em especial:

À minha esposa Josi pelas incontáveis horas que suportou para que esta obra viesse à tona;

Aos meus pais, Pedro e Cristina, pelo incentivo nos estudos da eletrônica;

Aos meus irmãos Luis Fernando e Cristiane;

A minha avó Juçá pelo convívio durante o curso técnico em Curitiba;

Ao meu tio Jorge por me abrigar em sua casa para que eu concluisse meu curso técnico;

Aos amigos que fiz durante os estudos no CEFET-PR: Fábio, Cristian, Gian Carlo, Líber, Daniel, Rodrigo, e tantos outros;

Ao meu amigo e sócio Giovanni por relevar os atrasos nos nossos projetos em andamento;

À Rosana Arruda, da Editora Érica, pelas dicas que forneceu durante a escrita deste livro;

À Cleide Donice da Hitech e André Rabner da Artimar pelas amostras e material fornecidos que permitiram o desenvolvimento deste livro;

Aos outros amigos, parentes e alunos do CTA que contribuíram direta ou indiretamente para que esta obra pudesse ser realizada.

A todos o meu muito obrigado!

## **Sobre o Autor**

Fábio Pereira é formado em Direito e Eletrônica.

Desde 1992 dedica grande parte do seu tempo, atuando na área de microprocessadores e microcontroladores desde 1992, tendo desenvolvido ou participado do desenvolvimento de diversos equipamentos com microprocessadores e microcontroladores, como: Z-80, 6502, 8086, 80386 (software), 8051, 89C8252, PICs (12, 14 e 16 bits).

Além do hardware, já desenvolveu diversos softwares em linguagens, como: ASSEMBLY, BASIC, PASCAL, C e DELPHI.

Atualmente o autor dedica-se a projetos na área de eletrônica e instrumentação automotiva e leciona também a disciplina de Microprocessadores no Centro Técnico de Automação - CTA em Joinville, Santa Catarina.

## **Sobre o Material Disponível na Internet**

O material disponível na Internet contém o código-fonte de todos os programas e exemplos do livro. Para utilizar esses arquivos é imprescindível que o leitor possua instalado em sua máquina o MPLAB.

Arquivo	Tamanho
Microcontroladores(pic).EXE	61 KB

### **Procedimento para Download**

Acesse o site da Editora Érica Ltda.: [www.editoraerica.com.br](http://www.editoraerica.com.br). A transferência do arquivo disponível pode ser feita de duas formas:

- 1) Por meio do módulo pesquisa.** Localize o livro desejado, digitando palavras-chaves (nome do livro ou do autor). Aparecerão os dados do livro e o arquivo para download, então dê um clique sobre o arquivo executável que será transferido.
- 2) Por meio do botão “Download”.** Na página principal do site, clique no item “Download”. Será exibido um campo, no qual devem ser digitadas palavras-chaves (nome do livro ou do autor). Serão exibidos o nome do livro e o arquivo para download. Dê um clique sobre o arquivo executável que será transferido.

### **Procedimento para Descompactação**

*Primeiro passo:* após ter transferido o arquivo, verifique o diretório em que se encontra e dê um duplo-clique sobre ele. Será exibida uma tela do programa WINZIP SELF-EXTRACTOR que conduzirá você ao processo de descompactação. Abaixo do Unzip To Folder, existe um campo que indica o destino dos arquivos que serão copiados para o disco rígido do seu computador.

C:\Microcontroladores\_PIC

*Segundo passo:* prossiga com a instalação, clicando no botão Unzip, o qual se encarrega de descompactar os arquivos. Logo abaixo dessa tela, aparece a barra de status a qual monitora o processo para que você acompanhe. Após o término, outra tela de informação surgirá, indicando que os arquivos foram descompactados com sucesso e estão no diretório criado. Para sair dessa tela, clique no botão OK. Para finalizar o programa WINZIP SELF-EXTRACTOR, clique no botão Close.

## Prefácio

Este livro destina-se aos profissionais e entusiastas da eletrônica que desejam ampliar seus conhecimentos sobre os microcontroladores PIC.

Fruto de uma longa experiência na programação e desenvolvimento de sistemas microprocessados e microcontrolados, este livro surgiu como uma forma de condensar tais conhecimentos em um único material de consulta.

Com uma linguagem clara e acessível, são vistos todos os detalhes de funcionamento dos PICs, desde a arquitetura interna, periféricos, memória e registradores, até os detalhes de funcionamento, técnicas de hardware e programação.

Na parte final do livro são apresentadas diversas rotinas e macros, formando um apanhado condensado das mais importantes técnicas necessárias ao desenvolvimento de aplicações com microcontroladores, como: operações matemáticas simples e complexas, conversão de bases numéricas, interface com displays, teclados, etc.

O resultado final é uma obra com um vasto material de pesquisa e consulta, capaz de atender desde o iniciante até os profissionais envolvidos nos mais complexos projetos com microcontroladores.

# **Índice Analítico**

<b>Capítulo 1 - Introdução .....</b>	17
<b>Capítulo 2 - Conceitos e Circuitos Digitais .....</b>	21
2.1 - Função AND (E).....	21
2.2 - Função OR (OU).....	22
2.3 - Função XOR (OU exclusivo).....	22
2.4 - Função NO (NÃO) .....	23
2.5 - Buffers .....	23
2.6 - Flip-flops .....	26
2.7 - Decodificadores/Demultiplexadores .....	30
2.8 - Registradores de Deslocamento .....	33
<b>Capítulo 3 - Microcontroladores PIC .....</b>	35
3.1 - Memória de Programa .....	37
3.2 - Memória RAM e Registradores .....	39
3.2.1 - Registrador W .....	41
3.2.2 - Contador de Programa - PC .....	41
3.2.3 - Pilha ( <i>Stack</i> ) .....	44
3.3 - Interrupções .....	46
3.3.1 - Tipos de Interrupção .....	50
3.3.1.1 - INTF - Interrupção Externa .....	51
3.3.1.2 - RBIF - Interrupção por Mudança de Estado no PortB .....	51
3.3.1.3 - EEIF - Interrupção de Escrita na EEPROM .....	51
3.3.1.4 - T0IF - Interrupção do Timer 0 .....	52
3.3.1.5 - TMR1IF - Interrupção do Timer 1 .....	52
3.3.1.6 - TMR2IF - Interrupção do Timer 2 .....	52
3.3.1.7 - CCP1IF - Interrupção do Módulo de Captura/ Comparação/ PWM .....	52
3.3.1.8 - TXIF - Interrupção de Transmissão da USART .....	53
3.3.1.9 - RCIF - Interrupção de Recepção da USART .....	53
3.3.1.10 - CMIF - Interrupção do Módulo Comparador Analógico .....	53
3.3.1.11 - CCP2IF - Interrupção do Módulo de Captura/Comparação/ PWM .....	53
3.3.1.12 - SSPIF - Interrupção do Módulo Serial Síncrono .....	54
3.3.1.13 - ADIF - Interrupção do Conversor Analógico/Digital.....	54
3.3.1.14 - ADCIF - Interrupção do Conversor Analógico/Digital do Tipo SLOPE .....	54
3.3.1.15 - OVFIF - Interrupção de Overflow do Timer do Conversor A/D SLOPE .....	54

3.3.1.16 - PSPIF - Interrupção de Escrita/Leitura na Porta Escrava Paralela .....	54
3.3.1.17 - LCDIF - Interrupção do LCD .....	55
3.3.1.18 - GPIF - Interrupção por Mudança de Estado no GPIO .....	55
3.3.2 - Latência de Interrupções .....	55
3.3.3 - Eventos de Interrupção e o Modo SLEEP .....	56
3.3.3.1 - Interrupções Globais Habilitadas (GIE='1') .....	56
3.3.3.2 - Interrupções Globais Desabilitadas (GIE='0') .....	56
<b>Capítulo 4 - Instruções .....</b>	<b>57</b>
4.1 - Manipulação de Registradores .....	59
4.2 - Aritmética .....	70
4.3 - Operações Lógicas .....	77
4.4 - Desvio .....	88
4.5 - Controle .....	96
<b>Capítulo 5 - PIC 16F627/628 .....</b>	<b>99</b>
5.1 - Descrição dos Pinos .....	100
5.2 - Estrutura Interna .....	101
5.3 - Registradores .....	102
5.3.1 - Registrador STATUS (0x03, 0x83, 0x103 e 0x183) .....	105
5.3.2 - Registrador OPTION_REG (0x81 e 0x181) .....	109
5.3.3 - Registrador PCON (0x8E) .....	110
5.4 - Interrupções .....	111
5.4.1 - Registrador INTCON (0x0B) .....	113
5.4.2 - Registrador PIE1 (0x8C) .....	114
5.4.3 - Registrador PIR1 (0x0C) .....	115
5.5 - Pinos de I/O .....	117
5.5.1 - Porta A .....	117
5.5.1.1 - Registrador PORTA (0x05) .....	118
5.5.1.2 - Registrador TRISA (0x85) .....	118
5.5.1.3 - Configuração Interna .....	119
5.5.1.4 - Observações sobre a Porta A .....	122
5.5.2 - Porta B .....	123
5.5.2.1 - Registrador PORTB (0x06 e 0x106) .....	124
5.5.2.2 - Registrador TRISB (0x86 e 0x186) .....	124
5.5.2.3 - Configuração Interna .....	125
5.5.2.4 - Observações sobre a Porta B .....	130
5.6 - Contador/Temporizador Timer 0 .....	131
5.6.1 - Observações sobre o Timer 0 .....	133
5.6.2 - Exemplos de Uso do Timer 0 .....	134
5.6.2.1 - Clock Interno .....	134

5.6.2.2 - Clock Externo .....	135
5.6.2.3 - Clock Externo Sem Prescaler .....	135
5.6.2.4 - Alterando o Fator de Divisão do Timer 0 .....	135
5.7 - Contador/Temporizador Timer 1 .....	136
5.7.1 - Registrador T1CON (0x10) .....	138
5.7.2 - Oscilador do Timer 1 .....	140
5.7.3 - Observações sobre o Timer 1 .....	140
5.7.4 - Exemplos de Programação do Timer 1 .....	141
5.7.4.1 - Clock Interno e Prescaler 1:1 .....	141
5.7.4.2 - Clock Externo em Modo Síncrono .....	142
5.7.4.3 - Clock Externo em Modo Assíncrono .....	142
5.7.4.4 - Clock Externo em Modo Assíncrono e Oscilador Externo Ligado .....	142
5.7.4.5 - Alterando o Fator de Divisão do Timer 1 .....	143
5.8 - Temporizador Timer 2 .....	143
5.8.1 - Registrador TMR2CON (12h) .....	145
5.8.2 - Observações sobre o Timer 2 .....	146
5.8.3 - Exemplo de Configuração do Timer 2 .....	146
5.9 - Módulo CCP .....	147
5.9.1 - Registrador CCP1CON (0x17) .....	147
5.9.2 - Registrador CCPR1L (0x15) .....	148
5.9.3 - Registrador CCPR1H (0x16) .....	148
5.9.4 - Modo de Captura .....	148
5.9.4.1 - Observações sobre o Modo de Captura .....	150
5.9.4.2 - Exemplo de Programação do CCP no Modo Captura .....	151
5.9.5 - Modo de Comparação .....	152
5.9.5.1 - Observações sobre o CCP em Modo de Comparação .....	153
5.9.5.2 - Exemplo de Configuração para Modo de Comparação .....	154
5.9.6 - Modo PWM .....	154
5.9.6.1 - Exemplo de Configuração do CCP em modo PWM .....	156
5.10 - Módulo Comparador Analógico .....	157
5.10.1 - Registrador CMCON (0x1F) .....	157
5.10.2 - Modo 0 - Comparadores em Reset .....	159
5.10.3 - Modo 1 - Dois Comparadores com Três Entradas Multiplexadas .....	159
5.10.4 - Modo 2 - Dois Comparadores com Quatro Entradas Multiplexadas .....	160
5.10.5 - Modo 3 - Dois Comparadores com uma Referência Externa Comum .....	160
5.10.6 - Modo 4 - Dois Comparadores Independentes .....	161
5.10.7 - Modo 5 - Um Comparador .....	161
5.10.8 - Modo 6 - Dois Comparadores com Referência Comum e Saídas Externas .....	162
5.10.9 - Modo 7 - Comparadores Desligados .....	162
5.10.10 - Observações sobre o Módulo Comparador .....	163
5.10.11 - Exemplo de Configuração do Comparador Analógico .....	163

5.11 - Referência Interna de Tensão.....	164
5.11.1 - Registrador VRCON (0x9F).....	164
5.11.2 - Observações sobre a Referência Interna de Tensão .....	166
5.11.3 - Exemplo de Configuração da Referência Interna .....	166
5.12 - USART .....	167
5.12.1 - Estrutura de Transmissão da USART.....	168
5.12.2 - Estrutura de Recepção da USART .....	169
5.12.3 - Registrador TXSTA (98h).....	170
5.12.4 - Registrador RCSTA (18h) .....	171
5.12.5 - Registrador SPBRG (0x99).....	173
5.12.6 - Transmissão no Modo Assíncrono .....	174
5.12.6.1 - Observações sobre a Transmissão Assíncrona .....	175
5.12.7 - Recepção no Modo Assíncrono .....	175
5.12.7.1 - Observações sobre a Recepção Assíncrona .....	176
5.12.7.2 - Recepção no modo Assíncrono com Endereçamento.....	177
5.12.8 - Transmissão Síncrona Modo Mestre.....	179
5.12.8.1 - Observações sobre o Modo Mestre Síncrono.....	180
5.12.9 - Recepção Síncrona em Modo Mestre .....	180
5.12.9.1 - Observações sobre Recepção Síncrona em Modo Mestre .....	181
5.12.10 - Transmissão Síncrona Modo Escravo .....	181
5.12.11 - Recepção Síncrona em Modo Escravo.....	182
5.13 - EEPROM Interna .....	183
5.13.1 - Registrador EEDATA (0x9A).....	183
5.13.2 - Registrador EEADDR (0x9B) .....	183
5.13.3 - Registrador EECON1 (0x9C).....	184
5.13.4 - Registrador EECON2 (0x9D).....	185
5.13.5 - Exemplos de Acesso à Memória EEPROM Interna .....	185
5.14 - Palavra de Configuração .....	186
5.15 - Circuito de Clock .....	189
5.15.1 - Cristal/Ressonador .....	189
5.15.2 - Oscilador com Resistor Externo .....	190
5.15.3 - Oscilador Ínterno de 4Mhz .....	190
5.15.4 - Clock Externo .....	191
5.16 - Temporizador de Power-Up.....	191
5.17 - Detector de Brown-Out .....	191
5.18 - Watchdog.....	192
5.19 - Modo SLEEP .....	193
5.20 - Reset .....	194
5.21 - ICSP .....	197
<b>Capítulo 6 - Linguagens e Ambientes de Programação.....</b>	<b>203</b>
<b>6.1 - MPLAB.....</b>	<b>203</b>

6.2 - Criação de um Projeto .....	207
6.3 - Compilação e Simulação de um Programa .....	210
6.3.1 - Adição de Janelas de Observação .....	211
6.3.2 - Cálculo do Tempo de Execução .....	211
6.3.3 - Linguagem Assembly MPASM .....	212
6.3.4 - IC-Prog .....	215
6.3.5 - Configuração do IC-PROG .....	216
6.4 - Outros Pacotes de Programação .....	218
6.5 - Projeto Passo a Passo com PIC .....	220
 <b>Capítulo 7 - Técnicas de Programação .....</b>	 225
7.1 - Princípios Básicos .....	225
7.1.1 - Atribuição de Valores .....	225
7.1.2 - Utilização dos Registradores GPR .....	226
7.1.3 - Comparação de Valores .....	226
7.1.3.1 - XORWF .....	227
7.1.3.2 - SUBWF .....	227
7.2 - Configurando as Portas de I/O .....	228
7.2.1. Considerações sobre Operações de I/O .....	230
7.3 - Sub-rotinas .....	232
7.4 - Interrupções .....	235
7.4.1 - Desativação do GIE .....	235
7.4.2 - Tratamento de Interrupções .....	237
7.4.3 - Sub-rotinas em Interrupções .....	239
7.5 - Matemática no PIC .....	240
7.5.1 - Adição .....	240
7.5.1.1 - Adição de 16 bits .....	242
7.5.2 - Subtração .....	243
7.5.2.1 - Subtração de 16 bits .....	245
7.5.3 - Multiplicação .....	247
7.5.3.1 - Multiplicação Inteira com RLF .....	247
7.5.3.2 - Multiplicação Didática 8x8 bits .....	248
7.5.3.3 - Multiplicação Inteira 8x8 bits .....	249
7.5.3.4 - Multiplicação Inteira 16x16 bits .....	250
7.5.4 - Divisão .....	252
7.5.4.1 - Divisão Inteira com RRF .....	252
7.5.4.2 - Divisão Didática 8/8 bits .....	253
7.5.4.3 - Divisão Inteira 8/8 bits .....	254
7.5.5 - Raiz Quadrada Inteira .....	256
7.5.6 - Resolução de Equações Simples .....	258
7.6 - Conversão Binário/Hexadecimal/Decimal .....	260
7.6.1 - Conversão Binário em Decimal (8 bits) .....	260

7.6.2 - Conversão Decimal em Binário (8 bits) .....	261
7.6.3 - Conversão Binário em Decimal (16 bits) .....	261
7.6.4 - Conversão Decimal em Binário (16 bits) .....	263
7.7 - Tabelas .....	265
7.8 - Acesso Indireto à Memória.....	268
7.9 - Configurando os Timers .....	269
7.9.1 - Configurando o Timer 0 .....	269
7.9.2 - Configurando o Timer 1 .....	270
7.9.3 - Configurando o Timer 2 .....	271
7.10 - Medindo a Largura de um Pulso .....	272
7.11 - Conversor A/D Delta-Sigma .....	274
7.12 - Interface com Displays.....	276
7.12.1 - Interface de Display 7 Segmentos com 4511 .....	276
7.12.2 - Interface de Display 7 Segmentos Decodificada por Software.....	278
7.12.3 - Display Multiplexado de 7 Segmentos .....	280
7.13 - Leitura de Teclado.....	283
7.14 - Detector de Paridade .....	287
7.15 - Expandindo o Conjunto de Instruções PIC.....	288
7.15.1 - Operações com o Registrador W .....	288
7.15.2 - Desvio Condicional.....	291
7.15.3 - Outras Funções .....	295
7.16 - Convertendo 16F84 em 16F62x.....	296
 <b>Capítulo 8 - Projetos.....</b>	 297
8.1 - Voltímetro Digital Simples.....	297
8.2 - Relógio Digital com Alarme .....	301
8.3 - Freqüencímetro/Contador de 7 Dígitos .....	309
 <b>Apêndice A - Bancos de Memória .....</b>	 317
<b>Apêndice B - Tabelas de Baud Rate.....</b>	321
<b>Apêndice C - Conjunto de Instruções 14 Bits .....</b>	325
<b>Apêndice D - Diretivas do MPASM .....</b>	327
<b>Apêndice E - Pinagem dos PICs .....</b>	345
<b>Apêndice F - Programador JDM .....</b>	347
<b>Apêndice G - Tabelas de Conversão .....</b>	349
<b>Índice Remissivo .....</b>	353

## Introdução

Para que possamos entender o funcionamento de um sistema com microprocessadores, é imperioso conhecer os conceitos básicos sobre computação e programação.

As primeiras máquinas programáveis foram teares (para produção de tecidos). Tais máquinas podem ser consideradas as precursoras dos modernos computadores binários, pois constituíam-se também de um sistema binário, e eram programadas por meio de cartões perfurados, que indicavam a seqüência de trabalho da máquina e assim o padrão tecido.

Após os teares, surgiram as primeiras máquinas de calcular mecânicas e posteriormente as eletromecânicas. Tais máquinas eram grandes e extremamente limitadas e a princípio não são consideradas como verdadeiros dispositivos programáveis. No entanto, elas constituíram um degrau importante para o desenvolvimento dos computadores eletrônicos que viriam a ser desenvolvidos posteriormente.

Mais tarde, com o desenvolvimento da eletrônica surgiram os primeiros computadores eletrônicos, como o ENIAC, que utilizava aproximadamente 18.000 válvulas eletrônicas, pesava 30 toneladas e consumia aproximadamente 150 KW.

Com o surgimento do transistor, a computação teve um grande avanço, pois os computadores poderiam ser construídos de forma menos dispendiosa e ocupando menos espaço, aumentando também sua velocidade e diminuindo o consumo e a dissipação de potência.

Com o avanço da eletrônica digital, no final dos anos 60, os grandes fabricantes de semicondutores passaram a idealizar dispositivos integrados que pudessem ser programados para executar uma seqüência qualquer de instruções, ou seja, executar um programa.

Até então, os computadores digitais eram grandes e dispendiosos, pois suas CPU (acrônimo inglês para **C**entral **P**rocessing **U**nit ou Unidade Central de Processamento) eram construídas com base em CI's digitais discretos e transistores.

Em 1969, a INTEL lançou o primeiro microprocessador digital do mercado: o i4004 de 4 bits, 46 instruções e clock de 740 khz. Ele era o resultado da integração de partes básicas de uma CPU em um chip único com mais de 2000 transistores!

A partir do i4004, teve início uma grande revolução na área da eletrônica, pois agora era mais fácil implementar dispositivos eletrônicos "inteligentes", como computadores, calculadoras, etc.

Após o i4004, surgiu o i4040, uma versão aprimorada do i 4004 com o dobro da capacidade de endereçamento de memória (8 k) e 60 instruções.

Mais algum tempo, e surgia o primeiro microprocessador de 8 bits: o i8008, com 16 k de capacidade de memória, 48 instruções e clock de até 800khz!

O i8008 foi o precursor da família mais importante na história mundial dos microprocessadores: o Intel 8080A, o primeiro microprocessador a implementar um barramento de endereços de 16 bits (64k de memória), clock de 3Mhz e 78 instruções. Com ele surgiram inúmeros Cl's periféricos de suporte, como o 8212, 8224 e 8228.

Foi com base na arquitetura do 8080 que surgiram inúmeras famílias de MPU's (acrônimo inglês para **Micro Processor Unit**, ou Unidade de Microprocessador), como: 8085, 8086 (utilizado nos primeiros PC's da IBM em 1980), o Z80 da Zilog e outras. Também nessa época surgiram famílias como o 6800 da Motorola, o 6502 da Mostek (utilizado no Apple II) dentre outros.

As aplicações para microprocessadores crescem a cada dia, assim como a necessidade de miniaturização e diminuição de custos. No entanto, para o seu funcionamento, os microprocessadores necessitam de outros componentes externos, como memória RAM (para o armazenamento de dados e variáveis), memória ROM (para armazenamento do programa), Cl's geradores de clock, endereçamento, dentre outros.

Então, em meados da década de 80, surgiram os primeiros MCU's (**Micro Controler Unit**, ou Unidade de Microcontrolador). Os microcontroladores, ao contrário de seus irmãos microprocessadores, são dispositivos mais simples, com memórias RAM e ROM internas, oscilador interno de clock, I/O interno, entre outros, sendo por isso chamados muitas vezes de computadores em um único chip. Tais características tornam mais simples o projeto de dispositivos inteligentes, pois os MCU's raramente necessitam de Cl's externos para funcionar, o que contribui para diminuição de custos e tamanho.

Os primeiros MCU's também eram derivados do 8080, como, por exemplo, o 8048 e posteriormente o 8051, todos da Intel. Este último tornou-se o MCU mais utilizado até hoje em tarefas de automação.

Hoje em dia, o mercado para MCU's é cada vez maior, existindo uma grande variedade de tipos, como, por exemplo, os da família 8051 (conhecidos como MCS-51), PIC, ST, COP, AVR, etc.

Antes de iniciarmos nossos estudos, vejamos alguns conceitos básicos sobre programação e dispositivos programáveis.

Basicamente, programar é fornecer uma seqüência determinada de comandos ou instruções para ser executada por uma pessoa ou máquina.

Em uma linha de produção de uma empresa, por exemplo, seus empregados seguem programas predeterminados do tipo pegue a peça da máquina A, coloque-a na

esteira, retire a peça da máquina B, coloque-a na máquina A, e assim por diante. Esta seqüência de instruções constitui-se num programa.

Em relação aos computadores, não é diferente. Também podemos fornecer uma seqüência determinada de instruções para ser executada pela máquina.

No entanto, nos computadores a quantidade ou diversidade de instruções que a máquina pode entender é bastante reduzida. Por isso devemos estar familiarizados com as instruções do microprocessador para podermos programá-lo.

Mas por que necessitariamos de um dispositivo programável?

A resposta é muito simples: antes da invenção do microprocessador, era necessário construir complexos circuitos de controle para diversas máquinas ou sistemas, e uma vez construídos tais circuitos, tornava-se muito caro e difícil alterá-los para atender a eventuais modificações necessárias no controle das máquinas em questão.

Por isso a necessidade de utilizar dispositivos programáveis, pois com eles agora é possível controlar a máquina a partir de um programa, e este pode ser facilmente alterado para a atender quaisquer necessidades que possam aparecer.

## Conceitos e Circuitos Digitais

A base da eletrônica digital está nas portas lógicas AND, OU, XOU e inversoras.

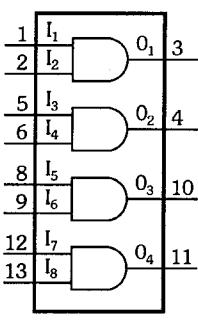
A partir da associação das portas lógicas formam-se diversos elementos da eletrônica digital, como flip-flops, decodificadores, codificadores, contadores, latches, buffers, shift registers, entre outros.

Por isso, antes de iniciarmos o estudo dos microcontroladores PIC, faremos uma pequena revisão sobre o funcionamento de tais circuitos, além de ver alguns exemplos comerciais de tais CIs.

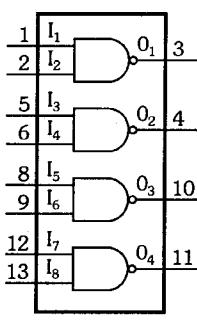
### 2.1 - Função AND (E)

*Tabela-verdade da função AND/NAND*

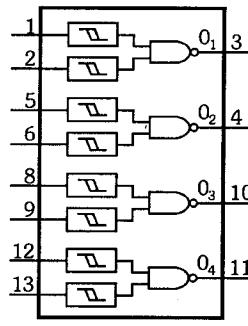
A	B	S (AND)	S (NAND)
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	0



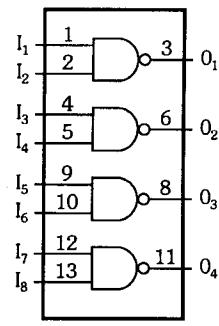
4081 AND CMOS



4011 NAND CMOS



4093 NAND CMOS  
Schmitt Trigger



74x00 NAND TTL

**Figura 2.1**

## 2.2 - Função OR (OU)

Tabela-verdade da função OR/NOR

A	B	S (OR)	S (NOR)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

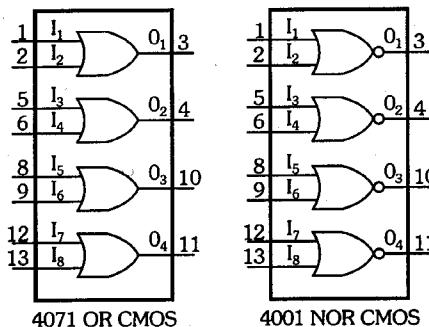


Figura 2.2

## 2.3 - Função XOR (OU exclusivo)

Tabela-verdade da função XOR/XNOR

A	B	S (XOR)	S (XNOR)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

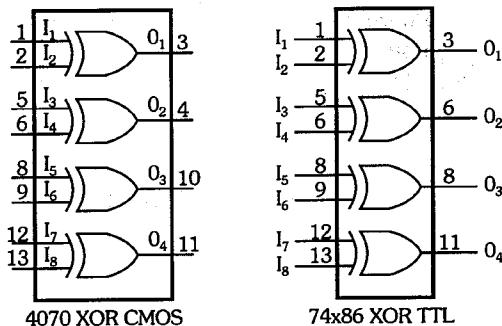
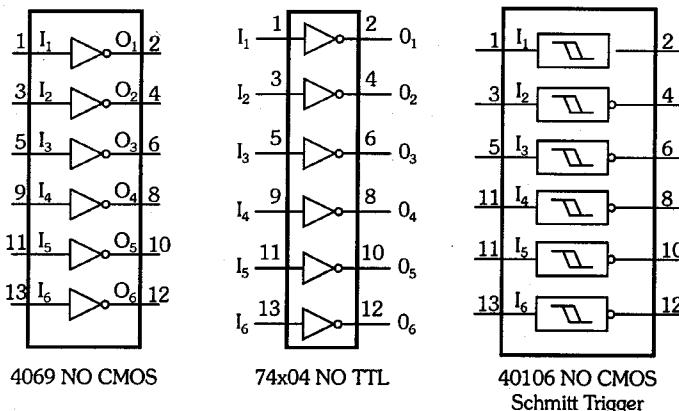


Figura 2.3

## 2.4 - Função NO (NÃO)

*Tabela-verdade da função NO*

E	S
0	1
1	0



*Figura 2.4*

## 2.5 - Buffers

Os buffers são elementos muito utilizados em eletrônica digital e possuem diversas utilidades, tais como: aumento da capacidade de corrente de saída (*fan-out*) de elementos lógicos, isolamento de circuitos, conversão de níveis, etc. Porém dentre todos, a principal utilização é realmente na ampliação da corrente de saída de elementos lógicos.

Na verdade, qualquer dispositivo eletrônico que fornece sinal a outro tem um limite de corrente de saída. Este limite, parametrizado pelos fabricantes, é normalmente chamado de *fan-out* no ramo da eletrônica digital.

Isto implica dizer que um dispositivo digital tem um limite máximo de saída e não devemos ultrapassá-lo sob pena de comprometer o nível de tensão de saída do dispositivo (devido ao excesso de corrente), o que pode provocar um funcionamento errático do circuito ou até mesmo a sua danificação permanente.

Desta forma, utilizam-se circuitos buffer para aumentar a capacidade de saída de tais dispositivos. Os buffers são normalmente utilizados em barramentos de MCUs ou MPUs e nesses casos podem ser chamados de buffers de barras, transceptores de barras ou ainda drivers de barras.

Os buffers apresentam-se em diversos tipos: inversores e não inversores, com saída em três estados ou não, unidirecionais ou bidirecionais (transceptores) ou conversores de nível (para compatibilizar tecnologias de barramento diferentes).

A seguir veremos alguns circuitos comerciais de buffers:

7406, 74xx06

**Seis portas inversoras buffer/driver de corrente, com saída em coletor aberto**

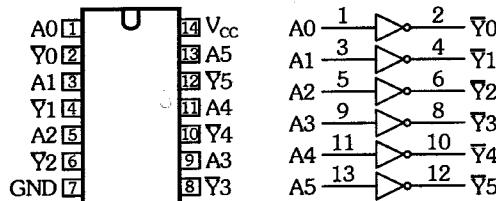


Figura 2.5

74365, 74xx365

**Seis buffers / drivers de linha com saída em três estados**

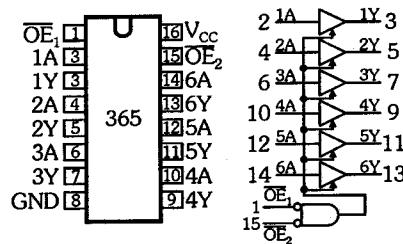


Figura 2.6

$\overline{OE}_1, \overline{OE}_2$  - Output enable - ativação das saídas (em '0' saídas ativas, em '1' saídas em alta impedância).

74240, 74xx240

**Buffer octal (8 buffers)/driver de linha, saída inversora três estados (organizado em dois grupos de 4 bits)**

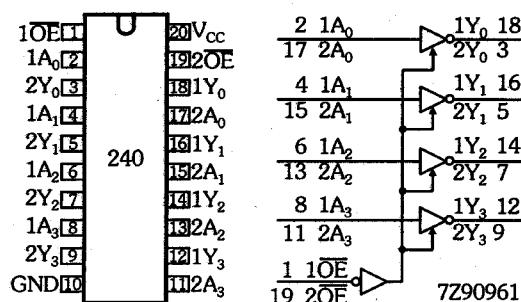


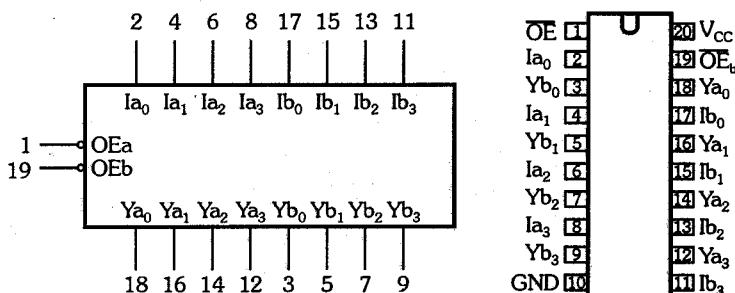
Figura 2.7

1OE - habilitação da saída do primeiro grupo de buffers (em '0' saídas ativadas, em '1' saídas em alta impedância).

2OE - habilitação da saída do segundo grupo de buffers (em '0' saídas ativadas, em '1' saídas em alta impedância).

74244, 74xx244

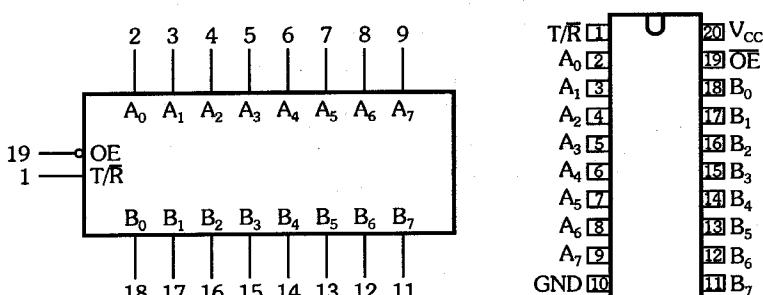
**Oito buffers com saídas em três estados (organizados em dois grupos de 4 bits)**

Ia<sub>0</sub>-Ia<sub>3</sub> - entradas Ib<sub>0</sub>-Ib<sub>3</sub> - entradasYa<sub>0</sub>-Ya<sub>3</sub> - saídas Yb<sub>0</sub>-Yb<sub>3</sub> - saídasOE<sub>a</sub> - habilita saídas "a" (ativo em nível '0')OE<sub>b</sub> - habilita saídas "b" (ativo em nível '0')

Quando OEx estiver em nível '1', as saídas permanecem em alta impedância.

Figura 2.8

Oito transceptores de barramento (buffers bidirecionais), saídas três estados



A0-A7 - entradas/saídas

B0-B7 - entradas/saídas

$\overline{OE}$  - habilita saídas (ativo em nível baixo), se em nível alto, saídas em três estados

T/R - sentido - nível '1' - de A para B, nível '0' - de B para A

Figura 2.9

## 2.6 - Flip-flops

Os flip-flops são os elementos básicos de sistemas digitais avançados, com eles são construídos contadores, registradores, registradores de deslocamento, memórias, etc.

Um flip-flop típico como o da figura 2.10 utiliza duas portas NOR. Observe que temos duas entradas, chamadas de S (set, setar) e R (reset), além de duas saídas: Q e  $\overline{Q}$ . Este tipo de flip-flop é chamado de Flip-flop SR.

A entrada S é definida como a que estando em nível lógico '1' leva a saída Q a nível lógico '1', e a entrada R é definida como a que estando em nível lógico '1' leva a saída Q a nível lógico '0'.

A tabela-verdade para o circuito da figura 2.10 é:

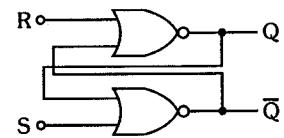


Figura 2.10

S	R	Q	$\overline{Q}$
0	0	Estado anterior	
0	1	0	1
1	0	1	0
1	1	0	0

Conforme vemos na tabela, o flip-flop SR apresenta um estado especial quando suas entradas estão desativadas (nível lógico '0'). Dizemos que neste caso o FF

memorizou o dado e vai permanecer assim até que seja alterado o estado de suas entradas.

Nas figuras 2.11 e 2.12, temos uma outra configuração para o FF SR, chamada de Flip-flop SR sincronizado. Ele tem este nome pelo fato de possuir uma terceira entrada denominada CLK (*Clock*) utilizada para sincronizar a saída com a entrada.

Desta forma, o comportamento do FF é praticamente idêntico ao do FF SR com uma diferença: se a entrada de clock estiver em nível lógico '1', as saídas do FF permanecerão no seu estado anterior. Ao contrário, se a entrada de clock estiver em nível '0', a saída do FF seguirá a tabela anterior.

Isto faz com que o FF somente sincronize suas saídas quando a entrada de clock estiver ativa.

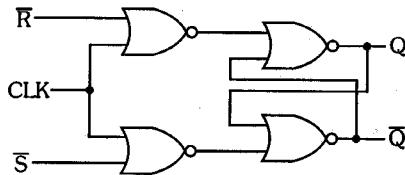


Figura 2.11

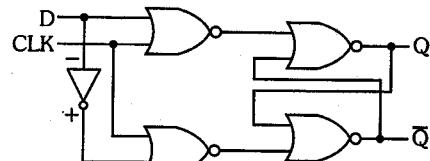


Figura 2.12

Na figura 2.12 temos uma variação do FF SR sincronizado, que é o chamado **Flip-flop “tipo D”**. Observe que as entradas deste FF sincronizado foram unidas por uma porta inversora. O FF tipo D é largamente utilizado na eletrônica digital, especialmente em circuitos como latches, memórias, dentre outros.

Neste tipo de dispositivo, o dado presente na entrada "D" é memorizado ao ativarmos o pino de clock do CI e permanecerá assim até aplicarmos um novo clock ou o dispositivo sofrer um reset.

Na figura abaixo, temos uma implementação comercial típica de um FF do tipo D com portas NAND.

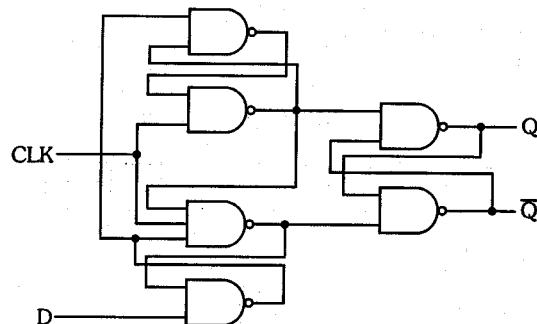
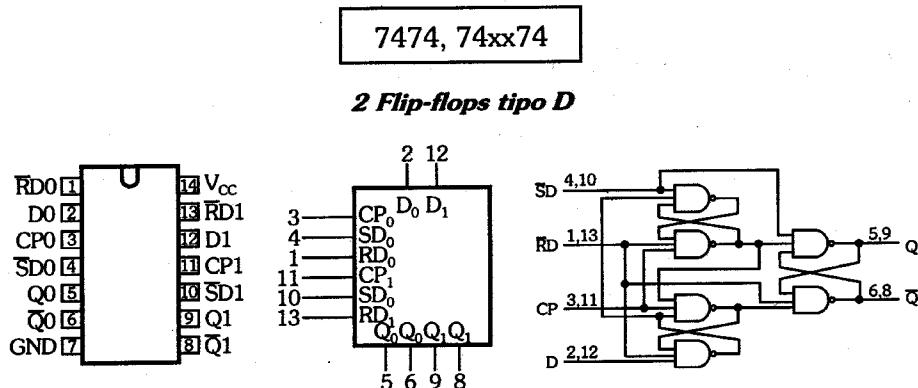


Figura 2.13

Normalmente, os FFs do tipo D comerciais são ativados por borda, ou seja, eles armazenam o dado presente na entrada D na transição do clock de '0' para '1' ou '1' para '0', dependendo do modelo.

Em seguida temos alguns exemplos de CI's Flip-flop comerciais:



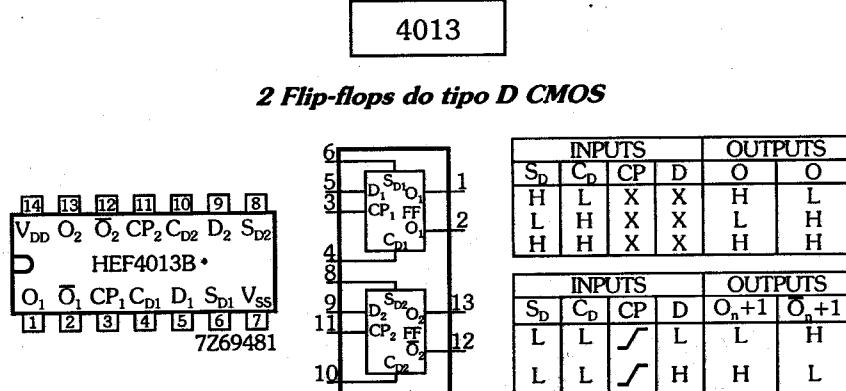
RD0/RD1 - reset ativo em nível baixo

SD0/SD1 - set ativo em nível baixo

D0/D1 - entrada de dados

CP0/CP1 - clock

**Figura 2.14**



CP<sub>0</sub>/CP<sub>1</sub> - entradas de Clock D<sub>1</sub>,D<sub>2</sub> - entradas de dados

S<sub>D1</sub>,S<sub>D2</sub> - entradas de preset (setam a saída Q em '1')

C<sub>D1</sub>,C<sub>D2</sub> - entradas de reset (resetam a saída Q (nível '0'))

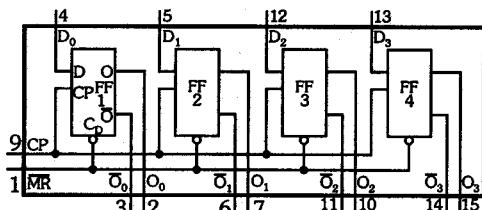
**Figura 2.15**

40175

**4 Flip-flops do tipo D CMOS**

16	15	14	13	12	11	10	9
V <sub>DD</sub>	O <sub>3</sub>	$\bar{O}_3$	D <sub>3</sub>	D <sub>2</sub>	$\bar{O}_2$	O <sub>2</sub>	CP
D	40175						
MR	O <sub>0</sub>	$\bar{O}_0$	D <sub>0</sub>	D <sub>1</sub>	$\bar{O}_1$	O <sub>1</sub>	V <sub>SS</sub>
1	2	3	4	5	6	7	8

7273707

D<sub>0</sub>, D<sub>1</sub>, D<sub>2</sub>, D<sub>3</sub> - entrada de dados

CP - clock geral para os quatro FF

MR - Master Reset - reset geral para os quatro FF, ativo em '0'

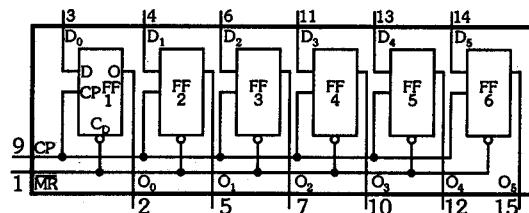
**Figura 2.16**

40174

**6 Flip-flops tipo D CMOS**

16	15	14	13	12	11	10	9
V <sub>DD</sub>	O <sub>5</sub>	D <sub>5</sub>	D <sub>4</sub>	O <sub>4</sub>	D <sub>3</sub>	O <sub>3</sub>	CP
D	40175						
MR	O <sub>0</sub>	D <sub>0</sub>	D <sub>1</sub>	O <sub>1</sub>	D <sub>2</sub>	O <sub>2</sub>	V <sub>SS</sub>
1	2	3	4	5	6	7	8

7273705



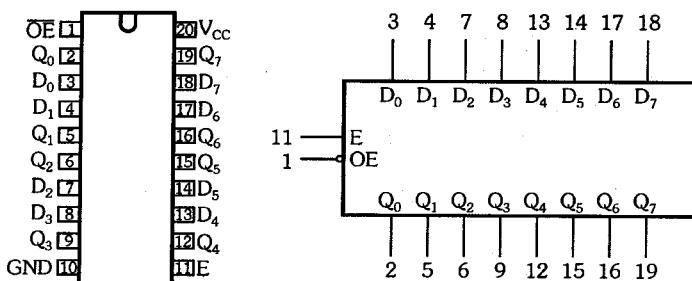
CP - Clock geral para os 6 FFs

MR - Master Reset - reset geral para todos os FF, ativo em '0'

**Figura 2.17**

Como já dissemos, uma das aplicações dos FFs são os latches ou registradores. Na verdade, um latch consiste num banco de FFs do tipo D cujos clocks estão interligados e são acionados simultaneamente. Assim, um único pulso de clock pode armazenar 4, 8, 16 ou 32 bits de dados de uma só vez. As aplicações dos latches são principalmente relacionadas a dispositivos com microcontroladores ou microprocessadores, em que os latches atuam como elementos de armazenamento temporário para dados ou endereços.

A seguir temos alguns exemplos de latches que possuem outra característica muito importante em dispositivos que devem ser ligados a barramentos, que são as saídas em três estados.

*Oito latches transparentes com saídas em três estados*

D0-D7 - entrada de dados

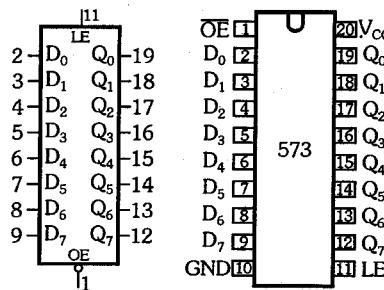
Q0-Q7 - saídas tri-state de dados

OE - entrada de habilitação das saídas (ativa em '0', em '1' saídas em alta impedância)

E - carrega os dados D para os flip-flops (ativa em '1')

Figura 2.18

74573, 74xx573

*Latch 8 bits orientado a barramento*

D0-D7 - entrada de dados

LE - latch enable - habilitação do latch, em '1' armazena os dados

OE - output enable - habilitação das saídas ('0' - saídas ativas '1' - saídas em alta impedância)

Figura 2.19

**2.7 - Decodificadores/Demultiplexadores**

Em seguida temos alguns exemplos de CIs decodificadores/demultiplexadores. Os decodificadores são circuitos integrados utilizados para transformar um código em outro, por exemplo: binário em decimal, BCD em 7 segmentos, etc.

Os decodificadores são muito utilizados em estruturas de endereçamento de memória e seleção de periféricos em circuitos microprocessados ou microcontrolados.

O funcionamento de um decodificador é bastante simples. Considere um decodificador BCD para decimal como o 7442 apresentado em seguida. Ele possui quatro entradas e dez saídas. Para cada código binário colocado nas suas entradas, uma e somente uma saída será ativada (no caso irá a nível '0'). Exemplo: se colocarmos o binário 0101 em suas entradas, teremos a saída 5 ativada (em nível '0').

7442, 74xx42

**Decodificador BCD-Decimal (4 bits para 10 saídas), ativo em '0'**

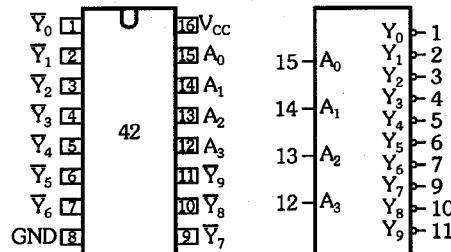
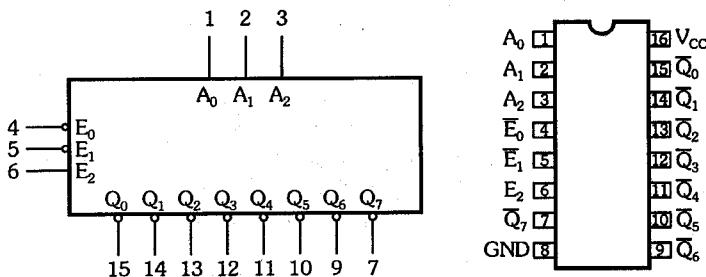


Figura 2.20

74138, 74xx138

**Decodificador/demultiplexador binário de 3 linhas para 8 saídas**



A<sub>0</sub>-A<sub>2</sub> - entradas seletoras

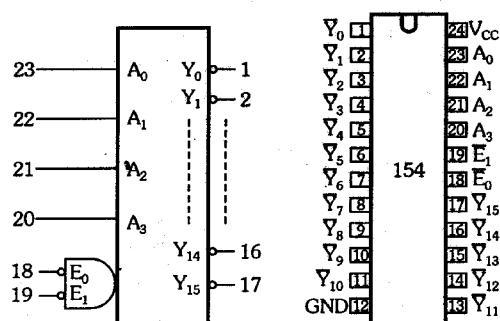
$\overline{E_0}, E_1, E_2$  - entradas de habilitação (a saída selecionada somente será ativada (nível '0') quando as entradas de habilitação estiverem E0='0', E1='0' e E2='1')

Q<sub>0</sub>-Q<sub>7</sub> - saídas decodificadas ativas em nível '0'

O valor binário colocado na entrada do CI é decodificado, ativando a saída correspondente em nível '0'

Figura 2.21

74154, 74xx154

**Decodificador/demultiplexador binário de 4 linhas para 16 saídas**

A0-A3 - entradas seletoras

Y0-Y15 - saídas decodificadas

$\overline{E_0}, \overline{E_1}$  - entradas de habilitação (em nível '0' o CI irá decodificar a entrada, em '1' as saídas permanecem em nível lógico '1', desativadas portanto)

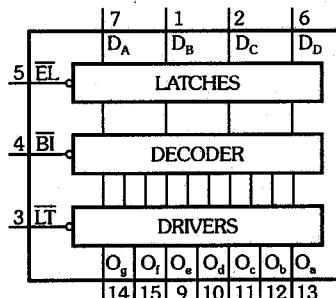
Figura 2.22

4511

**Decodificador/Latch/Driver BCD para 7 segmentos (Display)**

O CI 4511 é um decodificador/latch/driver CMOS BCD para displays de 7 segmentos.

Pelo fato de ser um decodificador BCD, o mesmo somente decodifica em suas saídas os valores entre 0 e 9.

D<sub>A</sub>, D<sub>B</sub>, D<sub>C</sub>, D<sub>D</sub> - Entradas de dadosO<sub>a</sub>, O<sub>b</sub>, O<sub>c</sub>, O<sub>d</sub>, O<sub>e</sub>, O<sub>f</sub> - saídas decodificadas p/display

EL - habilitação do Latch (quando em nível '0', o latch de entrada memoriza os dados)

BI - Blank Input - Apaga o display (ativa em '0')

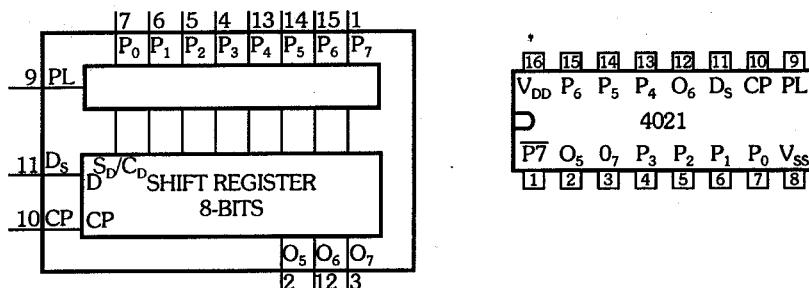
LT - Teste de display - acende todos os segmentos (ativa em '0')

Figura 2.23

## 2.8 - Registradores de Deslocamento

4021

O CI CMOS 4021 é um registrador de deslocamento de entrada paralela de 8 bits e saída serial com uma freqüência típica de operação de 13 Mhz quando operando em 5 volts.



P<sub>0</sub> a P<sub>7</sub> - entradas paralelas

PL - controle de carga paralela

D<sub>S</sub> - entrada de dados seriais

CP - clock serial (sensível à borda de subida)

O<sub>5</sub> a O<sub>7</sub> - saídas paralelas correspondentes aos últimos três estágios do registrador de deslocamento

**Figura 2.24**

UCN5810

O circuito integrado CMOS UCN5810 é fabricado pela Allegro Semiconductor e une em um mesmo encapsulamento um registrador de deslocamento de entrada serial e saída paralela de 10 bits, um latch de 10 bits e um driver de saída de 10 bits.

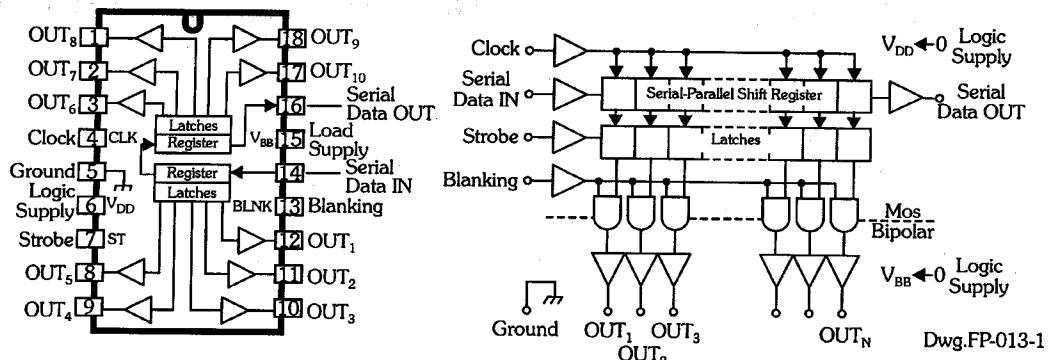
O registrador de deslocamento, responsável pela conversão da informação de entrada serial em uma informação paralela, pode operar a até 3,3 Mhz quando o CI é alimentado por 5V. Em tensões maiores, esta freqüência também aumenta sensivelmente. O registrador de deslocamento possui ainda uma saída serial que permite o cascamenteamento de diversos CI's em série.

O latch interno permite que a operação de conversão serial/paralelo transcorra sem interferir nos dados presentes na saída do dispositivo.

A transferência do registrador de deslocamento para os latches é controlada por uma entrada de strobe específica.

Finalmente, os drivers de saída são implementados com uma tecnologia híbrida formada por um conjunto darlington bipolar e um MOSFET DMOS, capazes de trabalhar com tensão de alimentação de até 60V e correntes de -40mA a +15mA. A tensão de alimentação dos drivers é separada da alimentação lógica.

Os drivers possuem ainda uma entrada de controle (**Blanking**) que quando ativada, faz com que as saídas sejam colocadas em nível lógico '0'.



OUT<sub>1</sub> a OUT<sub>10</sub> - saídas

SERIAL DATA IN - entrada de dados seriais (é amostrada a cada borda de subida do clock)

SERIAL DATA OUT - saída de dados seriais (utilizada para cascatear dispositivos em série)

CLOCK - entrada de clock para o registrador de deslocamento

STROBE - transferência de dados do registrador de deslocamento para os latches (ativo em '1')

BLANKING - controle de zeramento das saídas (não é um reset). Ativo em '1'

LOGIC SUPPLY - alimentação da parte lógica do CI (até 15 Volts)

LOAD SUPPLY - alimentação dos drivers de saída (até 60 Volts)

GROUND - terra de alimentação

**Figura 2.25**

## Microcontroladores PIC

Os microcontroladores PIC são uma família de dispositivos fabricados pela Microchip. Utilizando uma arquitetura RISC, com freqüências de clock de até 40Mhz, até 2048k word de memória de programa e até 3968 bytes de memória RAM. Além disso, podem ser encontrados com diversos periféricos internos, como: até quatro temporizadores/contadores, memória EEPROM interna, gerador/comparador/amostrador PWM, conversores A/D de até 12 bits, interface de barramento CAN, I2C, SPI, entre outros.

Existem basicamente três famílias de PICs diferenciadas pelo tamanho da palavra da memória de programa: 12, 14 e 16 Bits. Todos estes dispositivos possuem um barramento interno de dados de oito bits.

Recentemente a Microchip apresentou a sua mais nova linha de dispositivos, formando a quarta família chamada DsPIC com largura de dados de 16 bits (em vez dos 8 bits das outras famílias PIC), largura de memória de programa de 24 bits, além de implementar diversas facilidades que os tornam dispositivos intermediários aos DSP. De fato, eles são chamados de DSC (Digital Signal Controller - Controlador Digital de Sinais) numa analogia aos DSP (Digital Signal Processor - Processador Digital de Sinais).

O aumento no tamanho da palavra de programa possibilita um aumento no número de instruções: os PICs de 12 bits possuem 33 instruções, os de 14 bits, 35 instruções e os de 16 bits, até 77 instruções. Uma maior quantidade de instruções possibilita uma maior flexibilidade e eficiência na programação, mas com a contrapartida de oferecer um maior nível de dificuldade ao aprendizado.

Observe a tabela com alguns exemplos de cada família:

<b>12 Bits</b>	<b>14 Bits</b>	<b>16 Bits</b>
12C508	12C671	17C4x
12C509	12C672	17C75x
12CE518	12CE673	17C76x
12CE519	12CE674	18C2xx
16C54	14000	18C4xx
16C55	16C55x	18C858

12 Bits	14 Bits	16 Bits
16C56	16F62x	18F242
16C57	16C7x	18F252
16C58	16F7x	18F258
16C505	16F8x	18F442
16HV540	16F87x	18F458

A arquitetura de construção dos PICs é otimizada para a execução de um pequeno conjunto de instruções, mas com velocidade muito alta. De fato, podemos obter velocidades de até 5 MIPS (Milhões de Instruções por Segundo) com um PIC da série 16 trabalhando a 20Mhz (ou 10 MIPS a 40Mhz!), o que é um desempenho bastante expressivo, suficiente mesmo para as aplicações mais exigentes.

Em seguida destacamos algumas características básicas do desenho interno desses MCUs:

1. Capacidade de *pipeline* (enquanto executa uma instrução, o processador busca a próxima instrução na memória, de forma a acelerar a execução do programa);
2. Capacidade de execução de uma instrução por ciclo de máquina (as instruções que provocam desvio no programa são executadas em dois ciclos de máquina). Observe que um ciclo de máquina no PIC equivale a quatro ciclos de clock;
3. Cada instrução ocupa sempre apenas uma posição de memória de programa (devido à largura do barramento de dados da memória de programa ser maior, justamente para acomodar uma instrução inteira em apenas uma posição de memória);
4. Tempo de execução fixo para todas as instruções (com exceção das instruções de desvio). Devido ao fato de cada instrução do PIC ocupar apenas uma posição de memória, o tempo de execução é fixo, facilitando a determinação do tempo de execução de um programa.

### ***Observação***

As instruções que provocam desvio no fluxo do programa são executadas em dois ciclos de máquina. Isto porque a arquitetura de pipeline dos PICs busca uma nova instrução durante a execução da instrução atual. No entanto, quando a instrução atual provoca um desvio no programa, a CPU terá de gastar um ciclo na busca da instrução no novo endereço.

Outra característica importante da arquitetura PIC reside na semelhança e compatibilidade entre os diversos chips. Isto facilita grandemente a migração de um MCU para outro, pois os princípios básicos e grande parte dos registradores não diferem muito entre si.

Os princípios gerais que regem a arquitetura PIC (séries 12, 14, 16 e 18) são:

- ◆ Um registrador de intercâmbio geral, também chamado de registrador de trabalho (W);
- ◆ Um registrador destinado a armazenar flags resultantes de operações matemáticas, lógicas, etc. chamado STATUS;
- ◆ Um registrador destinado ao controle de interrupções, chamado INTCON;
- ◆ Registradores destinados à manipulação de dados nas portas (PORT ou GPIO).

### 3.1 - Memória de Programa

A memória para armazenamento de programas nos PICs das séries 12, 14 e 16 é interna, isto é, está embutida na pastilha do MCU, porém alguns dispositivos das séries 17 e 18 podem funcionar com memórias de programa externas.

Existem quatro tecnologias típicas de implementação da memória de programa:

- ◆ **ROM do tipo máscara:** alguns dispositivos podem ser encontrados com memória de programa do tipo máscara. Tais chips são fabricados com a memória de programa já programada de fábrica, num processo conhecido como máscara. Estes são os dispositivos que possuem os menores custos unitários, mas somente são viáveis em grandes quantidades. São identificados pelo sufixo "CR";
- ◆ **OTP:** os dispositivos OTP são fabricados com memórias de programa do tipo PROM, saindo de fábrica "virgens". Observe que tais dispositivos não podem ser apagados, o que impede sua utilização nas fases de desenvolvimento e testes. Tais dispositivos são identificados pelo sufixo "C";
- ◆ **EPROM:** normalmente os dispositivos ROM e OTP são também encontrados em versões com memória de programa do tipo EPROM. Tais dispositivos podem ser identificados pela janela característica das EPROMs, além do sufixo do encapsulamento, que pode ser "JW" para os dispositivos com encapsulamento do tipo DIP, ou "CL" para os dispositivos com encapsulamento do tipo PLCC. Os dispositivos EPROM são normalmente muito mais caros que os dispositivos ROM e OTP, somente sendo indicados para as etapas de desenvolvimento e testes;
- ◆ **FLASH:** os mais versáteis dispositivos da família PIC são os que possuem memória de programa do tipo FLASH. Tais dispositivos permitem um mínimo de 1000 ciclos de gravação/apagamento, sendo indicados tanto para as fases de desenvolvimento e testes, quanto para a implementação final dos circuitos. Atualmente, os custos dos dispositivos FLASH estão caindo grandemente, estando em alguns dispositivos bastante próximos aos PICs OTP equivalentes. Os dispositivos FLASH são identificados pelo sufixo "F".

A capacidade máxima de memória da série 16 é de 8kword, ou seja, 8192 posições de memória. No entanto, devido a limitações impostas pela estrutura das instruções RISC dos PICs, a memória de programa é dividida em blocos ou páginas, cada uma com 2kword ou 2048 posições.

Lembre-se que os dispositivos em estudo possuem somente 1kword (16F627) e 2kword (16F628), isto significa que eles somente possuem uma página de memória de programa.

Na figura 3.1 temos um diagrama da estrutura interna de endereçamento e memória de programa dos PICs da série 16. Observe que a primeira página da memória de programa (que tem início no endereço 0000h) existe em todos os modelos de PICs, porém o seu endereço final varia de acordo com o modelo do chip:

- ◆ 1FFh para os dispositivos com 512 words de memória;
- ◆ 3FFh para os dispositivos com 1kword de memória;
- ◆ 7FFh para os dispositivos com 2kword de memória.

0x0000	Vetor de Reset	
...		
0x0004	Vetor de Interrupção	Página 0
...		
0x07FF	Final da Página 0	
0x0800	Início da Página 1	
...		
...		
0xFFFF	Final da Página 1	Página 1
0x1000	Início da Página 2	
...		
...		
0x17FF	Final da Página 2	Página 2
0x1800	Início da Página 3	
...		
0xFFFF	Final da Página 3	Página 3

**Figura 3.1**

Observe que o funcionamento do mecanismo de paginação é transparente ao usuário, ou seja, no fluxo normal de um programa, a troca de uma página para outra é feita sem intervenção do usuário. Veja o exemplo seguinte:

Endereço	Instrução		
...			
0x07FE	MOVF PORTA,W	;	lê o conteúdo do PORTA e armazena em W
0x07FF	ADDLW 0x10	;	soma 10 hexadecimal ao conteúdo de W
0x0800	MOVWF PORTB	;	copia o conteúdo de W para o PORTB
0x0801	BCF INTCON,GIE	;	desliga as interrupções
...			

Pode-se verificar que após a execução da instrução ADDLW 0x10, localizada no endereço 0x07FF (portanto, o último endereço da página 0), o PC (registrador responsável pelo controle da sequência de execução das instruções) indicará a próxima

instrução a ser executada, que estará no endereço 0x07FF + 1, ou seja, no endereço 0x0800, que é o primeiro da página 1.

Isto significa que não há necessidade de utilizar o registrador PCLATH (registrador de acesso intermediário ao conteúdo do registrador PC) para trocar de página de memória, a não ser que seja necessário um desvio (GOTO) ou chamada de sub-rotina (CALL) para um endereço fora da página atual.

Maiores detalhes sobre o funcionamento da paginação de memória serão vistos no estudo do contador de programa dos PICs.

## 3.2 - Memória RAM e Registradores

A memória RAM dos PICs possui uma implementação um tanto peculiar quando comparada com outros dispositivos. De fato, a memória RAM dos PICs é utilizada essencialmente para abrigar os registradores internos e é dividida em duas partes distintas: uma destinada a abrigar os registradores de funções especiais (SFR) e outra para abrigar os registradores de propósitos gerais (GPR).

Os registradores **SFR** (Special Function Register - Registradores de Funções Especiais) são utilizados para controlar os periféricos e dispositivos internos, flags de estado do processador, entre outras funções.

Já os **GPR** (General Purpose Registers - Registradores de Propósito Geral) são utilizados para o armazenamento temporário de dados e informações do programa do usuário.

Os registradores SFR e GPR são muitas vezes referenciados simplesmente pela letra "F". Isto decorre da nomenclatura utilizada pela Microchip: *file registers*, ou registradores de arquivo.

Repare também que os registradores "F" (SFR e GPR) são mapeados dentro da memória RAM dos chips. Desta forma, o acesso aos registradores é feito pelo seu endereço de localização na memória RAM.

Um outro aspecto importante sobre a arquitetura dos registradores nos PICs é que a forma de implementação das instruções limita o endereçamento a um máximo de 7 bits ou 128 registradores.

Para contornar esta limitação, a Microchip implementou também na memória RAM a filosofia de paginação, criando bancos de memória, cada um com 128 posições.

Tal solução tornou possível o acesso a um conjunto muito maior de registradores, ao mesmo tempo em que manteve a compatibilidade de código entre os diversos chips da família PIC.

O controle da seleção do banco de memória em uso é feito por meio dos bits RP0 e RP1 localizados no registrador STATUS, como veremos adiante.

Desta forma, a arquitetura da série 16 está limitada a um conjunto de quatro bancos de memória, o que totaliza um máximo de 512 registradores (somando os registradores SFR e os GPR).

Outra característica importante e que deve ser ressaltada é o fato de que alguns registradores SFR podem ser encontrados em dois ou até mesmo nos quatro bancos de memória RAM, o que chamamos de registradores espelhados. Isto ocorre para otimizar o acesso a registradores muito utilizados (como o STATUS, INTCON, etc.). Desta forma torna-se menos necessária a tarefa de alteração do banco de memória em uso.

No entanto, o espelhamento não é um privilégio dos registradores SFR. Em inúmeros dispositivos encontramos também um conjunto de 16 registradores GPR localizados no final de cada banco de memória. Este também é um conjunto de registradores espelhados, ou seja: os últimos 16 registradores GPR de cada banco irão sempre acessar o conteúdo dos registradores equivalentes no banco 0. Isto permite que tenhamos uma área de armazenamento acessível a partir de qualquer banco de memória em que estejamos trabalhando, acelerando sensivelmente o tempo de execução de um programa (já que não necessitaremos de tantas mudanças de banco de memória).

Em alguns dispositivos podemos ter até mesmo um espelhamento completo dos registradores GPR (como no 16F84, por exemplo).

Devemos ainda ter em mente que, ao acessar um registrador, primeiramente necessitamos saber se ele se encontra localizado no banco de memória selecionado naquele instante pelo programa.

Caso ele pertença a outro banco, é imprescindível que se faça a mudança do banco de memória selecionado para o banco correspondente ao registrador. Caso contrário, o programa irá acessar um registrador, quando na verdade o programador desejava acessar outro. Este tipo de erro é relativamente comum e pode gerar os mais imprevisíveis resultados.

Dentre os registradores SFR, podemos destacar:

- ◆ **STATUS** - utilizado para armazenamento de flags matemáticos e de estado da CPU, além dos bits de seleção do banco de memória RAM;
- ◆ **INTCON** - utilizado para controle de interrupções;
- ◆ **OPTION\_REG** - utilizado para configurar o funcionamento de alguns periféricos internos do PIC;
- ◆ **PORTx** - utilizado para a leitura ou escrita de informações nos pinos externos do PIC;
- ◆ **TRISx** - utilizado para o controle da direção de funcionamento de cada pino da porta (se entrada ou saída).

Além dos registradores SFR e GPR, encontramos também outros com funções específicas, conforme veremos em seguida.

### 3.2.1 - Registrador W

O registrador **W** possui características peculiares e únicas:

- ◆ Pode ser utilizado como destino de diversas operações aritméticas e lógicas;
- ◆ Não pode ser utilizado como fonte de dados em diversas operações internas da CPU (rotação de dados, incremento, decremento, etc.);
- ◆ Ao contrário dos registradores SFR e GPR, o **W** não está mapeado na memória RAM dos PICs;
- ◆ É utilizado principalmente como "ponte" entre os registradores "f".

Dentre as características citadas, a mais importante, sem dúvida, é a utilização do registrador **W** como intermediário para troca de dados entre registradores "f" (SFRs e GPRs).

Isto ocorre devido a outra característica muito importante da arquitetura dos PICs da série 16: não é possível trocar diretamente informações entre os registradores "f".

A única forma de conseguir isso é copiando o conteúdo do registrador "f" de origem para o registrador **W** (por meio de uma instrução MOVF) e em seguida copiando o conteúdo de **W** para o registrador "f" de destino (por meio de uma instrução MOVWF).

### 3.2.2 - Contador de Programa - PC

O contador de programa (**PC** - Program Counter) é o responsável pelo controle da seqüência de execução das instruções no PIC. O registrador PC aponta sempre para a próxima instrução a ser executada pela CPU, desta forma, se desejarmos alterar o fluxo do programa, devemos alterar o conteúdo do PC.

Nos PICs da série 16, o contador de programa possui uma largura de 13 bits, o que implica dizer que é possível acessar até  $2^{13}$  ou 8192 instruções.

No entanto, ao contrário de muitos outros MCUs e MPUs, nos PICs não é possível ao usuário acessar diretamente o conteúdo do contador de programa.

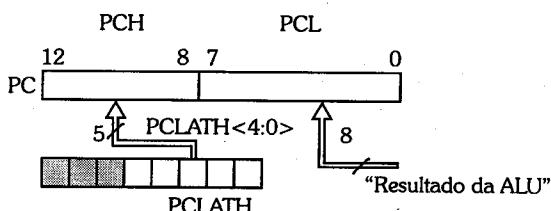
Podemos dizer que nos PICs, o PC de 13 bits é dividido em dois registradores básicos: PCL (responsável pelos 8 bits menos significativos do PC) e PCH (responsável pelos 5 bits mais significativos do PC).

Desses dois registradores, somente o PCL é diretamente acessível, podendo ser lido e escrito pelo programa do usuário.

Já o registrador PCH não pode ser lido pelo usuário, mas pode ser alterado pelo mesmo por intermédio do registrador PCLATH.

A interação entre os registradores PC, PCL e PCLATH pode ocorrer de quatro formas distintas:

**1. Execução de uma instrução que tem como destino o registrador PCL:**

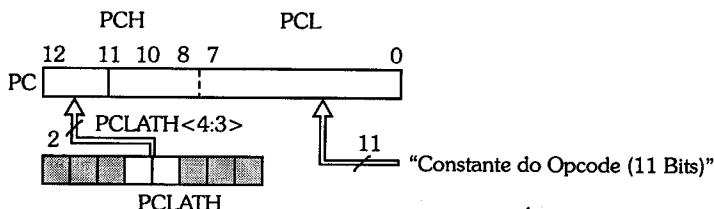


**Figura 3.2**

Neste caso, a execução de uma instrução cujo alvo é o registrador PCL fará com que o conteúdo do registrador PC seja carregado, em sua parte baixa com o valor escrito no registrador PCL e em sua parte alta (PCH) com os 5 bits LSB do registrador PCLATH.

Observe que apenas os 5 bits LSB do registrador PCLATH são implementados; os bits 7, 6 e 5 não estão implementados e são lidos como '0'.

**2. Execução de uma instrução GOTO:**



**Figura 3.3**

Quando uma instrução GOTO (desvio incondicional do fluxo do programa) é executada, o conteúdo do registrador PC é carregado da seguinte forma:

- 2.1.** O endereço de 11 bits incluso no opcode da instrução é carregado nos 11 bits inferiores do registrador PC;
- 2.2.** Os dois bits superiores do PC são carregados com os bits 4 e 3 do registrador PCLATH.

Observe que as instruções GOTO e CALL permitem saltos ou chamadas dentro de uma área máxima de  $2^{11}$  ou 2048 posições de memória. Desta forma, quando se fizer necessário um salto incondicional (GOTO) para um endereço fora da página de programa atual, será necessário programar o registrador PCLATH de acordo.

Tal procedimento somente é necessário nos dispositivos com mais de 2kword de memória (como os 16F873 e 16F874 com 4k e os 16F876 e 16F877 com 8k de memória). Nos dispositivos com memória de programa menor ou igual a 2kword, não é necessário utilizar o registrador PCLATH em qualquer

tipo de desvio, pois todos os endereços possíveis encontram-se dentro de uma mesma página de memória.

### 3. Execução de uma instrução CALL:

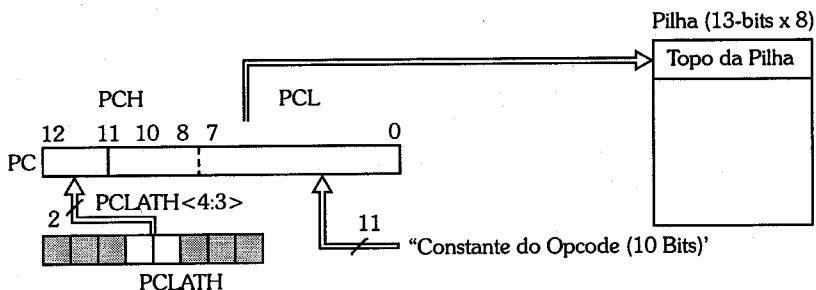


Figura 3.4

A execução de uma instrução CALL (chamada incondicional de sub-rotina) faz com que o conteúdo atual do registrador PC (que está apontando para a próxima instrução a ser executada após a instrução CALL) seja armazenado (PUSH) na pilha (STACK).

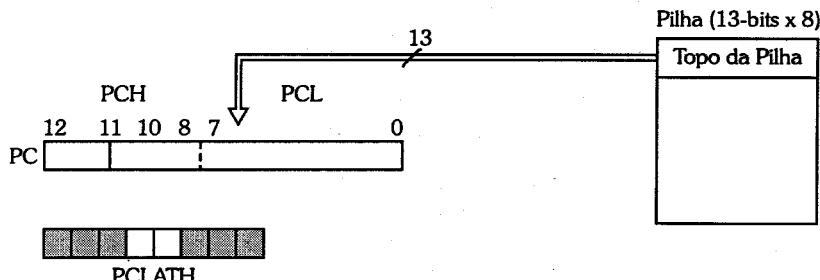
Em seguida, o registrador PC é carregado da seguinte forma:

- 3.1. O endereço de 11 bits incluso no opcode da instrução é carregado nos 11 bits inferiores do registrador PC;
- 3.2. Os dois bits superiores do PC são carregados com os bits 4 e 3 do registrador PCLATH.

Novamente observe que as instruções GOTO e CALL permitem saltos ou chamadas dentro de uma área máxima de  $2^{11}$  ou 2048 posições de memória. Desta forma, quando se fizer necessário uma chamada incondicional de sub-rotina (CALL) para um endereço fora da página de programa atual, será necessário programar o registrador PCLATH de acordo.

Tal procedimento somente é necessário nos dispositivos com mais de 2kword de memória (como os 16F873 e 16F874 com 4k e os 16F876 e 16F877 com 8k de memória). Nos dispositivos com memória de programa menor ou igual a 2kword, não é necessário utilizar o registrador PCLATH em qualquer tipo de desvio, pois todos os endereços possíveis encontram-se dentro de uma mesma página de memória.

#### 4. Execução de uma instrução RETURN, RETLW ou RETFIE:



**Figura 3.5**

A execução de uma instrução de retorno (RETURN, RETLW ou RETFIE) faz com que o conteúdo atual do registrador PC seja carregado com o valor que é puxado (POP) da pilha (STACK).

Observe que neste caso, o valor puxado da pilha possui um tamanho de 13 bits, preenchendo totalmente o registrador PC.

Desta forma, nas instruções de retorno, não é necessário configurar o registrador PCLATH em qualquer hipótese. Mesmo para dispositivos com mais de 2kword de memória.

### 3.2.3 - Pilha (*Stack*)

Outra estrutura encontrada no interior dos PICs é a pilha (em inglês *stack*) que nada mais é do que uma estrutura de dados normalmente localizada na memória RAM e que se caracteriza por apresentar apenas um ponto de acesso aos seus dados, denominado topo da pilha (em inglês TOS - **T**op **o**f **S**tack).

A estrutura utilizada para implementar a pilha é chamada de memória LIFO (**L**ast **I**n **F**irst **O**ut - último a entrar, primeiro a sair). Podemos exemplificar tal estrutura como uma pilha de pratos: os pratos são empilhados de baixo para cima. A ordem de esvaziamento é contrária: de cima para baixo, iniciando pelo último prato empilhado.

A principal função da pilha é armazenar endereços de retorno para sub-rotinas, isto é, quando da chamada de uma sub-rotina, o endereço da próxima instrução do programa (pois a atual já está em execução) é armazenado na pilha e o programa é desviado para o endereço da sub-rotina. Ao seu término, uma instrução especial normalmente chamada de retorno fará com que a CPU retire o último endereço armazenado na pilha e desvie o programa para ele, fazendo com que o fluxo do programa retorne ao ponto seguinte onde foi chamada a sub-rotina inicialmente. Alguns MCUs e MPUs permitem também o armazenamento de dados do usuário.

A pilha pode ser operada por meio de duas operações básicas:

- ◆ Empilhamento (*PUSH* - empurrar): quando a CPU armazena uma nova informação na pilha;

- ◆ Desempilhamento (*POP - Pull OPeration* - Operação de puxar): quando a CPU retira uma informação armazenada na pilha.

Nos PICs, a pilha não está localizada na área de memória RAM, mas em uma região separada (inacessível diretamente ao usuário) que é dedicada somente a ela.

O tamanho total dessa área é de oito posições (para os PICs de 14 bits), cada uma com 13 bits de capacidade (porque cada posição precisa ter capacidade suficiente para armazenar o valor do PC).

Além disso, a estrutura de implementação da pilha dos PICs é ligeiramente modificada em relação ao conceito antes descrito: o endereço topo da pilha não é acessível ao usuário e utiliza-se uma estrutura de memória LIFO circular. Isto significa que os pontos final e inicial da pilha são consecutivos, formando um círculo.

Desta forma, partindo do zero, podemos dizer que oito operações de empilhamento irão fazer com que a pilha fique totalmente ocupada. Neste caso, um novo empilhamento faria com que fosse sobreescrito o primeiro empilhamento antes realizado. Neste caso, o programa iria indubitavelmente terminar em um erro, já que o primeiro endereço de retorno foi alterado.

Observe ainda que a série 16 não implementa flags para verificação de estados da pilha, cabendo então ao programador observar cuidadosamente o fluxo do programa de forma a não ocorrerem empilhamentos além da capacidade da pilha (ou desempilhamentos aquém da capacidade).

Outro aspecto importante a respeito da estrutura de pilha dos PICs da série 16 é que ela não permite o armazenamento de dados do usuário, apenas endereços.

Além disso, não há instruções para manipular diretamente a pilha. Isto é feito indiretamente por meio de instrução CALL, RETURN, RETLW e RETFIE, além do evento de interrupção (que podemos considerar como um CALL).

Nas figuras seguintes podemos verificar o funcionamento da estrutura da pilha nos PICs:

8	vago
7	vago
6	vago
5	vago
4	vago
3	vago
2	vago
1	vago

Figura 3.6

8	vago
7	vago
6	vago
5	vago
4	vago
3	vago
2	vago
1	endereço 1

Figura 3.7

8	vago
7	vago
6	vago
5	vago
4	vago
3	vago
2	endereço 2
1	endereço 1

Figura 3.8

8	endereço 8
7	endereço 7
6	endereço 6
5	endereço 5
4	endereço 4
3	endereço 3
2	endereço 2
1	endereço 1

Figura 3.9

8	endereço 8
7	endereço 7
6	endereço 6
5	endereço 5
4	endereço 4
3	endereço 3
2	endereço 2
1	endereço 9

Figura 3.10

Na figura 3.6 temos a pilha totalmente vazia.

Na figura 3.7, temos a pilha após um empilhamento. Observe que este empilhamento poderia ser decorrente de uma chamada de sub-rotina ou de um evento de interrupção.

A figura 3.8 demonstra o estado da pilha após dois empilhamentos, o que poderia ocorrer com a chamada de uma sub-rotina de dentro de outra sub-rotina, ou o evento de uma interrupção durante a execução de uma sub-rotina. Observe que após o retorno da segunda sub-rotina, a pilha retornaria ao estado da figura 3.7.

Nas figuras 3.9 e 3.10 temos uma demonstração de uma sobrecarga da pilha. Na figura 3.10 efetuou-se um nono empilhamento, o que fez com que o primeiro endereço (endereço 1) que se encontrava na posição 1 da pilha se perdesse. Observe que a primeira operação de desempilhamento retira o endereço 9 da posição 1 da pilha, uma segunda operação de desempilhamento iria retirar o endereço 8 da posição 8, e assim por diante, até que uma nona operação de desempilhamento retire o endereço 9 que se encontra na posição 1 (e que na verdade deveria ser o endereço 1). Neste momento, poderíamos ter acontecimentos imprevisíveis no programa.

Observe que os erros provocados pela sobrecarga da pilha são muitas vezes difíceis de ser rastreados, devendo o programador tomar muito cuidado ao dimensionar suas sub-rotinas e especialmente observar que se forem utilizadas interrupções, deve-se considerar a pilha como tendo apenas sete posições, de forma a garantir a existência de uma posição vaga para ser ocupada a qualquer momento por uma interrupção.

### 3.3 - Interrupções

Antes de estudarmos as interrupções nos PICs, é necessário conhecer o conceito de interrupção.

É um evento externo ao programa que provoca a parada da sua execução, a verificação e tratamento do referido evento e em seguida o retorno do programa ao ponto em que havia sido interrompido.

As estruturas de interrupção são utilizadas para que a CPU tome conhecimento de eventos de alta prioridade para o programa.

Quando ocorre um evento de interrupção, a instrução em execução é completada, o conteúdo do contador de programa (ou equivalente) é armazenado na pilha e o programa é desviado para um endereço conhecido como vetor de interrupção.

No vetor de interrupção, deve existir uma sub-rotina que deverá providenciar o devido tratamento da interrupção e em seguida fazer o retorno da interrupção (o que normalmente é feito por uma instrução de retorno específica), o que conduz o programa devolva ao seu fluxo normal.

Existem dois tipos básicos de interrupção: a mascarável e a não mascarável.

- ◆ Na não mascarável, o evento de interrupção é atendido imediatamente pela CPU. Este tipo de interrupção não pode ser desativado (ou mascarado);
- ◆ Na mascarável, o evento de interrupção é atendido pela CPU desde que não esteja mascarado (desabilitado). Neste tipo de interrupção o programa define quais os momentos em que pode ou não ser interrompido.

Nos PICs, encontramos somente interrupções mascaráveis com um vetor de interrupção fixo (endereço 0x0004 na memória de programa).

As linhas PIC mais antigas possuem até quatro tipos de interrupção, entre:

- ◆ interrupção externa (INTF);
- ◆ interrupção por mudança de nível lógico no portb (RBIF);
- ◆ interrupção do timer0 (T0IF);
- ◆ interrupção de periférico (EEPROM - EEIF, Conversor A/D - ADIF);
- ◆ em alguns dispositivos de oito pinos encontramos a interrupção GPIF, relativa à porta de entrada/saída, que nesses dispositivos é chamada GPIO.

Tais interrupções são geralmente controladas pelo registrador INTCON.

Com a evolução dos dispositivos e a integração de novos periféricos internos, a Microchip passou a adotar uma nova arquitetura de interrupções. Ela ainda continuou baseada no registrador INTCON, mas com a adição de registradores destinados à sinalização e controle das interrupções dos novos dispositivos.

O fabricante passou a adotar então o termo "interrupção de periféricos" para todas as fontes de interrupção que não aquelas do registrador INTCON.

Desta forma, manteve-se a compatibilidade de software com os dispositivos mais antigos, além de permitir a adição de tantas interrupções de periféricos quantas forem necessárias.

Nessa arquitetura, as interrupções de periféricos são controladas por dois registradores: PIEx, responsável pelo controle de habilitação individual das interrupções dos periféricos, e PIRx, responsável pela sinalização individual dos eventos de interrupções dos periféricos.

Na figura 3.11, podemos verificar o diagrama completo da estrutura de interrupções para a série 16.

Observe que existe um total de 18 tipos diferentes de interrupção, nem todos disponíveis num mesmo chip ao mesmo tempo.

O funcionamento das interrupções nos PICs é baseado em uma estrutura na qual encontramos três sinais:

- ◆ O primeiro, chamado de GIE (Global Interrupt Enable - Habilitação Global de Interrupções), é responsável pelo controle geral das interrupções. Se o bit GIE estiver em nível lógico '1', podem ocorrer interrupções. Se o bit GIE estiver em nível lógico '0', não podem ocorrer interrupções. Podemos considerar o GIE como uma chave geral: se estiver ligada, podem ocorrer interrupções; se estiver desligada, não podem ocorrer interrupções;
- ◆ O segundo é o sinal de controle específico de cada interrupção. Se o sinal de habilitação individual da interrupção estiver habilitado (nível lógico '1'), pode ocorrer uma interrupção daquele tipo. Caso o bit esteja em nível lógico '0', a

interrupção individual estará desligada e não poderá interromper a CPU. Observe que cada fonte de interrupção do PIC vai possuir um sinal de controle individual de habilitação de interrupção;

- ◆ O terceiro sinal é o sinalizador (flag) de interrupção e é ativado (em nível lógico '1') pela fonte de interrupção sempre que ocorre um evento de interrupção. Esse sinal (flag) é que, após passar pelo controle individual de interrupção e pelo controle global de interrupção, vai efetivamente interromper a CPU.

Conforme podemos verificar na figura 3.11, cada *flag* individual de interrupção (representado pela sigla *xxIF*) é aplicado a uma porta AND, cuja função é realizar o controle de habilitação/desabilitação individual da interrupção.

Se a interrupção estiver habilitada (por meio do seu controle de habilitação individual *xxIE* em nível lógico '1'), o sinal de interrupção (*flag*) vai aparecer na saída da porta AND, sendo então aplicado a uma porta OR cuja função é unir todas as interrupções em um sinal único de interrupção.

Nos PICs básicos (como o 16F84) encontramos apenas uma porta OR (a porta A na figura 3.11). O sinal de saída dessa porta OR é utilizado para duas finalidades:

- ◆ Wake-up (acordar): é utilizado para fazer com que a CPU retorne do modo de baixo consumo de energia (SLEEP);
- ◆ Gerar o sinal de interrupção para a CPU, caso o sinal GIE esteja habilitado (nível lógico '1').

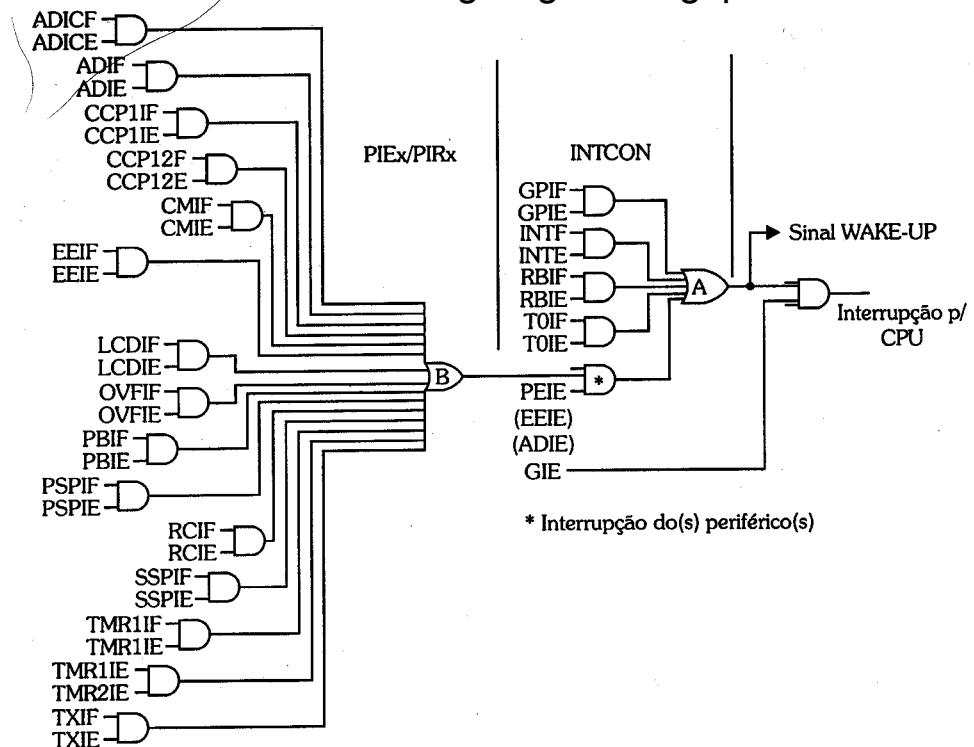
Da mesma forma verificamos que as interrupções dos periféricos são também combinadas em uma porta OR (porta B no diagrama), formando um sinal único de interrupção de periférico.

Observe que esse sinal único é controlado por uma porta AND que o combina com o sinal PEIE (Peripheral Interrupt Enable - Habilitação de Interrupção de Periféricos).

Assim, podemos verificar que o bit GIE funciona como uma "chave geral", um elemento de controle global. Se ele estiver em nível lógico '1', a CPU pode ser interrompida por qualquer uma das interrupções individuais (desde que ela esteja habilitada também!).

Caso o bit GIE esteja em nível lógico '0', as interrupções globais estarão desabilitadas, isto é, não será possível que qualquer interrupção (mesmo que individualmente habilitada) possa interromper a CPU.

Uma vez que ocorra uma interrupção, o bit GIE é desligado e o programa é desviado para o endereço 0x0004 (vetor de interrupções) no qual deverá haver uma sub-rotina para dar o devido tratamento à interrupção. Após o tratamento da interrupção, uma instrução RETFIE irá ligar novamente o bit GIE e fará com que o programa retorne ao ponto em que estava quando ocorreu a interrupção.



\* Interrupção do(s) periférico(s)

**Figura 3.11**

Um detalhe importante que devemos ter em mente quando escrevemos um programa que utiliza interrupções é sempre desligar o flag da interrupção que foi tratada, isto porque os flags de interrupção somente são setados pelo evento de interrupção, mas eles nunca são apagados pelo evento. O programador é que deverá providenciar o reset do flag que gerou uma interrupção, sob pena de o programa entrar em um "loop" infinito, ou seja, como após o tratamento da interrupção, o flag continua setado, ao terminar a sub-rotina de interrupção e retornar ao programa principal, o PIC irá detectar uma nova interrupção (que na realidade é a mesma, pois o flag não foi zerado) e novamente acontecerá uma interrupção e assim por diante *ad infinitum...*

Podemos concluir então que um sinal de interrupção somente irá interromper o programa em execução:

1. Se o controle individual de interrupção (xxIE) estiver habilitado;
2. Caso o sinal seja proveniente de um dos periféricos, o sinal PEIE esteja habilitado;
3. Se o sinal GIE estiver habilitado.

Observe ainda que o sinal para "acordar" a CPU (WAKE-UP) independe do estado do bit GIE.

**Repare que os flags de interrupção (xxIF) somente são setados pelo evento de interrupção. Cabe ao programador apagar o flag de interrupção após o tratamento dela!**

Desta forma, uma rotina de tratamento de interrupção (ISR) típica deve adotar os seguintes procedimentos:

1. Salvar os registradores que podem ser alterados dentro da ISR (normalmente o W, STATUS e eventualmente o PCLATH). Esta operação é chamada de salvamento de contexto;
2. Verificar o tipo de evento de interrupção que ocorreu;
3. Apagar o flag indicador da interrupção;
4. Tratar a interrupção;
5. Restaurar o contexto anterior (registradores W e STATUS aos seus valores originais);
6. Retornar da interrupção (RETFIE).

#### **Observações importantes sobre interrupções nos PICs:**

1. O vetor de interrupções é localizado no endereço 0x0004;
2. Os flags de interrupção (xxIF) somente são setados pelo evento de interrupção, nunca ressetados. O apagamento dos flags de interrupção deve ser providenciado pelo software do usuário;
3. Os flags de interrupção (xxIF) são setados independentemente do respectivo controle de habilitação. Vejamos um exemplo com a interrupção INTF: mesmo que o controle INTE esteja desabilitado (nível '0'), ainda assim, o flag INTF será setado pelo evento de interrupção externo (e assim permanecerá até que seja apagado pelo software do usuário, ou ocorra um reset da CPU);
4. Sempre que o processador desvia para o endereço 0x0004 para o tratamento de uma interrupção, o bit GIE é desligado;
5. A execução de uma instrução de retorno de interrupção (RETFIE) faz com que o bit GIE seja habilitado, permitindo uma nova ocorrência de interrupção.

### **3.3.1 - Tipos de Interrupção**

Como já dito, podemos encontrar até 18 tipos diferentes de interrupção na família PIC. Vejamos agora uma breve descrição de cada tipo de interrupção.

### 3.3.1.1 - INTF - Interrupção Externa

Esta interrupção consiste na forma básica de interrupção disponível em qualquer microcontrolador ou microprocessador.

Trata-se basicamente de um flip-flop que é setado pelo evento de interrupção e ressetado pelo programa do usuário.

Nos PICs podemos também selecionar a borda de sensibilidade da interrupção externa (se ela será ativada numa transição de nível '0' para nível '1' ou numa transição de nível '1' para nível '0' do sinal externo). Esta seleção é feita pelo bit INTEDG no registrador OPTION\_REG.

O flag de sinalização da interrupção externa é chamado de INTF e o controle de habilitação é chamado de INTE. Ambos encontram-se sempre no registrador INTCON.

### 3.3.1.2 - RBIF - Interrupção por Mudança de Estado no PortB

Este tipo de interrupção ocorre com a alteração de estado lógico em um dos pinos (RB7 a RB4) do PIC.

O PIC mantém internamente um conjunto de flip-flops que registra o último estado lido nos pinos RB7 a RB4. Os níveis provenientes desses flip-flops é constantemente comparado com os níveis lógicos dos pinos e, caso ocorra uma diferença, é gerado um sinal de interrupção.

Devido a esta característica de implementação, o sinal de interrupção pode ser apagado simplesmente pela leitura do registrador PORTB, o que provoca uma atualização do conteúdo dos registradores antes mencionados e por conseguinte anula a diferença de níveis existente.

Observe ainda que apenas os pinos programados como entrada é que são considerados na avaliação da porta B. Os pinos programados como saída não causam qualquer efeito com relação à interrupção por mudança na porta B.

O flag sinalizador dessa interrupção é chamado de RBIF e o sinal de controle é chamado de RBIE. Ambos encontram-se sempre no registrador INTCON.

### 3.3.1.3 - EEIF - Interrupção de Escrita na EEPROM

O evento de escrita na EEPROM interna é setado pelo hardware toda vez que uma operação de escrita na memória EEPROM interna é completada.

O flag sinalizador da interrupção é o EEIF e o bit de controle é chamado de EEIE. Em alguns dispositivos PIC o flag EEIE está localizado no registrador INTCON (como no 16F84, por exemplo). Nos dispositivos mais complexos, ambos os sinais encontram-se no registrador EECON1.

### 3.3.1.4 - T0IF - Interrupção do Timer 0

A interrupção do timer 0 ocorre sempre que o módulo contador/temporizador timer 0 estoura a sua contagem, ou seja, na transição de 255 (8 bits em '1') para 0 (ou 8 bits em '0') do timer 0.

O flag sinalizador da interrupção é chamado de T0IF e o bit de controle é chamado de T0IE. Ambos encontram-se sempre no registrador INTCON.

Maiores detalhes sobre o funcionamento da interrupção do timer 0 podem ser obtidos no estudo detalhado do módulo timer 0.

### 3.3.1.5 - TMR1IF - Interrupção do Timer 1

O evento de interrupção do timer 1 é disparado sempre que ocorre um estouro de contagem do módulo timer 1, ou seja, na transição de 65535 (16 bits em '1') para 0 (16 bits em '0').

O flag sinalizador dessa interrupção é chamado de TMR1IF e o bit de controle da habilitação é chamado de TMR1IE.

Maiores detalhes sobre o funcionamento da interrupção do timer 1 podem ser obtidos no estudo detalhado do módulo timer 1.

### 3.3.1.6 - TMR2IF - Interrupção do Timer 2

O evento de interrupção do timer 2 ocorre após "n" coincidências entre o valor da contagem do timer 2 e o valor do registrador de período PR2.

O valor de "n" é relativo ao fator de divisão do pós-divisor (postscaler) e pode assumir um dos seguintes valores: 1, 2, 4, 8 ou 16.

O flag sinalizador da interrupção do timer 2 é chamado TMR2IF e o bit de controle de habilitação da interrupção é chamado TMR2IE.

Maiores detalhes sobre o funcionamento da interrupção do timer 2 podem ser obtidos no estudo detalhado do módulo timer 2.

### 3.3.1.7 - CCP1IF - Interrupção do Módulo de Captura/Comparação/PWM

O evento de interrupção do módulo CCP é ativado em duas modalidades de funcionamento do módulo:

1. No modo de captura, quando uma captura é realizada;
2. No modo de comparação, quando uma comparação verdadeira é realizada.

O flag sinalizador da interrupção é chamado de CCP1IF e o bit de controle de habilitação da interrupção é chamado de CCP1IE.

Maiores detalhes sobre o funcionamento das interrupções no módulo CCP1 podem ser obtidos no estudo detalhado do módulo CCP1.

### 3.3.1.8 - TXIF - Interrupção de Transmissão da USART

A interrupção de transmissão da USART é disparada sempre que um caractere é passado para o registrador de deslocamento de transmissão (TSR).

O flag de sinalização do evento de interrupção é chamado de TXIF e o bit de controle da interrupção é chamado de TXIE.

Observe que o flag TXIF é apagado pela escrita de um novo caractere no registrador TXREG.

Maiores detalhes sobre o funcionamento da interrupção de transmissão podem ser vistos no estudo detalhado da USART.

### 3.3.1.9 - RCIF - Interrupção de Recepção da USART

A interrupção de recepção da USART é disparada sempre que um caractere é recebido e encontra-se disponível no registrador RCREG.

O flag de sinalização da interrupção de recepção da USART é chamado RCIF e o bit de controle de habilitação da interrupção é chamado de RCIE.

Observe que o flag RCIF é apagado pela leitura do(s) caractere(s) recebido(s) no registrador RCREG.

Maiores detalhes sobre o funcionamento da interrupção de recepção podem ser vistos no estudo detalhado da USART.

### 3.3.1.10 - CMIF - Interrupção do Módulo Comparador Analógico

O evento de interrupção do módulo comparador analógico é gerado sempre que ocorre uma mudança no estado de saída dos comparadores.

O flag de sinalização da interrupção é chamado CMIF e o bit de controle é chamado CMIE.

Observe que o flag CMIF pode ser apagado pela leitura do registrador CMCON.

Maiores detalhes sobre o funcionamento da interrupção do comparador analógico podem ser obtidos no estudo detalhado do módulo.

### 3.3.1.11 - CCP2IF - Interrupção do Módulo de Captura/Comparação/PWM

O módulo CCP2 possui funcionamento idêntico ao módulo CCP1. No PIC em estudo (16F627/16F628) o segundo módulo CCP não se encontra implementado.

O flag sinalizador da interrupção é chamado de CCP2IF e o bit de controle da interrupção é chamado de CCP2IE.

### 3.3.1.12 - SSPIF - Interrupção do Módulo Serial Síncrono

A interrupção do módulo serial síncrono é utilizada para sinalizar a finalização de uma transmissão/recepção.

O módulo serial síncrono implementa no hardware os protocolos de comunicação I<sub>2</sub>C e SPI.

Os PICs em estudo não dispõem deste módulo.

O flag sinalizador da interrupção é chamado de SSPIF e o bit de controle da interrupção é chamado de SSPIE.

### 3.3.1.13 - ADIF - Interrupção do Conversor Analógico/Digital

A interrupção do conversor analógico/digital é utilizada para sinalizar o término de um ciclo de conversão A/D.

Os PICs em estudo não dispõem de conversores A/D internos.

O flag sinalizador da interrupção é chamado ADIF e o bit de controle é chamado ADIE.

### 3.3.1.14 - ADCIF - Interrupção do Conversor Analógico/Digital do Tipo SLOPE

A interrupção do conversor A/D do tipo Slope é utilizada para sinalizar a mudança de estado no comparador do conversor A/D. Este tipo de conversor é utilizado na série PIC 14000.

O flag de interrupção é chamado ADCIF e o bit de controle de habilitação de interrupção é chamado ADCIE.

### 3.3.1.15 - OVFIF - Interrupção de Overflow do Timer do Conversor A/D SLOPE

Esta interrupção também está relacionada ao conversor A/D do tipo Slope mencionado anteriormente. Ela é utilizada para sinalizar o estouro de contagem do temporizador utilizado no conversor.

O flag de interrupção é chamado de OVFIF e o bit de controle de interrupção é chamado de OVFI.

### 3.3.1.16 - PSPIF - Interrupção de Escrita/Leitura na Porta Escrava Paralela

Esta interrupção é utilizada para sinalizar que ocorreu uma operação de escrita ou leitura na porta escrava paralela. Essa porta é encontrada nos dispositivos de 40 pinos

ou mais (tais como o 16F877, por exemplo) e é utilizada para troca de informações paralelas com outros dispositivos.

O flag indicador da interrupção é chamado PSPIF e o bit de controle de habilitação da interrupção é chamado de PSPIE.

### **3.3.1.17 - LCDIF - Interrupção do LCD**

Este tipo de interrupção está relacionada aos dispositivos que implementam drivers LCD internos (tais como os 16C92x, por exemplo).

O flag indicador da interrupção é chamado LCDIF e o bit de controle de habilitação da interrupção é chamado LCDIE.

### **3.3.1.18 - GPIF - Interrupção por Mudança de Estado no GPIO**

Este tipo de interrupção está relacionado aos dispositivos de oito pinos (tais como o 12C671, por exemplo) e que não utilizam a nomenclatura de PORTA, PORTB, etc. Tais dispositivos implementam apenas uma porta de entrada/saída que é chamada de GPIO.

Em tais dispositivos, a interrupção RBIF passa então a chamar-se GPIF e o bit de controle de habilitação da interrupção passa de RBIE para RBIF. Os bits continuam nas mesmas posições do registrador INTCON.

#### ***Observação***

As interrupções citadas nos itens 3.3.1.5 até 3.3.1.17 são periféricas e portanto os bits estão localizados nos registradores PIRx e PIEx respectivamente.

## **3.3.2 - Latência de Interrupções**

A latência de uma interrupção é definida como o atraso de tempo existente entre a ocorrência de um evento de interrupção e o efetivo desvio para o vetor de interrupções (nos PICs o endereço 0x0004).

No caso dos PICs podemos classificar as interrupções em duas categorias: síncronas e assíncronas.

Nas interrupções síncronas, tipicamente geradas por periféricos internos sincronizados com a CPU, a latência é de três ciclos de instrução, ou seja, após o evento de interrupção, o PIC executará ainda até outras três instruções antes de ocorrer o desvio para o vetor 0x0004.

Nas interrupções assíncronas, normalmente geradas por eventos externos (INT, por exemplo), a latência pode variar entre 3 e 3,75 ciclos de instrução, conforme o momento em que ocorre o evento de interrupção (já que nesse caso o evento não é sincronizado com a CPU).

### 3.3.3 - Eventos de Interrupção e o Modo SLEEP

Uma das formas de fazer com que o PIC retorne ao modo normal de funcionamento após um estado de SLEEP é a interrupção.

Conforme já visto, os sinais individuais de interrupção são combinados para formar um único sinal, sendo este utilizado para "acordar" (wake-up) a CPU.

No entanto, existem duas possibilidades distintas para a CPU após a saída de um modo SLEEP. Estas possibilidades são definidas pelo estado do bit GIE (controle global de interrupções) antes da entrada em modo SLEEP. São elas:

#### 3.3.3.1 - Interrupções Globais Habilitadas (GIE='1')

Caso o bit GIE esteja habilitado antes do modo SLEEP, um evento de interrupção terá três efeitos principais:

1. A CPU irá sair do modo SLEEP;
2. O programa será desviado para o tratamento da interrupção;
3. Após o retorno da interrupção, o programa prosseguirá normalmente do ponto em que havia sido interrompido.

Observe que devido à arquitetura do tipo *Pipeline* dos PICs, a instrução seguinte à instrução SLEEP será executada antes do desvio para o tratamento de interrupção. Isto porque a instrução seguinte à SLEEP já havia sido buscada na memória.

O flag PD estará em nível lógico '0', indicando que a CPU entrou em modo de baixa potência.

#### 3.3.3.2 - Interrupções Globais Desabilitadas (GIE='0')

Caso o bit GIE esteja desabilitado antes da execução da instrução SLEEP, um evento de interrupção terá apenas um efeito: fazer com que a CPU saia do modo de baixa potência e retorne à execução normal do programa.

Após o evento de interrupção, a CPU voltará a operar normalmente e o fluxo do programa seguirá da instrução seguinte à instrução SLEEP.

Também neste caso, o flag PD estará em nível lógico '0', indicando que a CPU entrou em modo de baixa potência.

#### Observações

1. Se um evento de interrupção ocorrer antes do início da execução de uma instrução SLEEP, a instrução será completada como um NOP, sem surtir nenhum dos seus efeitos característicos.
2. Se um evento de interrupção ocorrer durante ou após a execução de uma instrução SLEEP, o dispositivo entrará brevemente no modo SLEEP, "acordando" em seguida. Neste caso, devido ao breve período em modo de baixa potência, teremos o flag PD em nível '0'.

## Instruções

O MCU estudado neste curso pertence à categoria chamada Mid-range pela Microchip e possui uma largura de memória de programa de 14 bits.

Os chips de 14 bits possuem um conjunto de 35 instruções (contra 33 da série básica de 12 bits).

O conjunto de instruções dos PICs caracteriza-se pelo seu número reduzido de instruções, facilidade de aprendizado, flexibilidade e potência.

Parte das qualidades do conjunto de instruções deve ser creditada à simetria das mesmas, ou seja, praticamente cada instrução possui outra que realiza a operação inversa.

No entanto, antes de estudarmos as instruções propriamente ditas, vejamos a forma como são construídas as instruções RISC do PIC:

### 1. Instrução do tipo MNEMÔNICO **k**

Neste tipo de instrução, encontramos um operando **k** cujo valor será utilizado pela instrução de uma das duas formas:

- ◆ Armazenado no registrador W;
- ◆ Utilizado como parte do endereço de destino em instruções de desvio.

### 2. Instrução do tipo MNEMÔNICO **REG**

Neste tipo de instrução, encontramos um operando **REG** que representa um dos registradores SFR ou GPR disponíveis no PIC.

Observe que **REG** é representado por um código binário de 7 bits, o que nos limita a um máximo de 128 registradores. Daí a necessidade de implementar os bancos de memória no PIC.

#### *Observação*

O registrador W não pode ser colocado como operando nesta instrução.

### 3. Instrução do tipo MNEMÔNICO REG, DES:

Neste tipo de instrução, encontramos dois operandos: **REG** e **DES**.

**REG** representa um dos registradores SFR ou GPR disponíveis no PIC.

**DES** consiste em um bit indicador do destino da instrução. Podemos ter duas possibilidades:

1. Se no lugar de **DES** tivermos a letra "W", ou o número "0", teremos que o destino da instrução é o registrador W;
2. Se no lugar de **DES** tivermos a letra "f", ou o número "1", teremos que o destino da instrução é o próprio registrador representado por **REG**.

#### **Observação**

Lembre-se que o W não é um registrador SFR ou GPR!

### 4. Instrução do tipo MNEMÔNICO REG, BIT

Neste tipo de instrução, encontramos dois operandos: **REG** e **BIT**:

- ◆ **REG** representa um dos registradores SFR ou GPR disponíveis no PIC.  
Obs.: Lembre-se que W não é um registrador SFR ou GPR!
- ◆ **BIT** representa dos oito bits que compõem o registrador REG é o alvo da instrução. Por exemplo, numa instrução "MNEMÔNICO PORTB, 1", temos que o bit 1 da porta B será o alvo da instrução.

Antes de iniciarmos o estudo detalhado de cada instrução, vejamos em seguida alguns aspectos e convenções importantes adotados no texto:

1. Existem instruções que alteram indiretamente os flags no registrador STATUS, ou seja, devido à natureza de sua operação, certas instruções, após a realização de sua operação, alteram um ou mais flags de acordo com o resultado. Desta forma, além de alterar diretamente o registrador de destino da operação, essas instruções alteram também **indirectamente** o registrador STATUS;
2. **f** é uma abreviação de *file register* e é utilizado para especificar um dos registradores internos do PIC, tanto um SFR quanto um GPR (com exceção do W);
3. **d** é utilizado para especificar o destino da instrução. W ou F;
4. **b** é utilizado para especificar um dos 8 bits (0 a 7) do registrador especificado pelo primeiro operando;
5. Cada exemplo de instrução traz consigo uma tabela, demonstrando o estado dos flags C, DC e Z, antes e depois da execução da instrução. Observe que o

flag alterado direta ou indiretamente pela instrução possui o valor grafado em negrito e itálico para maior destaque. Os flags cujos valores estão grafados sem destaque não foram alterados pela instrução. Os valores grafados como X significam que o estado do flag não é conhecido naquele instante;

6. O tempo de execução de cada instrução aparece grafado como ciclos de máquina. Lembre-se que um ciclo de máquina do PIC equivale a quatro ciclos de clock ( $F_{osc}$ );
7. A base numérica utilizada nos exemplos é a hexadecimal, sendo os valores grafados como  $0xXX$ , onde XX é o valor hexadecimal representado. Esta representação é oficialmente utilizada pela Microchip no MPASM. Utilizamos a base hexadecimal pela facilidade de conversão para binário e também por ser a base numérica nativa dos sistemas digitais. Exemplo:  $0x10$  é igual a 16 decimal.

## 4.1 - Manipulação de Registradores

Estas instruções são utilizadas para armazenar, movimentar ou modificar dados nos registradores:

**MOVLWk**

Flags afetados: nenhum

Tempo de execução: 1 ciclo

Copia o valor da constante especificada pelo operando "k" para o registrador W ( $W = k$ ).

k no caso é uma constante (número) entre 0 e 255. Ao final da execução da instrução, o conteúdo registrador W será igual ao valor da constante representada por k.

Os flags C, DC e Z não são afetados por esta instrução e permanecerão com seus valores inalterados.

### Exemplo

Considere a execução da instrução MOVLW 0x10:

MOVLW 0x10		C	DC	Z
	antes	X	X	X
	depois	X	X	X

A instrução MOVLW copiou o valor da constante k (no caso 10 hexadecimal) para o registrador W. Então, após a execução da instrução, o conteúdo de W será igual a 0x10.

Os flags C, DC e Z não são alterados e permanecem com os mesmos valores.

**MOVWF f**

Flags afetados: nenhum

Tempo de execução: 1 ciclo

Copia o conteúdo do registrador **W** para o registrador **f** (**f = W**).

Após a execução da instrução MOVWF, o conteúdo do registrador W será copiado para o registrador especificado (**f**). O conteúdo do registrador W não será alterado; somente o registrador **f**.

O operando **f** pode especificar qualquer um dos registradores SFR ou GPR.

Os flags C, DC e Z não são afetados indiretamente por esta instrução e permanecerão com seus valores inalterados.

**Exemplo**

- a) Considere inicialmente que o conteúdo do registrador W é igual a 50 hexadecimal.

MOVWF PORTA		C	DC	Z
	antes	X	X	X
	depois	X	X	X

A instrução MOVWF fará com que o conteúdo presente no registrador W seja copiado para o registrador PORTA. Após a execução da instrução PORTA=0x50 e W=0x50.

Os flags C, DC e Z não são alterados e permanecem com os mesmos valores.

- b) Considere inicialmente que o conteúdo do registrador W é igual a 2A hexadecimal.

MOVWF TMR0		C	DC	Z
	antes	X	X	X
	depois	X	X	X

A instrução MOVWF fará com que o conteúdo presente no registrador W seja copiado para o registrador TMR0. Após a execução da instrução TMR0=0x2A e W=0x2A.

Os flags C, DC e Z não são alterados e permanecem com os mesmos valores.

**MOVF f, d**

Flags afetados: Z

Tempo de execução: 1 ciclo

Copia o conteúdo do registrador especificado pelo operando **f** para o destino especificado pelo operando **d**.

**d = W (ou d = '0')** - o conteúdo de **f** é armazenado no registrador W.

**d = F (ou d = '1')** - o conteúdo de **f** é armazenado no registrador especificado pelo operando **f**.

Esta instrução afeta indiretamente o flag Z no registrador STATUS:

Z = '1', se o valor armazenado no destino for zero;

Z = '0', se o valor armazenado no destino for diferente de zero.

### Exemplo

- a) Considere inicialmente que o conteúdo do registrador TRISA é igual a 20 hexadecimal.

MOVF TRISA,W		C	DC	Z
	antes	X	X	X
	depois	X	X	0

A instrução MOVF fará com que o conteúdo presente no registrador TRISA seja copiado para o destino indicado (no caso o registrador W). Após a execução da instrução TRISA=0x20 e W=0x20.

Observe que o flag Z foi apagado (Z = '0'), pois o valor escrito no destino é diferente de zero.

Os flags C e DC permanecem inalterados.

- b) Considere inicialmente que o conteúdo do registrador TRISA é igual a 20 hexadecimal.

MOVF TRISA,F		C	DC	Z
	antes	X	X	X
	depois	X	X	0

A instrução MOVF fará com que o conteúdo presente no registrador TRISA seja copiado para o destino indicado (no caso o próprio TRISA). Após a execução da instrução TRISA=0x20.

Observe que o flag Z foi apagado (Z = '0'), pois o valor escrito no destino é diferente de zero.

Os flags C e DC permanecem inalterados.

- c) Considere inicialmente que o conteúdo do registrador TRISB é igual a 0 decimal.

MOVF TRISB,W		C	DC	Z
	antes	X	X	X
	depois	X	X	1

A instrução MOVF fará com que o conteúdo presente no registrador TRISB seja copiado para o destino indicado (no caso o registrador W). Após a execução da instrução TRISB=0x00 e W=0x00.

Observe que o flag Z foi setado ( $Z = '1'$ ), pois o valor escrito no destino é igual a zero.

Os flags C e DC permanecem inalterados.

<b>CLRF</b>	<b>f</b>	Flags afetados: <b>Z</b>	Tempo de execução: 1 ciclo
-------------	----------	--------------------------	----------------------------

Apaga o conteúdo do registrador indicado pelo operando (**f**) (armazena 0 no mesmo), ou seja: ( $f = 0$ ).

Esta instrução afeta o flag Z:

$Z = '1'$ , pois o valor armazenado no destino é zero.

### Exemplos

- a) Considere inicialmente que o conteúdo do registrador TMR0 é igual a A0 hexadecimal.

CLRF	TMR0		C	DC	Z
		antes	X	X	X
		depois	X	X	<b>1</b>

A instrução CLRF fará com que o conteúdo do registrador f (no caso o TMR0) seja apagado. Após a execução da instrução, o conteúdo de TMR0 será igual a 0x00.

A instrução também irá setar o flag Z ( $Z = '1'$ ), indicando que o resultado foi igual a zero.

Os flags C e DC permanecem inalterados.

- b) Considere inicialmente que o conteúdo do registrador INTCON é igual a 11 hexadecimal.

CLRF	INTCON		C	DC	Z
		antes	X	X	X
		depois	X	X	<b>1</b>

A instrução CLRF fará com que o conteúdo do registrador f (no caso o INTCON) seja apagado. Após a execução da instrução, o conteúdo de INTCON será igual a 0x00.

A instrução também irá setar o flag Z ( $Z = '1'$ ), indicando que o resultado foi igual a zero.

Os flags C e DC permanecem inalterados.

- c) Considere inicialmente que o conteúdo do registrador STATUS é igual a 00011111 binário:

CLRF	STATUS		C	DC	Z
		antes	X	X	X
		depois	X	X	<b>1</b>

A instrução CLRF fará com que o conteúdo do registrador f (no caso o STATUS) seja apagado. Note, entretanto, que o registrador STATUS possui os bits PD e TO que são somente para leitura, não podendo ser apagados por software. Além disso, os flags C, DC e Z estão localizados no próprio STATUS, o que implicará que após a execução da instrução teremos STATUS = 00011100 binário!

A instrução também irá setar o flag Z ( $Z=1$ ), indicando que o resultado foi igual a zero.

Os flags C e DC permanecem inalterados.

**CLRW**

Flags afetados: **Z**

Tempo de execução: 1 ciclo

Apaga o conteúdo do registrador W (armazena 0 no mesmo), ou seja: ( $W = 0$ ).

Esta instrução afeta o flag Z:

$Z = '1'$ , pois o resultado armazenado em W é sempre zero.

### **Exemplo**

A instrução CLRW fará com que o conteúdo do registrador W seja apagado. Após a execução da instrução, o conteúdo de W será igual a 0.

A instrução também irá setar o flag Z, indicando que o resultado foi igual a zero.

Os flags C e DC permanecem inalterados.

CLRW		C	DC	Z
	antes	X	X	X
	depois	X	X	<b>1</b>

**INCF f, d**

Flags afetados: Z

Tempo de execução: 1 ciclo

Incrementa o conteúdo do registrador **f** em 1. O resultado (**f+1**) é armazenado no destino indicado pelo operando **d**:

**d = W** (ou **d = '0'**) - o resultado (**f+1**) é armazenado no registrador **W**.

**d = F** (ou **d = '1'**) - o resultado (**f+1**) é armazenado no registrador especificado pelo operando **f**.

Esta instrução afeta o flag **Z**:

**Z = '1'**, se o valor armazenado no destino for zero;

**Z = '0'**, se o valor armazenado no destino for diferente de zero.

### Exemplo

- a) Considere inicialmente que o conteúdo do registrador **W** é igual a 5 hexadecimal e o conteúdo do registrador **FSR** é igual a 9 hexadecimal:

			C	DC	Z
INCF	FSR, F	antes	X	X	X
		depois	X	X	0

A instrução INCF fará o incremento do registrador **f** (no caso o **FSR**). Isto é, irá adicionar 1 ao conteúdo do mesmo. Assim, o resultado ( $0x09 + 1 = 0x0A$ ) será armazenado no destino (no caso, **F**, ou seja, o próprio registrador **FSR**). Após a execução da instrução teremos: **FSR = 0x0A**, **W = 0x05**.

A instrução também irá apagar o flag **Z** (**Z = '0'**), indicando que o resultado foi diferente de zero.

Os flags **C** e **DC** permanecem inalterados.

- b) Considere inticialmente que o conteúdo do registrador **W** é igual a 5 hexadecimal e o conteúdo do registrador **TMR0** é igual a 6 hexadecimal:

			C	DC	Z
INCF	TMR0, W	antes	X	X	X
		depois	X	X	0

A instrução INCF fará o incremento do registrador **f** (no caso **TMR0**). Isto é, irá adicionar 1 ao conteúdo do mesmo. Assim, o resultado ( $0x06 + 1 = 0x07$ ) será armazenado no destino (no caso o registrador **W**). Após a execução da instrução teremos: **TMR0 = 0x06**, **W = 0x07**.

Observe que como o registrador **TMR0** não era o destino da instrução, o seu conteúdo não foi alterado!

A instrução também irá apagar o flag Z, indicando que o resultado foi diferente de zero.

Os flags C e DC permanecem inalterados.

- c) Considere inicialmente que o conteúdo do registrador W é igual a 5 hexadecimal e o conteúdo do registrador INDF é igual a FF hexadecimal:

			C	DC	Z
INCF	INDF, F	antes	1	0	0
		depois	1	0	<b>1</b>

A instrução INCF fará o incremento do registrador f (no caso INDF). Isto é, irá adicionar 1 a ele. Assim, o resultado ( $0xFF + 1 = 0x00$ ) será armazenado no destino (no caso, F, ou seja, o próprio registrador INDF). Após a execução da instrução teremos: INDF = 0x00, W = 0x05.

A instrução também irá setar o flag Z, indicando que o resultado foi igual a zero.

Os flags C e DC permanecem inalterados.

<b>DECF</b>	<b>f, d</b>	Flags afetados: <b>Z</b>	Tempo de execução: 1 ciclo
-------------	-------------	--------------------------	----------------------------

Decrementa um do conteúdo do registrador representado pelo operando **f**. O resultado (**f-1**) é armazenado no destino indicado pelo operando **d**.

**d** = W (ou **d** = '0') - o resultado é armazenado no registrador W;

**d** = F (ou **d** = '1') - o resultado é armazenado no registrador representado por **f**.

Esta instrução afeta o flag Z:

Z = '1', se o valor armazenado no destino for zero;

Z = '0', se o valor armazenado no destino for diferente de zero.

### Exemplo

- a) Considere inicialmente que o conteúdo do registrador W é igual a 5 hexadecimal e o conteúdo do registrador TMR0 é igual a 9 hexadecimal:

			C	DC	Z
DECF	TMR0, F	antes	1	0	1
		depois	1	0	<b>0</b>

A instrução DECF fará o decremento do registrador f (no caso TMR0). Isto é, irá subtrair 1 do conteúdo do mesmo. Assim, o resultado ( $0x09 - 1 = 0x08$ )

será armazenado no destino (no caso, F, ou seja, o próprio registrador TMR0). Após a execução da instrução teremos: TMR0 = 0x08 e W = 0x05.

A instrução também irá apagar o flag Z, indicando que o resultado foi diferente de zero.

Os flags C e DC permanecem inalterados.

- b) Considere inicialmente que o conteúdo do registrador W é igual a 5 hexadecimal e o conteúdo do registrador TMR0 é igual a 10 hexadecimal:

		C	DC	Z
DECF	TMR0, W	antes	1	0
		depois	1	0

A instrução DECF fará o decremento do registrador f (no caso TMR0). Isto é, irá subtrair 1 do conteúdo do mesmo. Assim, o resultado ( $0x10 - 1 = 0xF$ ) será armazenado no destino (no caso o registrador W). Após a execução da instrução teremos: TMR0 = 0x0F e W = 0x0F.

Observe que o registrador TMR0 não foi alterado pela instrução.

A instrução também irá apagar o flag Z, indicando que o resultado foi diferente de zero.

Os flags C e DC permaneceram inalterados.

- c) Considere inicialmente que o conteúdo do registrador W é igual a 5 hexadecimal e o conteúdo do registrador TMR0 é igual a 1 :

		C	DC	Z
DECF	TMR0, W	antes	1	0
		depois	1	0

A instrução DECF fará o decremento do registrador f (no caso TMR0). Isto é, irá subtrair 1 do conteúdo do mesmo. Assim, o resultado ( $0x01 - 1 = 0x00$ ) será armazenado no destino (no caso o registrador W). Após a execução da instrução teremos: TMR0 = 0x01, W = 0x00.

A instrução também irá setar o flag Z (Z= '1'), indicando que o resultado foi igual a zero.

Os flags C e DC permanecem inalterados.

<b>BCF</b>	<b>f, b</b>	Flags afetados: nenhum	Tempo de execução: 1 ciclo
------------	-------------	------------------------	----------------------------

Apaga (coloca em nível lógico '0') o bit representado pelo operando b do registrador representado pelo operando f.

Esta instrução não afeta nenhum flag.

### **Exemplo**

- a) Considere inicialmente que o conteúdo do registrador STATUS é igual a 19 hexadecimal ou em binário B'00011001':

BCF STATUS, 0		C	DC	Z
	antes	1	X	X
	depois	0	X	X

A instrução BCF irá apagar (colocar em nível '0') o bit representado por b (no caso o bit 0) do registrador representado por f (no caso o STATUS). Assim, o primeiro bit do registrador (bit 0) será colocado em zero.

Após a execução da instrução, o conteúdo do registrador STATUS será igual a B'00011000', ou 0x18 em hexadecimal.

Observe que apesar de a instrução não afetar indiretamente nenhum um dos flags, no presente caso foi determinado que ela alterasse **diretamente** o bit 0 do registrador STATUS, que na verdade é o flag de CARRY.

Poderíamos escrever a mesma instrução da seguinte forma: BCF STATUS, C.

Os flags DC e Z permanecem inalterados.

- b) Considere inicialmente que o conteúdo do registrador TRISB é igual a FF hexadecimal ou em binário B'11111111':

BCF TRISB, 1		C	DC	Z
	antes	X	X	X
	depois	X	X	X

A instrução BCF irá apagar (colocar em nível '0') o bit representado por b (no caso o bit 1) do registrador representado por f (no caso o TRISB). Assim, o segundo bit do registrador (bit 1) será colocado em zero.

Após a execução da instrução, o conteúdo do registrador TRISB será igual a B'11111101', ou 0xFD em hexadecimal.

Os flags C, DC e Z permanecem inalterados.

### **Observação**

No presente caso, devemos ter o cuidado de observar que o registrador TRISB encontra-se localizado no banco 1 ou 3!

A instrução BCF TRISB,1 configura o pino RB1 do port B como uma saída!

**BSF****f, b**

Flags afetados: nenhum

Tempo de execução: 1 ciclo

Seta (coloca em nível lógico '1') o bit representado pelo operando **b** do registrador representado pelo operando **f**.

Esta instrução não afeta nenhum flag.

### **Exemplo**

- a) Considere inicialmente que o conteúdo do registrador STATUS é igual a 18 hexadecimal ou em binário B'00011000':

BSF	STATUS, 2		C	DC	Z
		antes	X	X	X
		depois	X	X	1

A instrução BSF irá setar (colocar em nível '1') o bit representado por **b** (no caso o bit 2) do registrador representado por **f** (no caso o STATUS). Assim, o terceiro bit do registrador (bit 2) será colocado em nível lógico '1'.

Após a execução da instrução, o conteúdo do registrador STATUS será igual a B'00011100', ou em hexadecimal 0x1C.

Observe que apesar de a instrução não afetar indiretamente nenhum um dos flags, no presente caso a instrução irá alterar diretamente o bit 2 do registrador STATUS, que na verdade é o flag de ZERO, desta forma, Z=1 após a execução da instrução.

Poderíamos escrever a mesma instrução da seguinte forma: BSF STATUS, Z.

Os flags DC e C permanecem inalterados.

- b) Considere inicialmente que o conteúdo do registrador TRISB é igual a 0 ou em binário B'00000000':

BSF	TRISB, 1		C	DC	Z
		antes	X	X	X
		depois	X	X	X

A instrução BSF irá setar (colocar em nível '1') o bit representado por **b** (no caso o bit 1) do registrador representado por **f** (o registrador TRISB). Assim, o segundo bit do registrador (bit 1) será colocado em nível '1'.

Após a execução da instrução, o conteúdo do registrador TRISB será igual a B'00000010', ou em hexadecimal 0x02.

Os flags C, DC e Z permanecem inalterados.

**Observação**

No presente caso, devemos ter o cuidado de observar que o registrador TRISB encontra-se localizado no banco 1 ou 3!

A instrução BCF TRISB,1 configura o pino RB1 do port B como uma entrada!

**SWAPF f, d**

Flags afetados: nenhum

Tempo de execução: 1 ciclo

Troca os nibbles do registrador representado pelo operando **f**, isto é, troca os dígitos hexadecimais do registrador ( $B'xxxxyyyy'$  por  $B'yxxxxxxx'$ , ou  $0xXY$  por  $0xYX$ ). O resultado é armazenado no destino indicado pelo operando **d**.

**d** = W (ou **d** = '0') - o resultado é armazenado no registrador W;

**d** = F (ou **d** = '1') - o resultado é armazenado no registrador representado por **f**.

Esta instrução não afeta nenhum flag.

**Exemplo**

- a) Considere inicialmente que o conteúdo do registrador TRISB é igual a 0F hexadecimal ou em binário  $B'00001111'$ :

SWAPF	TRISB, F		C	DC	Z
		antes	X	X	X
		depois	X	X	X

A instrução SWAPF irá trocar os nibbles do registrador TRISB, armazenando o resultado (0xF0) no próprio TRISB.

Após a execução da instrução, o conteúdo do registrador TRISB será igual a  $B'11110000'$ , ou 0xF0 em hexadecimal.

Os flags C, DC e Z permanecem inalterados.

**Observação**

No presente caso, devemos ter o cuidado de observar que o registrador TRISB encontra-se localizado no banco 1 ou 3!

- b) Considere inicialmente que o conteúdo do registrador PORTA é igual a 78 hexadecimal ou em binário  $B'01111000'$  e o conteúdo do registrador W é igual a 0x00:

SWAPF	PORTA, W		C	DC	Z
		antes	X	X	X
		depois	X	X	X

A instrução SWAPF irá trocar os nibbles do registrador PORTA, armazenando o resultado (0x87) no registrador W.

Após a execução da instrução, o conteúdo do registrador PORTA será igual a 0x78 e W igual a B'10000111', ou 0x87 em hexadecimal.

Os flags C, DC e Z permanecem inalterados.

## 4.2 - Aritmética

As instruções aritméticas são utilizadas para, obviamente, realizar operações aritméticas (soma ou subtração).

**ADDLW k**

Flags afetados: **Z, DC e C**

Tempo de execução: 1 ciclo

Adição aritmética do valor representado pela constante "k" ao conteúdo do registrador W. O resultado (W+k) é armazenado no registrador W.

Esta instrução afeta os flags C, DC e Z:

C='1', se o resultado ultrapassar o valor 255;

C='0', se o resultado não ultrapassar o valor 255;

DC='1', se houver transbordo do terceiro para o quarto bit do registrador;

DC='0', se não houver transbordo do terceiro para o quarto bit do registrador;

Z='1', se o valor armazenado no destino for zero;

Z='0', se o valor armazenado no destino for diferente de zero.

### Observação

Para determinarmos o valor real armazenado no destino (W) quando superior a 255, basta subtrair 256 decimal do resultado. O valor resultante será o efetivamente armazenado no destino. Neste caso, o flag C estará em nível '1', indicando que houve a condição de estouro do registrador.

### Exemplo

- a) Considere inicialmente que o conteúdo do registrador W é igual a 20 hexadecimal:

ADDLW 0x15		C	DC	Z
	antes	X	X	X
	depois	0	0	0

A instrução ADDLW irá adicionar o valor da constante (15 hexadecimal) ao conteúdo de W (20 hexadecimal).

Após a execução da instrução, o conteúdo do registrador W será igual a  $(0x20 + 0x15 = 0x35)$ .

Os flags C, DC e Z são alterados:

Z = 0 indica que o valor armazenado em W é diferente de zero;

DC = 0 indica que não houve transporte do 3º para o 4º bit do registrador;

C = 0 indica que o resultado não ultrapassou 255 (ou 0xFF hexadecimal).

- b) Considere inicialmente que o conteúdo do registrador W é igual a F1 hexadecimal:

ADDLW 0x15		C	DC	Z
	antes	X	X	X
	depois	1	0	0

A instrução ADDLW irá adicionar o valor da constante (0x15) ao conteúdo de W (0xF1).

Após a execução da instrução, o conteúdo do registrador W será igual a  $(0xF1 + 0x15 = 0x105)$ .

Observe que o resultado 0x105 possui nove bits. Assim, somente os oito primeiros bits são armazenados no destino (W = 0x05). O nono bit será armazenado no carry, fazendo com que C = 1.

Os flags C, DC e Z são alterados:

Z = 0 indica que o valor armazenado em W é diferente de zero;

DC = 0 indica que não houve transporte do 3º para o 4º bit do registrador;

C = 1 indica que o resultado ultrapassou 255 (ou 0xFF hexadecimal).

- c) Considere inicialmente que o conteúdo do registrador W é igual a FF hexadecimal:

ADDLW 0x01		C	DC	Z
	antes	X	X	X
	depois	1	1	1

A instrução ADDLW irá adicionar o valor da constante (0x01) ao conteúdo de W (0xFF).

Após a execução da instrução, o conteúdo do registrador W será igual a  $(0xFF + 0x01 = 0x100)$ .

Observe que o resultado 0x100 possui nove bits. Assim, somente os oito primeiros bits são armazenados no destino (W = 0x00). O nono bit será armazenado no carry, fazendo com que C = '1'.

Os flags C, DC e Z são alterados:

Z = 1 indica que o valor armazenado em W é igual a zero;

DC = 1 indica que houve transporte do 3º para o 4º bit do registrador;

C = 1 indica que o resultado ultrapassou 255 (ou 0xFF hexadecimal).

### ADDDWF f, d

Flags afetados: **Z, DC e C**

Tempo de execução: 1 ciclo

Adição aritmética do conteúdo do registrador W ao conteúdo do registrador especificado por **f**. O resultado (W+f) é armazenado no destino indicado por **d**.

**d** = W (ou **d** ='0')- o resultado (W+f) é armazenado no registrador W -> W=W+f;

**d** = F (ou **d** ='1')- o resultado (W+f) é armazenado no registrador **f** -> **f**=W+f;

Esta instrução afeta os flags C, DC e Z:

C = '1', se o resultado ultrapassar o valor 255;

C = '0', se o resultado não ultrapassar o valor 255;

DC = '1', se houver transbordo do terceiro para o quarto bit do registrador;

DC = '0', se não houver transbordo do terceiro para o quarto bit do registrador;

Z = '1', se o valor armazenado no destino for zero;

Z = '0', se o valor armazenado no destino for diferente de zero.

### Observação

Para determinarmos o valor real armazenado no registrador de destino, quando superior a 255, basta subtrair 256 decimal do resultado. O novo valor resultante será o efetivamente armazenado no registrador de destino. Neste caso, o flag C estará em nível '1', indicando que houve a condição de estouro do registrador.

### Exemplo

- a) Considere inicialmente que o conteúdo do registrador W é igual a 20 hexadecimal e INTCON = 7 hexadecimal:

ADDWF INTCON, F		C	DC	Z
	antes	X	X	X
	depois	0	0	0

A instrução ADDWF irá adicionar o conteúdo de W (0x20) ao conteúdo do registrador INTCON (0x07). O resultado (0x27) será armazenado no destino indicado (F), ou seja, no registrador INTCON.

Após a execução da instrução, o conteúdo do registrador W será igual a 0x20 e INTCON = 0x27.

Os flags C, DC e Z são alterados:

Z = 0 indica que o valor armazenado em W é diferente de zero;

DC = 0 indica que não houve transporte do 3º para o 4º bit do registrador;

C = 0 indica que o resultado não ultrapassou 255 (ou 0xFF hexadecimal).

- b) Considere inicialmente que o conteúdo do registrador W é igual a F1 hexadecimal e que PORTA = 10 hexadecimal:

ADDWF	PORTA, W		C	DC	Z
		antes	X	X	X
		depois	1	0	0

A instrução ADDWF irá adicionar o conteúdo de W (0xF1) ao valor lido do PORTA (0x10). O resultado (0x101) será armazenado no registrador de destino, que no caso é o W. No entanto, como o resultado 0x101 possui nove bits, somente os oito primeiros serão armazenados no destino (W = 0x01). O nono bit será armazenado no carry, fazendo com que C = '1'.

Veja que o resultado pode também ser calculado:  $257 - 256 = 1$  !

Após a execução da instrução, o conteúdo do registrador W será igual a 0x01. O registrador PORTA não foi alterado e permanece com 0x10 hexadecimal.

Os flags C, DC e Z são alterados:

Z = 0 indica que o valor armazenado em W é diferente de zero;

DC = 0 indica que não houve transporte do 3º para o 4º bit do registrador;

C = 1 indica que o resultado ultrapassou 255 (ou 0xFF hexadecimal).

- c) Considere inicialmente que o conteúdo do registrador W é igual a FF hexadecimal e FSR = 01 hexadecimal:

ADDWF	FSR, F		C	DC	Z
		antes	X	X	X
		depois	1	1	1

A instrução ADDWF irá adicionar o conteúdo de W (0xFF) ao conteúdo do registrador FSR (0x01). O resultado (0x100 hexadecimal) será armazenado no destino indicado (o próprio registrador FSR).

Observe que o resultado 0x100 possui nove bits. Assim, somente os oito primeiros bits são armazenados no destino (FSR = 0x00). O nono bit será armazenado no carry, fazendo com que C = '1'. Isto equivale a  $256 - 256 = 0$ .

Os flags C, DC e Z são alterados:

Z = 1 indica que o valor armazenado em W é igual a zero;

DC = 1 indica que houve transporte do 3º para o 4º bit do registrador;

C = 1 indica que o resultado ultrapassou 255 (ou 0xFF hexadecimal).

**SUBLW k**

Flags afetados: **Z, DC e C**

Tempo de execução: 1 ciclo

Subtrai o conteúdo do registrador W do valor do operando "k". O resultado ( $k-W$ ) é armazenado no registrador W.

Esta instrução afeta os flags C, DC e Z:

C = '1', indica que o resultado é positivo ou igual a zero (não houve empréstimo);

C = '0', indica que o resultado é negativo (houve empréstimo);

DC = '1', não houve empréstimo do 4º para o 3º bit;

DC = '0', houve empréstimo do 4º para o 3º bit;

Z = '1', se o valor armazenado no destino for zero;

Z = '0', se o valor armazenado no destino for diferente de zero.

### **Observação**

Para determinarmos o valor real armazenado no registrador de destino (W), quando negativo, basta adicionar 256 decimal ao resultado. O novo valor resultante será o efetivamente armazenado no registrador de destino (W). Neste caso, o flag C estará em nível '0', indicando que houve a condição de empréstimo na subtração.

### **Exemplo**

a) Considere inicialmente que o conteúdo do registrador W é igual a 0x01:

SUBLW 0x09		C	DC	Z
	antes	X	X	X
	depois	<b>1</b>	<b>1</b>	<b>0</b>

A instrução SUBLW irá subtrair o conteúdo de W (0x01) do valor da constante k (0x09). O resultado ( $0x09 - 0x01 = 0x08$ ) será armazenado no registrador W.

Após a execução da instrução, o conteúdo do registrador W será igual a 0x08.

Os flags C, DC e Z são alterados:

Z = 0 indica que o valor armazenado em W é diferente de zero;

DC = 1 indica que não houve empréstimo do 4º para o 3º bit do registrador;

C = 1 indica que o resultado é positivo.

- b) Considere inicialmente que o conteúdo do registrador W é igual a 0x02:

SUBLW 0x00		C	DC	Z
	antes	X	X	X
	depois	0	0	0

A instrução SUBLW irá subtrair o conteúdo de W (0x02) do valor da constante k (0x00). O resultado ( $0x00 - 0x02 = -0x02$ ) será armazenado no registrador W. Observe entretanto que o valor  $-0x02$  é escrito no PIC como 0xFE ou 254 decimal. Para obtermos o valor real armazenado no registrador, devemos sempre adicionar 256 decimal ao resultado: -2 decimal + 256 decimal = 254 decimal.

Após a execução da instrução, o conteúdo do registrador W será igual a 0xFE.

Os flags C, DC e Z são alterados:

Z = 0 indica que o valor armazenado em W é diferente de zero;

DC = 0 indica que houve empréstimo do 4º para o 3º bit do registrador;

C = 0 indica que o resultado é negativo.

- c) Considere inicialmente que o conteúdo do registrador W é igual a 0x10:

SUBLW 0x10		C	DC	Z
	antes	X	X	X
	depois	1	1	1

A instrução SUBLW irá subtrair o conteúdo de W (0x10) do valor da constante k (0x10). O resultado ( $0x10 - 0x10 = 0x00$ ) será armazenado no registrador W.

Após a execução da instrução, o conteúdo do registrador W será igual a 0x00.

Os flags C, DC e Z são alterados:

Z = 1 indica que o valor armazenado em W é igual a zero;

DC = 1 indica que não houve empréstimo do 4º para o 3º bit do registrador;

C = 1 indica que o resultado é positivo ou zero.

**SUBWF f, d**

Flags afetados: **Z, DC e C**

Tempo de execução: 1 ciclo

Subtrai o conteúdo do registrador W do conteúdo do registrador especificado pelo operando **f**. O resultado (**f-W**) é armazenado no destino indicado pelo operando **d**.

**d = W** (ou **d = '0'**) - o resultado (**f-W**) é armazenado no registrador **W** -> **W=f-W**;

**d = F** (ou **d = '1'**) - o resultado (**f-W**) é armazenado no registrador **f** -> **f=f-W**.

Esta instrução afeta os flags C, DC e Z:

C = '1', indica que o resultado é positivo ou igual a zero (não houve empréstimo);

C = '0', indica que o resultado é negativo (houve empréstimo);

DC = '1', não houve empréstimo do 4º para o 3º bit;

DC = '0', houve empréstimo do 4º para o 3º bit;

Z = '1', se o valor armazenado no destino for zero;

Z = '0', se o valor armazenado no destino for diferente de zero.

### **Observação**

Para determinarmos o valor real armazenado no registrador de destino, quando negativo, basta adicionar 256 decimal ao resultado. O novo valor resultante será o efetivamente armazenado no registrador de destino. Neste caso, o flag C estará em nível '0', indicando que houve a condição de empréstimo na subtração.

### **Exemplo**

- a) Considere inicialmente que o conteúdo do registrador W é igual a 0x20 (32 decimal) e INTCON = 0x07 (7 decimal):

SUBWF    INTCON, F		C	DC	Z
	antes	X	X	X
	depois	0	1	0

A instrução SUBWF irá fazer a subtração do conteúdo do registrador W (32 decimal) do conteúdo do registrador INTCON (7 decimal). O resultado ( $32 - 7 = 25$  decimal) será armazenado no destino especificado pelo segundo operando, no caso o próprio registrador INTCON.

Observe, no entanto, que o número -25 é representado no PIC como 0xE7, ou 231 decimal. Para obtermos o valor real armazenado no registrador de destino, devemos somar 256 decimal (ou 0xFF hexadecimal) ao resultado negativo. Assim,  $-25 + 256 = 231$  decimal (ou 0xE7 hexadecimal).

Após a execução da instrução teremos: INTCON= 0xE7 e W=0x20.

Os flags C, DC e Z são alterados:

Z = 0 indica que o valor armazenado em W é diferente de zero;

DC = 1 indica que não houve transporte do 4º para o 3º bit do registrador;

C = 0 indica que o resultado é negativo.

- b) Considere inicialmente que o conteúdo do registrador W é igual a 0x09 (9 decimal) e PORTA = 0x10 (16 decimal):

SUBWF	PORTA, W		C	DC	Z
		antes	X	X	X
		depois	<b>1</b>	<b>0</b>	<b>0</b>

A instrução SUBWF irá fazer a subtração do conteúdo do registrador W (9 decimal) do conteúdo do registrador PORTA (16 decimal). O resultado ( $16 - 9 = 7$  decimal) será armazenado no destino especificado pelo segundo operando, no caso o registrador W.

Após a execução da instrução teremos: PORTA=0x10 e W=0x07.

Os flags C, DC e Z são alterados:

Z = 0 indica que o valor armazenado em W é diferente de zero;

DC = 0 indica que houve transporte do 4º para o 3º bit do registrador;

C = 1 indica que o resultado é positivo ou zero.

- c) Considere inicialmente que o conteúdo do registrador W é igual a 0x10 (16 decimal) e PORTA = 0x10 (16 decimal):

SUBWF	PORTA, W		C	DC	Z
		antes	X	X	X
		depois	<b>1</b>	<b>1</b>	<b>1</b>

A instrução SUBWF irá fazer a subtração do conteúdo do registrador W (16 decimal) do conteúdo do registrador PORTA (16 decimal). O resultado ( $16 - 16 = 0$  decimal) será armazenado no destino especificado pelo segundo operando, no caso o registrador W.

Após a execução da instrução teremos: PORTA= 0x10 e W= 0x00.

Os flags C, DC e Z são alterados:

Z = 1 indica que o valor armazenado em W é diferente de zero;

DC = 1 indica que não houve transporte do 4º para o 3º bit do registrador;

C = 1 indica que o resultado é positivo ou zero.

## 4.3 - Operações Lógicas

Existem também instruções específicas para realizar operações lógicas no PIC, as quais podem ser: AND, OR, XOR, NO, rotação à direita e rotação à esquerda.

**ANDLW k**Flags afetados: **Z**

Tempo de execução: 1 ciclo

AND lógico bit a bit do conteúdo da constante representada pelo operando "k" com o conteúdo do registrador W. O resultado (W AND k) é armazenado no registrador W.

Esta instrução afeta o flag Z:

$Z = '1'$ , se o valor armazenado no registrador W for zero;

$Z = '0'$ , se o valor armazenado no registrador W for diferente de zero.

**Exemplo**

Considere inicialmente que o conteúdo do registrador W é igual a 0x95 hexadecimal ou B'10010101':

ANDLW 0x0F		C	DC	Z
	antes	X	X	X
	depois	X	X	<b>0</b>

A instrução ANDLW irá realizar um AND lógico bit a bit do conteúdo do registrador W com o valor da constante k (no caso 0x00). Observe que a operação lógica é feita da seguinte forma:

BIT	7	6	5	4	3	2	1	0
W (0x95)	1	0	0	1	0	1	0	1
<b>AND</b>								
Constante (0x0F)	0	0	0	0	1	1	1	1
Resultado	0	0	0	0	0	1	0	1

Após a execução da instrução, o conteúdo do registrador W será igual a 0x05.

O flag Z será apagado ( $Z=0$ ), pois o resultado escrito no registrador W é diferente de zero.

Os flags DC e C não são alterados.

**ANDWF f, d**Flags afetados: **Z**

Tempo de execução: 1 ciclo

AND lógico do conteúdo do registrador W com o conteúdo do registrador especificado por **f**. O resultado (W AND f) é armazenado no destino indicado por **d**.

**d** = W (ou **d** = '0') - o resultado é armazenado no registrador **W**;

**d** = F (ou **d** = '1') - o resultado é armazenado no registrador **f**.

Esta instrução afeta o flag Z:

Z = '1', se o valor armazenado no destino for zero;

Z = '0', se o valor armazenado no destino for diferente de zero.

### **Exemplo**

Considere inicialmente que o conteúdo do registrador W é igual a 0x77 hexadecimal ou B'01110111' e PORTA= 0xFF ou B'11111111' binário:

ANDWF PORTA, W		C	DC	Z
	antes	X	X	X
	depois	X	X	0

A instrução ANDWF irá realizar um AND lógico bit a bit do conteúdo do registrador W (no caso 0x77) com o conteúdo do registrador PORTA (no caso 0xFF). O resultado da operação lógica será armazenado no registrador indicado pelo destino, ou seja: W. Observe que a operação lógica é feita da seguinte forma:

BIT	7	6	5	4	3	2	1	0
W (0x77)	0	1	1	1	0	1	1	1
<b>AND</b>								
PORATA (0xFF)	1	1	1	1	1	1	1	1
Resultado	0	1	1	1	0	1	1	1

Após a execução da instrução, o conteúdo do registrador W será igual a 0x77 e PORTA= 0xFF.

O flag Z será apagado (Z=0), pois o resultado escrito no registrador W é diferente de zero.

Os flags DC e C não são alterados.

**IORLW k**

Flags afetados: **Z**

Tempo de execução: 1 ciclo

OR lógico entre o valor constante especificado pelo operando "k" e o conteúdo do registrador W. O resultado (W OR k) é armazenado no registrador W.

Esta instrução afeta o flag Z:

Z = '1', se o valor armazenado no registrador W for zero;

Z ='0', se o valor armazenado no registrador W for diferente de zero.

**Exemplo**

Considere inicialmente que o conteúdo do registrador W é igual a 0x95 hexadecimal ou B'10010101':

IORLW    0x0F		C	DC	Z
	antes	X	X	X
	depois	X	X	0

A instrução IORLW irá realizar um OR lógico bit a bit do conteúdo do registrador W com o valor da constante k (no caso 0x00). Observe que a operação lógica é feita da seguinte forma:

BIT	7	6	5	4	3	2	1	0
W (0x95)	1	0	0	1	0	1	0	1
<b>OR</b>								
Constante (0x0F)	0	0	0	0	1	1	1	1
Resultado	1	0	0	1	1	1	1	1

Após a execução da instrução, o conteúdo do registrador W será igual a 0x9F.

O flag Z será apagado ( $Z=0$ ), pois o resultado escrito no registrador W é diferente de zero.

Os flags DC e C não são alterados.

**IORWF f, d**Flags afetados: **Z**

Tempo de execução: 1 ciclo

OR lógico do conteúdo do registrador W com o conteúdo do registrador especificado pelo operando **f**. O resultado ( $W \text{ OR } f$ ) é armazenado no destino indicado por **d**.

**d** = W (ou **d** = '0') - o resultado é armazenado no registrador W;

**d** = F (ou **d** = '1') - o resultado é armazenado no registrador **f**.

Esta instrução afeta o flag Z:

$Z = '1'$ , se o valor armazenado no destino for zero;

$Z = '0'$ , se o valor armazenado no destino for diferente de zero.

**Exemplo**

Considere inicialmente que o conteúdo do registrador W é igual a 0x77 hexadecimal ou B'01110111' e PORTA= 0xFF ou B'11111111' binário:

IORLW	PORTA, F		C	DC	Z
		antes	X	X	X
		depois	X	X	0

A instrução IORWF irá realizar um OR lógico bit a bit do conteúdo do registrador W (no caso 0x77) com o conteúdo do registrador PORTA (no caso 0xFF). O resultado da operação lógica será armazenado no registrador indicado pelo destino, ou seja, o registrador PORTA. Observe que a operação lógica é feita da seguinte forma:

BIT	7	6	5	4	3	2	1	0
W (0x77)	0	1	1	1	0	1	1	1
<b>OR</b>								
PORTA (0xFF)	1	1	1	1	1	1	1	1
Resultado	1	1	1	1	1	1	1	1

Após a execução da instrução, o conteúdo do registrador PORTA será igual a 0xFF e W = 0x77.

O flag Z será apagado (Z=0), pois o resultado escrito no registrador W é diferente de zero.

Os flags DC e C não são alterados.

### XORLW k

Flags afetados: **Z**      Tempo de execução: 1 ciclo

XOR lógico do valor da constante k com o conteúdo do registrador W. O resultado (W XOR k) é armazenado no registrador W.

Esta instrução afeta o flag Z:

Z = '1', se o valor armazenado no registrador W for zero;

Z = '0', se o valor armazenado no registrador W for diferente de zero.

### Exemplo

Considere inicialmente que o conteúdo do registrador W é igual a 0x04 hexadecimal ou B'00000100':

XORLW	0x12		C	DC	Z
		antes	X	X	X
		depois	X	X	0

A instrução XORLW irá realizar um XOR lógico bit a bit do conteúdo do registrador W com o valor da constante k (no caso 0x12). Observe que a operação lógica é feita da seguinte forma:

BIT	7	6	5	4	3	2	1	0
W (0x04)	0	0	0	0	0	1	0	0
<b>XOR</b>								
Constante (0x12)	0	0	0	1	0	0	1	0
Resultado	0	0	0	1	0	1	1	0

Após a execução da instrução, o conteúdo do registrador W será igual a 0x16.

O flag Z será apagado ( $Z=0$ ), pois o resultado escrito no registrador W é diferente de zero.

Os flags DC e C não são alterados.

### **XORWF f, d**

Flags afetados: **Z**

Tempo de execução: 1 ciclo

XOR lógico do conteúdo do registrador W com conteúdo do registrador especificado por **f**. O resultado ( $W \text{ XOR } f$ ) é armazenado no destino indicado por **d**.

**d** = W (ou **d** = '0') - o resultado é armazenado no registrador W;

**d** = F (ou **d** = '1') - o resultado é armazenado no registrador **f**.

Esta instrução afeta o flag Z:

$Z = '1'$ , se o valor armazenado no destino for zero;

$Z = '0'$ , se o valor armazenado no destino for diferente de zero;

### **Exemplo**

Considere inicialmente que o conteúdo do registrador W é igual a 0x22 hexadecimal ou B'00100010' e PORTA= 0x22 ou B'00100010' binário:

XORWF PORTA, W		C	DC	Z
	antes	X	X	X
	depois	X	X	<b>1</b>

A instrução IORWF irá realizar um OR lógico bit a bit do conteúdo do registrador W (no caso 0x22) com o conteúdo do registrador PORTA (no caso 0x22). O resultado da operação lógica será armazenado no registrador indicado pelo destino, ou seja, o registrador W. Observe que a operação lógica é feita da seguinte forma:

BIT	7	6	5	4	3	2	1	0
W (0x22)	0	0	1	0	0	0	1	0
<b>XOR</b>								
PORTA (0x22)	0	0	1	0	0	0	1	0
Resultado	0	0	0	0	0	0	0	0

Após a execução, o conteúdo do registrador W será igual a 0x00 e PORTA= 0x22.

O flag Z será setado ( $Z=1$ ), pois o resultado escrito no registrador W é igual a zero.

Os flags DC e C não são alterados.

**COMF f, d**

Flags afetados: **Z**

Tempo de execução: 1 ciclo

Complementa (inverte) o nível lógico dos bits do registrador especificado por **f**. O resultado ( $\bar{f}$ ) é armazenado no destino **d**.

**d** = W (ou **d** = '0') - o resultado é armazenado no registrador W;

**d** = F (ou **d** = '1') - o resultado é armazenado no registrador **f**.

Esta instrução afeta o flag Z:

$Z = '1'$ , se o valor armazenado no destino for zero;

$Z = '0'$ , se o valor armazenado no destino for diferente de zero.

### Exemplo

Considere inicialmente que o conteúdo do registrador W é igual a 0x01 hexadecimal e PORTA= 0x82 ou B'10000010' binário:

COMF	PORTA, W		C	DC	Z
		antes	X	X	X
		depois	X	X	<b>0</b>

A instrução COMF irá inverter os bits do conteúdo do registrador PORTA (no caso 0x82) com o conteúdo do registrador PORTA (no caso 0x22). O resultado da operação lógica será armazenado no registrador indicado pelo destino, ou seja, o registrador W. Observe que a operação lógica é feita da seguinte forma:

BIT	7	6	5	4	3	2	1	0
PORTA (0x22)	0	0	1	0	0	0	1	0
Resultado	1	1	0	1	1	1	0	1

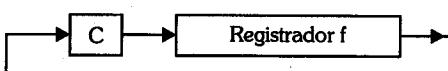
Após a execução da instrução, o conteúdo do registrador W será igual a 0xDD e PORTA= 0x82.

O flag Z será apagado ( $Z=0$ ), pois o resultado escrito no registrador W é diferente de zero.

Os flags DC e C não são alterados.

**RLF****f, d**Flags afetados: **C**

Tempo de execução: 1 ciclo

**Figura 4.1**

O conteúdo do registrador especificado por **f** é deslocado um bit à esquerda, o valor do flag **C** é armazenado no bit 0 do registrador e o bit excedente dele é armazenado no flag **C**. O resultado final é armazenado no registrador destino indicado por **d**.

**d = W** (ou **d = '0'**) - o resultado é armazenado no registrador **W**;

**d = F** (ou **d = '1'**) - o resultado é armazenado no registrador **f**.

Observe que a rotação acontece da seguinte forma:

C -> Bit 0 -> Bit 1 -> Bit 2 -> Bit 3 -> Bit 4 -> Bit 5 -> Bit 6 -> Bit 7 -> C

Ou seja: o bit **C** é armazenado no bit 0 do registrador de destino, o conteúdo anterior do bit 0 é armazenado no bit 1, o conteúdo anterior do bit 1 é armazenado no bit 2 e assim por diante até que o conteúdo anterior do bit 7 seja armazenado no flag de carry (**C**).

Esta instrução afeta o flag **C**.

### Observação

Uma rotação à esquerda consiste matematicamente em uma multiplicação por 2.

### Exemplo

- a) Considere inicialmente que o conteúdo do registrador TRISB= 0x42 ou B'01000010' binário, que o flag **C** = 1 e o conteúdo de **W**= 0x22:

RLF TRISB, W		C	DC	Z
	antes	1	X	X
	depois	0	X	X

A instrução RLF irá rotacionar o conteúdo do registrador TRISB um bit à esquerda, armazenando o resultado no destino especificado, no caso o registrador **W**:

BIT	C	7	6	5	4	3	2	1	0
Antes da rotação	1	0	1	0	0	0	0	1	0
		<-	<-	<-	<-				
Resultado	0	1	0	0	0	0	1	0	1

Observe que o conteúdo inicial do bit C ('1') foi colocado no bit 0 do registrador, que foi rotacionado à esquerda, fazendo com que o seu bit 7 ('0') seja armazenado no carry (C).

Repare que o valor presente inicialmente no flag C foi armazenado bit 0 do registrador, o valor anterior do bit 0 é armazenado no bit 1, o valor anterior do bit 1 é armazenado no bit 2, o valor anterior do bit 2 é armazenado no bit 3 e assim por diante até que o valor anterior do bit 7 é armazenado no flag C.

Após a execução da instrução, o conteúdo do registrador W será igual a 0x85 e TRISB= 0x42.

O flag C será apagado (C = 0), pois o bit excedente da rotação foi zero.

Os flags DC e Z não são alterados.

- b) Considere inicialmente que o conteúdo do registrador TRISB = 0x8F ou B'10001111' binário, que o flag C = '0' e o conteúdo de W= 0x22:

RLF TRISB, F		C	DC	Z
	antes	0	X	X
	depois	1	X	X

A instrução RLF irá rotacionar o conteúdo do registrador TRISB um bit à esquerda, armazenando o resultado no destino especificado, no caso o próprio registrador TRISB:

BIT	C	7	6	5	4	3	2	1	0
Antes da rotação	0	1	0	0	0	1	1	1	1
		<--	<--	<--	<--				
Resultado	1	0	0	0	1	1	1	1	0

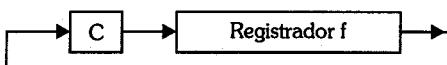
Observe que o conteúdo inicial do bit C ('0') foi colocado no bit 0 do registrador, que foi rotacionado à esquerda, fazendo com que o seu bit 7 ('1') seja armazenado no carry (C).

Repare que o valor presente inicialmente no flag C foi armazenado bit 0 do registrador, o valor anterior do bit 0 é armazenado no bit 1, o valor anterior do bit 1 é armazenado no bit 2, o valor anterior do bit 2 é armazenado no bit 3 e assim por diante até que o valor anterior do bit 7 é armazenado no flag C.

Após a execução da instrução, o conteúdo do registrador W será igual a 0x22 e TRISB = 0x1E.

O flag C será setado (C = 1), pois o bit excedente da rotação foi um.

Os flags DC e Z não são alterados.

**RRF****f, d**Flags afetados: **C** Tempo de execução: 1 ciclo**Figura 4.2**

O conteúdo do registrador especificado por **f** é deslocado um bit à direita, o valor do flag **C** é armazenado no bit 7 do registrador e o bit excedente dele é armazenado no flag **C**. O resultado final é armazenado no registrador destino indicado por **d**.

**d = W** (ou **d = '0'**) - o resultado é armazenado no registrador **W**;

**d = F** (ou **d = '1'**) - o resultado é armazenado no registrador **f**.

Observe que a rotação acontece da seguinte forma:

C -> Bit 7 -> Bit 6 -> Bit 5 -> Bit 4 -> Bit 3 -> Bit 2 -> Bit 1 -> Bit 0 -> C

Ou seja: o conteúdo do flag **C** é armazenado no bit 7 do registrador de destino; por sua vez, o conteúdo anterior do bit 7 é armazenado no bit 6, o conteúdo anterior do bit 6 é armazenado no bit 5 e assim por diante, até que o conteúdo anterior do bit 0 seja armazenado no flag **C**.

Esta instrução afeta o flag **C**.

### Observação

Uma rotação à direita consiste matematicamente em uma divisão (inteira) por 2.

### Exemplo

- a) Considere inicialmente que o conteúdo do registrador **TRISB** = 0x42 ou 'B'01000010' binário, que o flag **C** = 1 e o conteúdo de **W** = 0x22:

RRF TRISB, W		C	DC	Z
	antes	1	X	X
	depois	0	X	X

A instrução **RRF** irá rotacionar o conteúdo do registrador **TRISB** um bit à esquerda, armazenando o resultado no destino especificado, no caso o registrador **W**:

BIT	C	7	6	5	4	3	2	1	0
TRISB antes da rotação	1	0	1	0	0	0	0	1	0
		-->	-->	-->	-->				
Resultado	0	1	0	1	0	0	0	0	1

Observe que o conteúdo inicial do bit C ('1') foi colocado no bit 7 do registrador, que foi rotacionado à direita, fazendo com que o seu bit 0 ('0') seja armazenado no carry (C).

Repare que o valor presente inicialmente no flag C foi armazenado bit 7 do registrador, o valor anterior do bit 7 é armazenado no bit 6, o valor anterior do bit 6 é armazenado no bit 5, o valor anterior do bit 5 é armazenado no bit 4 e assim por diante até que o valor anterior do bit 0 é armazenado no flag C.

Após a execução da instrução, o conteúdo do registrador W será igual a 0xA1 e TRISB= 0x42.

O flag C será apagado (C = 0), pois o bit excedente da rotação foi zero.

Os flags DC e Z não são alterados.

- b) Consideré inicialmente que o conteúdo do registrador TRISB= 0x8F ou B'10001111' binário, que o flag C = 0 e o conteúdo de W= 0x22:

RRF TRISB, F		C	DC	Z
	antes	0	X	X
	depois	1	X	X

A instrução RLF irá rotacionar o conteúdo do registrador TRISB um bit à esquerda, armazenando o resultado no destino especificado, no caso o próprio registrador TRISB:

BIT	C	7	6	5	4	3	2	1	0
TRISB antes da rotação	0	1	0	0	0	1	1	1	1
		-->	-->	-->	-->	-->	-->	-->	-->
Resultado	1	1	1	0	0	0	1	1	1

Observe que o conteúdo inicial do bit C ('0') foi colocado no bit 7 do registrador, que foi rotacionado à direita, fazendo com que o seu bit 0 ('1') seja armazenado no carry (C).

Repare que o valor presente inicialmente no flag C foi armazenado bit 7 do registrador, o valor anterior do bit 7 é armazenado no bit 6, o valor anterior do bit 6 é armazenado no bit 5, o valor anterior do bit 5 é armazenado no bit 4 e assim por diante até que o valor anterior do bit 0 é armazenado no flag C.

Após a execução da instrução, o conteúdo do registrador W será igual a 0x22 e TRISB= 0xC7.

O flag C será setado (C = 1) pois o bit excedente da rotação foi um.

Os flags DC e Z não são alterados.

## 4.4 - Desvio

As instruções de desvio são utilizadas para desviar o fluxo do programa para outro local. Eles podem ser classificados em duas espécies: os desvios com retorno e os sem retorno.

Os desvios com retorno, também denominados de chamada de sub-rotina, ocorrem quando o fluxo do programa é desviado para outro ponto (sub-rotina) e após a execução dela, o fluxo retorna ao ponto inicial do desvio e segue adiante. Devemos lembrar que as chamadas a sub-rotinas fazem uso da pilha.

Nos desvios sem retorno, o fluxo do programa é desviado para outro ponto, seguindo então a partir de lá.

Podemos classificar ainda os desvios sem retorno em outras duas categorias: os desvios condicionais e os incondicionais:

Nos desvios condicionais, ocorre o desvio do programa se a condição for verdadeira.

Nos desvios incondicionais, o desvio ocorre independente de qualquer condição.

**GOTO k**

Flags afetados: nenhum

Tempo de execução: 2 ciclos

Desvia o programa para o ponto especificado por k. Este é o chamado desvio incondicional sem retorno. A CPU irá desviar o programa para outro ponto, prosseguindo a execução daquele ponto.

Lembre-se de que nos dispositivos com mais que 2k de memória de programa, pode ser necessário modificar o conteúdo do registrador PCLATH de acordo com o endereço especificado por k estar ou não dentro da mesma página de memória atual.

Esta instrução não altera nenhum flag.

### ***Exemplo***

Considere:

...

**GOTO PONTO2**

....

....

**PONTO2:**

....

No exemplo apresentado, o fluxo do programa será desviado do ponto atual em que se encontra a instrução GOTO para o ponto em que está localizado o PONTO2. O endereço referente ao PONTO2 será carregado no PC e o programa fará o desvio.

Os flags C, DC e Z não são alterados.

**CALL k**

Flags afetados: nenhum

Tempo de execução: 2 ciclos

Chamada de sub-rotina especificada por k. Na chamada de sub-rotina, a CPU irá guardar o conteúdo atual do PC (contador de programa) + 1 na pilha, e em seguida desviará para a sub-rotina. Ao término da sub-rotina, uma instrução RETURN (ou equivalente) irá retirar o (PC+1) da pilha e retornará ao ponto seguinte do qual foi chamada a sub-rotina, dando prosseguimento normal ao programa.

Lembre-se de que nos dispositivos com mais que 2k de memória de programa, pode ser necessário modificar o conteúdo do registrador PCLATH de acordo com o endereço especificado por k estar ou não dentro da mesma página de memória atual.

Nenhum flag é afetado.

### **Exemplo**

Considere:

...  
CALL PONTO2  
...

...  
PONTO2:  
...

...  
RETURN

No exemplo apresentado, o fluxo do programa será desviado do ponto atual em que se encontra a instrução CALL para o ponto em que está localizado o PONTO2. O endereço da instrução seguinte ao CALL (PC+1) é armazenado no pilha e o endereço referente ao PONTO2 é carregado no PC. Em seguida é feito o desvio.

Os flags C, DC e Z não são alterados.

**RETURN**

Flags afetados: nenhum

Tempo de execução: 2 ciclos

Retorno de sub-rotina. Após a execução da sub-rotina, a instrução RETLW retornará para o ponto de onde foi chamada a sub-rotina.

Como já foi visto, a instrução RETURN irá realizar uma operação de desempilhamento (POP), retirando um endereço da pilha e copiando-o para o registrador W.

Nenhum flag é alterado.

### **Exemplo**

Considere:

...  
CALL PONTO2  
...

...

PONTO2:  
...

RETURN

No exemplo apresentado, foi executada uma instrução CALL que desviou o programa para o PONTO2. O programa continuou a execução daquele ponto até encontrar a instrução RETURN, que fará com que o último endereço armazenado na pilha seja novamente colocado no PC, fazendo com o programa desvie para o ponto seguinte ao de onde ocorreu o CALL.

Os flags C, DC e Z não são alterados.

**RETLW k**

Flags afetados: nenhum

Tempo de execução: 2 ciclos

Retorno de sub-rotina. Após a execução da sub-rotina, a instrução RETLW retornará para o ponto seguinte de onde foi chamada a sub-rotina. Além disso, copia o valor k para o registrador W (W=k).

Como já foi visto, a instrução RETURN irá realizar uma operação de desempilhamento (POP), retirando um endereço da pilha e copiando-o para o registrador W.

Nenhum flag é alterado.

### **Exemplo**

Considere:

...  
CALL PONTO2  
...

...

...

PONTO2:  
...

RETLW 0x01

No exemplo apresentado, foi executada uma instrução CALL que desviou o programa para o PONTO2. O programa continuou a execução daquele ponto até encontrar a instrução RETURN, que fará com que o valor da constante k (no caso 0x01) seja armazenado no registrador W e o último endereço armazenado na pilha seja

novamente colocado no PC, fazendo com o programa desvie para o ponto seguinte ao de onde ocorreu o CALL.

Após a instrução, o registrador W = 0x01.

Os flags C, DC e Z não são alterados.

### RETFIE

Flags afetados: nenhum

Tempo de execução: 2 ciclos

Retorno de interrupção. Esta instrução retorna ao ponto seguinte do qual ocorreu a interrupção. Ela liga o bit GIE (GIE = '1') do INTCON.

Como já foi visto, a instrução RETURN irá realizar uma operação de desempilhamento (POP), retirando um endereço da pilha e copiando-o para o registrador W.

Nenhum flag é alterado.

### Exemplo

Considere:

```
...
CALL      PONTO2
...
...
PONTO2:
...
RETFIE
```

No exemplo mostrado, foi executada uma instrução CALL que desviou o programa para o PONTO2. O programa continuou a execução daquele ponto até encontrar a instrução RETURN, que fará com que o bit GIE no registrador INTCON seja setado (GIE=1) e o último endereço armazenado na pilha seja novamente colocado no PC, fazendo com o programa desvie para o ponto seguinte de onde ocorreu o CALL.

Após a instrução, o bit GIE = 1.

Os flags C, DC e Z não são alterados.

### BTFSC f, b

Flags afetados: nenhum

Tempo de execução: 1 ou 2 ciclos

Instrução de desvio condicional testa o bit especificado por **b** do registrador especificado por **f** e pula (não executa) a próxima instrução se o bit estiver em nível '0'. Esta instrução levará um ciclo de máquina para ser completada caso não ocorra o desvio (condição não é verdadeira) e dois ciclos de máquina no caso de ocorrer o desvio (a condição é verdadeira).

Esta instrução não afeta nenhum flag.

### **Exemplo**

- a) Considere que o conteúdo do registrador STATUS seja 0x18 (B'00011000' binário):

BTFSC STATUS, 2  
CLRW  
ADDLW 0x02

No exemplo exibido, a instrução BTFSC irá testar o bit 2 do registrador STATUS (o flag Z) e caso ele esteja em nível lógico '0', irá pular a próxima instrução. Como sabemos que o bit 2 do registrador STATUS está em '0', a condição será verdadeira. Então, após a execução da instrução BTFSC o programa será desviado para a instrução ADDLW, prosseguindo dali em diante. Observe que no caso, a instrução CLRW não foi executada.

Os flags C, DC e Z não são alterados.

### **Observação**

Poderíamos escrever BTFSC STATUS, Z no lugar de BTFSC STATUS,2.

- b) Considere que o conteúdo do registrador INTCON seja 0x01 (B'00000001' binário):

BTFSC INTCON,0  
CLRW  
ADDLW 0x02

Neste exemplo, a instrução BTFSC irá testar o bit 0 do registrador INTCON e caso ele esteja em nível lógico '0', irá pular a próxima instrução. Como sabemos com antecedência que o bit 0 do INTCON está em nível lógico '1' a condição não será verdadeira e o programa seguirá a execução normalmente.

Após a instrução BTFSC, o programa seguirá normalmente, executando o CLRW, ADDLW, etc.

Os flags C, DC e Z não são alterados.

**BTFS** f, b

Flags afetados: nenhum

Tempo de execução: 1 ou 2 ciclos

Instrução de desvio condicional testa o bit especificado pelo operando **b** do registrador especificado por **f** e pula (não executa) a próxima instrução se o bit estiver em nível '1'. Esta instrução levará um ciclo de máquina para ser completada caso não

ocorra o desvio (condição não é verdadeira) e dois ciclos de máquina no caso de ocorrer o desvio (a condição é verdadeira).

Esta instrução não afeta nenhum flag.

### **Exemplo**

- a) Considere que o conteúdo do registrador STATUS seja 0x18 (B'00011000' binário):

BTFS S STATUS, 2

CLRW

ADDLW 0x02

No exemplo exibido, a instrução BTFS irá testar o bit 2 do registrador STATUS (o flag Z) e caso ele esteja em nível lógico '1', irá pular a próxima instrução. Como sabemos que o bit 2 do registrador STATUS está em '1', a condição não será verdadeira e o programa seguirá a execução normalmente.

Após a instrução BTFS, o programa seguirá normalmente, executando o CLRW, ADDLW, etc.

Os flags C, DC e Z não são alterados.

- b) Considere que o conteúdo do registrador PORTA seja 0xA5 (B'10100101' binário):

BTFS S PORTA, 7

GOTO PONTOA

ADDLW 0x10

No exemplo apresentado, a instrução BTFS irá testar o bit 7 do registrador PORTA e caso ele esteja em nível lógico '1', irá pular a próxima instrução. Como sabemos que o bit 7 do registrador PORTA está em '1', a condição será verdadeira. Então, após a execução da instrução BTFS o programa será desviado para a instrução ADDLW, prosseguindo dali em diante. Observe que no caso, a instrução GOTO não foi executada.

Os flags C, DC e Z não são alterados.

**DECFSZf, d**

Flags afetados: nenhum

Tempo de execução: 1 ou 2 ciclos

Instrução de desvio condicional, decrementa o conteúdo do registrador especificado por **f** (subtraí 1 do valor). Pula a próxima instrução se o resultado (após o decremento) for igual a zero. O resultado (**f-1**) é armazenado no destino indicado por **d**.

**d** = W (ou **d** = '0') - o resultado é armazenado no registrador W;

**d** = F (ou **d** = '1') - o resultado é armazenado no registrador **f**.

Esta instrução não afeta nenhum flag.

Esta instrução pode demorar um ou dois ciclos de máquina para ser executada:

- ◆ Um ciclo se ela não provocar desvio no programa (o valor não for 0);
- ◆ Dois ciclos se ela provocar o desvio do programa (o valor é igual a 0).

### ***Exemplo***

- a) Considere que o conteúdo do registrador FSR seja 0x10:

DECFSZFSR, F

MOVLW0x05

ADDLW 0x01

No exemplo dado, a instrução DECFSZ irá decrementar o conteúdo do registrador FSR e armazenar o resultado no destino especificado (o próprio FSR). Como o resultado do decremento ( $0x10 - 0x01 = 0x0F$ ) não é igual a zero, a próxima instrução será executada normalmente.

Após a execução da instrução DECFSZ, o valor do registrador FSR será igual a 0x0F hexadecimal.

Os flags C, DC e Z não são alterados.

- b) Considere que o conteúdo do registrador FSR seja 0x01:

DECFSZFSR, F

MOVLW0x05 ; não executa esta instrução!

ADDLW 0x01

No exemplo dado, a instrução DECFSZ irá decrementar o conteúdo do registrador FSR e armazenar o resultado no destino especificado (o próprio FSR). Como o resultado do decremento ( $0x01 - 0x01 = 0x00$ ) é igual a zero, a próxima instrução não será executada e o programa será desviado para a instrução ADDLW.

Após a execução da instrução DECFSZ, o valor do registrador FSR será igual a zero.

Os flags C, DC e Z não são alterados.

### ***Observação***

Apesar do resultado da operação anterior ser zero, a instrução não irá alterar o flag Z!

**INCFSZ f, d**

Flags afetados: nenhum

Tempo de execução: 1 ou 2 ciclos

Instrução de desvio condicional, incrementa o conteúdo do registrador especificado por **f** (adiciona 1 ao valor dele). Pula a próxima instrução se o resultado (após o incremento) for igual a zero. O resultado ( $f+1$ ) é armazenado no destino indicado por **d**.

**d** = W (ou **d** = '0') - o resultado é armazenado no registrador W;

**d** = F (ou **d** = '1') - o resultado é armazenado no registrador F.

Esta instrução não afeta nenhum flag.

Esta instrução pode demorar um ou dois ciclos de máquina para ser executada:

- ◆ Um ciclo se ela não provocar desvio no programa (o valor não for 0);
- ◆ Dois ciclos se ela provocar o desvio do programa (o valor é igual a 0).

### **Exemplo**

- a) Considere que o conteúdo do registrador FSR seja 0x00:

```
INCFSZ FSR, W
MOVLW 0x05
ADDLW 0x01
```

No exemplo exibido, a instrução INCFSZ irá incrementar o conteúdo do registrador FSR e armazenar o resultado no destino especificado (o registrador W). Como o resultado do incremento ( $0x00 + 0x01 = 0x01$ ) não é igual a zero, a próxima instrução será executada normalmente.

Após a execução da instrução INCFSZ, o valor do registrador FSR será igual a zero e o registrador W será igual a 0x01.

Os flags C, DC e Z não são alterados.

- b) Considere que o conteúdo do registrador FSR seja 0xFF:

```
INCFSZ FSR, F
MOVLW 0x05      ; não executa esta instrução
ADDLW 0x01
```

No exemplo apresentado, a instrução INCFSZ irá incrementar o conteúdo do registrador FSR e armazenar o resultado no destino especificado (o próprio FSR). Como o resultado do incremento ( $0xFF + 0x01 = 0x00$ ) é igual a zero, a próxima instrução não será executada e o programa será desviado para a instrução ADDLW.

Após a execução da instrução INCFSZ, o valor do registrador FSR será igual a zero.

Os flags C, DC e Z não são alterados.

## 4.5 - Controle

Utilizada para funções internas e controle da CPU.

**NOP**

Flags afetados: nenhum

Tempo de execução: 1 ciclo

Sem operação. Esta instrução não executa nenhum operação. Não altera nenhum flag. O processador gasta um ciclo de máquina sem executar nenhuma operação.

**CLRWDT**

Flags afetados:  $\overline{\text{TO}}$  e  $\overline{\text{PD}}$

Tempo de execução: 1 ciclo

Limpa (zera) o contador do watchdog. As instruções CLRWDT são utilizadas para periodicamente, dentro do programa, resetar o watchdog, impedindo que o mesmo venha a ressetar a CPU.

Flags afetados:  $\overline{\text{TO}}$ ,  $\overline{\text{PD}}$  são setados pela instrução.

### Exemplo

Considere:

CLRWDT		$\overline{\text{TO}}$	$\overline{\text{PD}}$
	antes	X	X
	depois	<b>1</b>	<b>1</b>

A instrução irá fazer com que o conteúdo do temporizador do watchdog seja zerado, impedindo temporariamente que o watchdog ressete a CPU. Além disso, os flags  $\overline{\text{TO}}$  e  $\overline{\text{PD}}$  são setados (colocados em nível lógico '1').

Os flags C, DC e Z não são alterados.

**SLEEP**

Flags afetados:  $\overline{\text{TO}}$  e  $\overline{\text{PD}}$

Tempo de execução: 1 ciclo.

Aciona o modo de baixa potência da CPU. O processamento é parado e o conteúdo da memória é preservado.

Flags afetados:  $\overline{\text{TO}} = '1'$  e  $\overline{\text{PD}} = '0'$ .

### Exemplo

a) Considere:

CLRWDT		$\overline{\text{TO}}$	$\overline{\text{PD}}$
	antes	X	X
	depois	<b>1</b>	<b>0</b>

A instrução irá fazer com que o PIC entre no modo power-down ou SLEEP, também irá setar o flag TO e zerar o flag PD.

Os flags C, DC e Z não são alterados.

### ***Observação***

Existem ainda duas outras instruções, chamadas OPTION e TRIS, a primeira pode ser utilizada para armazenar um valor diretamente no registrador OPTION\_REG. A segunda é utilizada para armazenar um valor diretamente no registrador TRIS especificado.

Estas instruções não são abordadas na presente obra pois são de utilização não recomendada pela Microchip.

O fabricante aconselha a não utilização destas instruções em nenhum projeto, pois apesar de funcionarem na maioria dos dispositivos atualmente disponíveis, tais instruções são limitadas e incompatíveis com diversos novos membros da família PIC.

## PIC 16F627/628

O dispositivo que iremos estudar nesta obra é o PIC 16F627 e o PIC 16F628. Estes dispositivos são membros da chamada família intermediária (Mid-range) da Microchip.

Suas principais características são:

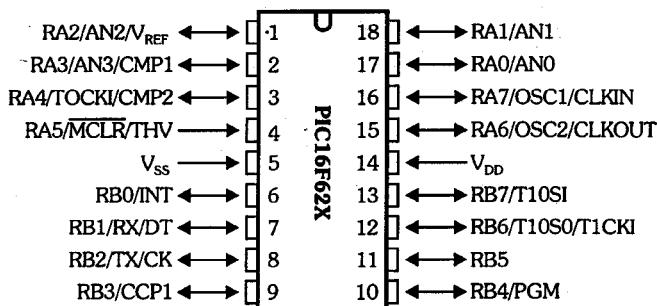
- ◆ Baixo custo;
- ◆ Facilidade de programação;
- ◆ Grande diversidade de periféricos internos;
- ◆ Compatibilidade em nível de software e de hardware com outros PICs de 18 pinos (como o 16F84, por exemplo);
- ◆ Memória de programa do tipo FLASH;
- ◆ Excelente velocidade de execução.

Além disso podemos também destacar as seguintes especificações:

- ◆ 1024 x 14 bits de memória FLASH (2048 x 14 bits para o 16F628);
- ◆ 224 x 8 bits de memória SRAM disponíveis para o usuário;
- ◆ 128 x 8 bits de memória EEPROM interna;
- ◆ Pilha com 8 níveis;
- ◆ 15 pinos de I/O (entrada ou saída);
- ◆ 1 pino de entrada;
- ◆ 1 timer/contador de 8 bits;
- ◆ 1 timer/contador de 16 bits;
- ◆ 1 timer de 8 bits;
- ◆ 1 canal PWM com captura e amostragem (CCP);
- ◆ 1 canal de comunicação USART serial;
- ◆ 2 comparadores analógicos com referência interna programável de tensão;
- ◆ 1 timer watchdog;
- ◆ 10 fontes de interrupção independentes;
- ◆ Capacidade de corrente de 25 mA por pino de I/O;
- ◆ 35 instruções;

- ◆ Freqüência de operação desde DC (0 Hz) até 20 Mhz;
- ◆ Oscilador 4Mhz/37Khz interno;
- ◆ Tensão de operação entre 3.0 a 5.5V (2.0 a 5.5 para versão LF);
- ◆ Compatível pino a pino com 16F84 e outros PICs de 18 pinos.

## 5.1 - Descrição dos Pinos



**Figura 5.1**

Pino	Função	Tipo	Descrição
1	RA2/AN2/Vref	Entrada/saída	Porta A bit 2 / Entrada comparador analógico / Saída da referência de tensão
2	RA3/AN3/CMP1	Entrada/saída	Porta A bit 3 / Entrada comparador analógico / Saída comparador 1
3	RA4/T0CKI/CMP2	Entrada/saída	Porta A bit 4 / Entrada de clock externo do timer 0 / Saída comparador 2. Esse pino possui saída em <b>dreno aberto</b> .
4	RA5/MCLR /THV	Entrada	Porta A bit 5 / Reset CPU / Tensão de programação
5	Vss	Alimentação	Terra
6	RB0/INT	Entrada/saída	Porta B bit 0 / Entrada de interrupção externa
7	RB1/RX/DT	Entrada/saída	Porta B bit 1 / Recepção USART (modo assíncrono) / Dados (modo síncrono)
8	RB2/TX/CK	Entrada/saída	Porta B bit 2 / Transmissão USART (modo assíncrono) / Clock (modo síncrono)
9	RB3/CCP1	Entrada/saída	Porta B bit 3 / Entrada/saída do módulo CCP
10	RB4/PGM	Entrada/saída	Porta B bit 4 / Entrada de programação LVP
11	RB5	Entrada/saída	Porta B bit 5
12	RB6/T1OSO/T1CKI	Entrada/saída	Porta B bit 6 / Saída oscilador TMR1 / Entrada Clock TMR1

Pino	Função	Tipo	Descrição
13	RB7/T1OSI	Entrada/saída	Porta B bit 7 / Entrada oscilador TMR1
14	V <sub>DD</sub>	Alimentação	Alimentação positiva
15	RA6/OSC2/CLKOUT	Entrada/saída	Porta A bit 6 / Entrada para cristal oscilador / Saída de clock
16	RA7/OSC1/CLKIN	Entrada/saída	Porta A bit 7 / Entrada para cristal oscilador / Entrada de clock externo
17	RA0/AN0	Entrada/saída	Porta A bit 0 / Entrada comparador analógico
18	RA1/AN1	Entrada/saída	Porta A bit 1 / Entrada comparador analógico

## 5.2 - Estrutura Interna

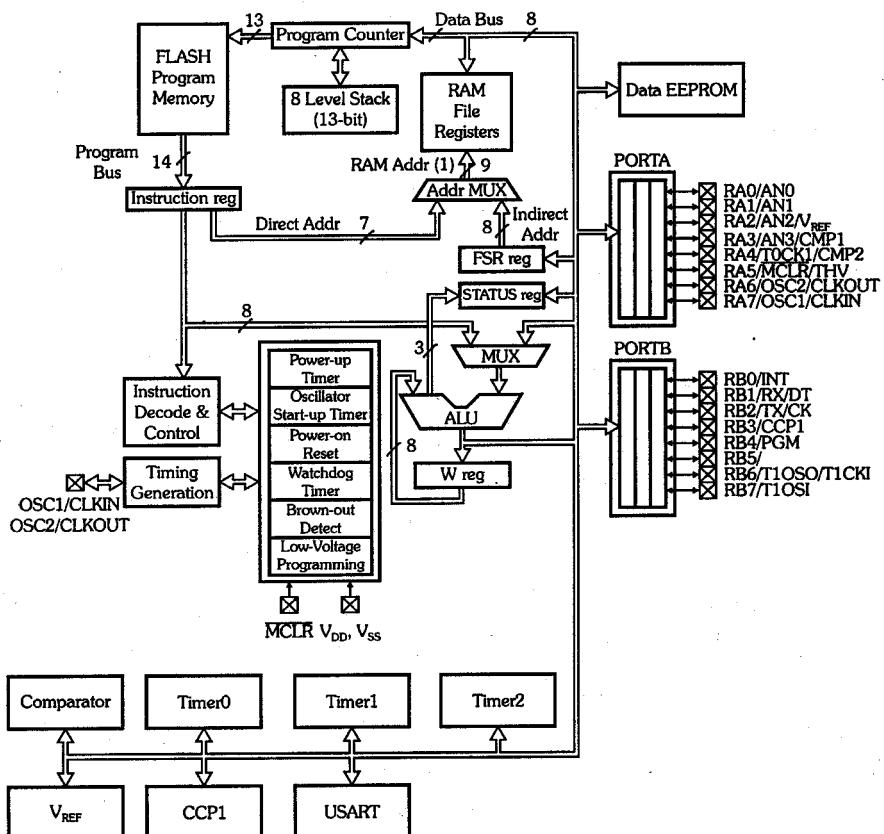


Figura 5.2

Na figura 5.2 podemos verificar a estrutura interna do PIC 16F62x. Podemos destacar a memória EEPROM interna, a memória RAM estática (também chamada file

*registers* - registradores de arquivo), a pilha (*stack*), o contador de programa PC (**Program Counter**).

Ressalta-se a diferença de largura do barramento de dados da memória de programa (*program bus*) (14 bits) e a do resto do processador (*data bus*) (que é de 8 bits). A diferença de largura entre o barramento de dados da memória de programa e da memória RAM decorre da arquitetura Harvard utilizada.

## 5.3 - Registradores

Na figura 5.3 temos o mapa de memória do 16F62x, no qual podemos localizar todos os registradores SFR e GPR, bem como seus respectivos endereços e bancos.

■ Posições não implementadas, lidas como "0".

\* Não é um registrador físico

*Figura 5.3*

Em seguida veremos uma breve descrição dos registradores SFR dos PICs 16F62x e estudaremos cada um em maiores detalhes.

STATUS	Utilizado para verificação de estado geral do processador, nele ficam armazenados os flags C (carry), DC (digit carry) e Z (zero), além dos flags TO e PD relativos ao Watchdog e modo Sleep respectivamente. Também nesse registrador encontramos os bits de seleção de banco de memória RP0 e RP1. O registrador STATUS está presente em todos os bancos de memória do PIC.
OPTION_REG	Utilizado para configurar algumas funções internas do PIC (prescaler, timer0, resistores pull-up, bordas de interrupção e clock, etc.). Localizado nos bancos de memória 1 e 3.
PCON	Registrador de controle de periféricos internos do chip. Localizado no banco de memória 1.
INDF	Registrador de acesso indexado. Utilizado para acesso indireto a uma posição de memória, juntamente com o registrador FSR. O princípio de funcionamento do acesso indireto é muito simples: o registrador INDF irá apontar a posição de memória a ser acessada e o registrador FSR irá fazer o acesso a ela. O registrador INDF está presente em todos os bancos de memória do PIC.
FSR	O registrador FSR é utilizado para acessar indiretamente o conteúdo da posição de memória indicada pelo registrador INDF. O registrador FSR está presente em todos os bancos de memória do PIC.
PCL	Utilizado para acesso aos 8 bits menos significativos do contador de programa (PC) do PIC. Esse registrador é de leitura/escrita. O registrador PCL está presente em todos os bancos de memória do PIC.
PCLATH	Utilizado para acesso aos 5 bits mais significativos do contador de programa do PIC (PC). Esse registrador deve ser utilizado somente para escrita, pois a sua leitura não corresponde ao valor atual do PC. O registrador PCLATH está presente em todos os bancos de memória do PIC.
INTCON	Utilizado para o controle das interrupções no PIC. O registrador INTCON está presente em todos os bancos de memória do PIC.
PIR1	Registrador sinalizador de interrupções periféricas (EEPROM, Comparador, USART, CCP PWM, TMR1 e TMR2). O registrador PIR1 está localizado no banco de memória 0.
PIE1	Registrador de controle de interrupções periféricas. Localizado no banco de memória 1.
PORTA	Utilizado para acesso aos pinos externos da porta A. Localizado no banco de memória 0.

PORPB	Utilizado para acesso aos pinos externos da porta B. Localizado nos bancos de memória 1 e 3.
TRISA	Localizado no banco de memória 1, é utilizado para configurar os pinos da porta A. Cada bit do registrador configura o respectivo pino da porta:  Bit = '0' - configura o pino como saída; Bit = '1' - configura o pino como entrada.
TRISB	Localizado nos bancos de memória 1 e 3, é utilizado para configurar os pinos da porta B. Cada bit do registrador configura o respectivo pino da porta:  Bit = '0' - configura o pino como saída; Bit = '1' - configura o pino como entrada.
TMR0	O registrador TMR0 é utilizado para acessar diretamente a contagem do módulo timer 0. Esse registrador pode ser lido ou escrito, o que significa dizer que ele pode também alterar o valor da contagem do timer 0. O registrador TMR0 pode ser encontrado nos bancos de memória 0 e 2.
TMR1L	Tal qual o TMR0, o TMR1L é utilizado para acessar a contagem do timer 1. Nesse caso o TMR1L acessa os 8 bits inferiores do timer 1. O registrador TMR1L está localizado apenas no banco de memória 0.
TMR1H	Acesso aos 8 bits mais significativos do timer 1. O registrador TMR1H está localizado apenas no banco de memória 0.
T1CON	Utilizado para controle do timer 1 (ligado/desligado, prescaler, seleção de clock, etc.). Este registrador é encontrado somente no banco de memória 0.
TMR2	Acesso aos 8 bits do timer 2. Este registrador é encontrado somente no banco de memória 0.
T2CON	Utilizado para controle do timer 2. Este registrador é encontrado somente no banco de memória 0.
PR2	Registrador de período do timer 2 (utilizado na geração de PWM), localizado no banco de memória 1.
EEDATA	Utilizado para acesso aos dados da EEPROM. Este registrador é encontrado somente no banco de memória 1.
EEADR	Utilizado para endereçamento da EEPROM. Este registrador é encontrado somente no banco de memória 1.
EECON1	Localizado no banco de memória 1, é utilizado para controlar as operações com a EEPROM.

EECON2	Localizado no banco de memória 1, é utilizado para "autorizar" a escrita na EEPROM.
CCPR1L	Metade menos significativa do registrador CCPR1. Este registrador é encontrado somente no banco de memória 0.
CCPR1H	Metade mais significativa do registrador CCPR1. Este registrador é encontrado somente no banco de memória 0.
CCP1CON	Registrador de controle do módulo CCP1. Este registrador é encontrado somente no banco de memória 0.
TXREG	Registrador de dado a ser transmitido pela USART. Este registrador é encontrado somente no banco de memória 0.
RCREG	Registrador de dado recebido pela USART. Este registrador é encontrado somente no banco de memória 0.
TXSTA	Registrador de estado da USART. Este registrador é encontrado somente no banco de memória 1.
RCSTA	Registrador de controle da USART. Este registrador é encontrado somente no banco de memória 0.
SPBRG	Registrador de gerador de clock da USART. Este registrador é encontrado somente no banco de memória 1.
CMCON	Registrador de controle dos comparadores analógicos. Este registrador é encontrado somente no banco de memória 0.
VRCON	Registrador de controle do módulo de referência de tensão interna. Este registrador é encontrado somente no banco de memória 1.

No apêndice A, podemos encontrar os diagramas detalhados dos quatro bancos de memória dos chips em estudo.

### 5.3.1 - Registrador STATUS (0x03, 0x83, 0x103 e 0x183)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
JRP	RP1	RP0	TO	PD	Z	DC	C
bit7							bit0

R/W - leitura/escrita  
R - somente leitura  
O valor após o traço é o valor após reset POR.

Figura 5.4

Esse registrador é composto de vários FLAGS (sinalizadores) que nada mais são do que bits utilizados para sinalizar uma condição específica. Em seguida veremos a descrição de cada flag desse registrador detalhadamente:

- C - Carry/Borrow (transporte/empréstimo) - é utilizado para sinalizar a condição de transporte ou empréstimo de bit após uma operação matemática ou lógica.

Podemos dividir o tipo de sinalização do flag C em duas categorias distintas:

## 1. Operações Matemáticas

### a. Adição.

Em operações de adição, o flag C irá indicar se houve transporte de bit, ou seja, se o resultado da operação ultrapassou a capacidade de armazenamento de 8 bits do registrador. Neste caso o bit C estará em nível lógico '1' indicando o transporte de bit.

Se não tiver ocorrido transporte de bit, ou seja, o resultado da operação não ultrapassou a capacidade de armazenamento de 8 bits do registrador, o flag C estará em nível lógico '0'.

A seguir, temos dois exemplos de adição binária: com transporte e sem transporte.

#### 1. $255 \text{ decimal} + 1 \text{ decimal} = 256 \text{ decimal}$

Em binário temos  $11111111(0xFF) + 1(0x01) = 100000000(0x100)$ . Observe que o resultado contém nove bits. Neste caso o registrador irá armazenar apenas os oito primeiros (00000000) e o nono (1) será armazenado no flag C, indicando que houve transporte de bit.

### Observação

Sempre que o resultado de uma soma for maior que 255, o valor armazenado no registrador de destino será igual ao resultado menos 256.

#### 2. $2 \text{ decimal} + 2 \text{ decimal} = 4 \text{ decimal}$

Em binário temos  $00000010(0x02) + 00000010(0x02) = 00000100(0x04)$ . Neste caso, o resultado é um valor de oito bits, que pode ser armazenado no registrador de destino. Assim, o flag C irá conter o valor 0, indicando que não ocorreu transporte de bit.

### b. Subtração:

Em subtrações, a polaridade do flag C é invertida em relação à adição.

Assim, se numa subtração for necessário o empréstimo de bit, o flag C sinalizará esta condição com nível lógico '0'. Se por outro lado não houver empréstimo de bit, o flag C sinalizará tal condição com nível lógico '1'.

O princípio para este comportamento é o seguinte: sempre antes de uma subtração o chip irá setar o bit de carry. Esta operação consiste, na realidade, na criação de uma reserva de 256 para a eventualidade de a subtração necessitar de empréstimo.

Assim, sempre que a operação necessitar de empréstimo para ser concluída, o PIC irá retirar do carry a "reserva" e utilizar o seu valor para concluir a operação.

A seguir, temos dois exemplos de subtração: a primeira com empréstimo de bit e a segunda não.

**1. 255 decimal - 1 decimal = 254 decimal**

Em binário temos  $11111111(0xFF) - 1(0x01) = 11111110(0xFE)$ . Observe que não houve necessidade de emprestar um bit, pois o subtraendo (1) é menor do que o minuendo(255), assim o flag C estará em nível '1' indicando que não houve empréstimo.

**2. 2 decimal - 3 decimal = -1 (na verdade para o PIC é 255) decimal**

Em binário temos  $00000010(0x02) - 00000011(0x03) = 11111111(0xFF)$ . Neste outro exemplo, o subtraendo (3) é maior que o minuendo (2), resultando em uma resposta negativa. Neste caso, o flag C irá conter o valor 0 indicando o empréstimo de bits.

***Observação***

Sempre que o resultado de uma subtração for negativo, o valor a ser armazenado no registrador de destino será igual ao resultado mais 256.

**2. Operações Lógicas**

Nas operações lógicas de rotação de bit, o flag C irá atuar como um bit adicional do registrador, ou seja, fará com que tenhamos (para efeito da operação) um registrador de 9 bits. O flag C poderá ocupar a posição de bit mais significativo (MSB) ou menos significativo (LSB), dependendo da operação de rotação.

Maiores detalhes sobre as operações de rotação de bit podem ser vistos nas instruções RRF e RLF.

**DC** - Digit Carry/Digit Borrow (Transbordo de dígito/Empréstimo de dígito) - o flag DC possui a mesma função do flag C, mas enquanto o flag C refere-se a um registrador de 8 bits, o flag DC refere-se a um conjunto de 4 bits (nibble) do registrador (ou um dígito do registrador, se considerarmos que um registrador de 8 bits pode guardar dois dígitos hexadecimais). Observe os exemplos seguintes:

**1. 15 decimal + 2 decimal = 17 decimal**

Em binário teremos  $00001111(0xF) + 00000010 (0x02) = 00010001(0x11)$ . Como podemos ver, a soma excedeu o primeiro dígito do registrador (4 primeiros bits), e como se trata de uma soma, teremos que o flag DC terá nível 1.

**2. 16 decimal - 1 decimal = 15 decimal**

Em binário teremos 00010000(0x10) - 00000001(0x01) = 00001111(0xF). Neste caso, a subtração fez com que fosse necessário emprestar um bit do dígito mais significativo; tal fato é sinalizado com DC=0.

Observe que na subtração, o DC possui polaridade oposta à da soma (como ocorre com o C).

**Z** - Zero - este flag é utilizado para sinalizar que uma operação resultou em zero, ou seja, que o valor zero foi armazenado no registrador de destino.

Se a operação resultou em '0', o flag Z do registrador STATUS é ativado (nível '1'); se a operação resultou em um número diferente de '0', o flag Z é apagado (nível '0').

### **Observação**

Nem todas as instruções irão afetar ou alterar o estado do flag Z (como também o C e o DC). O mesmo somente será alterado pelas instruções que explicitamente o alterem indiretamente, ou por instruções cujo destino direto seja o registrador STATUS.

**PD** - Power Down (desligamento) - este flag é utilizado para sinalizar o estado do controle de energia do processador.

Se o mesmo estiver em nível '0', indicará que o processador executou uma instrução SLEEP e entrou em modo de baixa potência.

Se o flag estiver em nível '1', indicará que, ou o processador não entrou em modo de baixa energia desde que foi ligado, ou foi executada uma instrução CLRWD (a qual coloca os flags PD e TO em nível '1', além de ressetar o watchdog).

**TO** - Time Out (estouro de tempo) - indica o estado em que se encontra a contagem do watchdog.

Se o flag estiver em nível '0', estará indicando que o watchdog chegou ao final da contagem e provocou um reset no processador.

Se estiver em nível '1', estará indicando que o watchdog ainda não atingiu a contagem final, ou foi executada uma instrução CLRWD.

Note que se o watchdog estiver desabilitado, este flag estará sempre em nível lógico '1'.

**RP0 e RP1** - Selecionam o banco de memória que a CPU irá utilizar, conforme a tabela seguinte:

<b>RP1</b>	<b>RP0</b>	<b>Banco selecionado</b>
0	0	Banco 0
0	1	Banco 1

RP1	RP0	Banco selecionado
1	0	Banco 2
1	1	Banco 3

**IRP** - bit de seleção de banco de memória para endereçamento indireto.

IRP	Banco selecionado
1	banco 2 e 3 (0x100h a 0x1FFh)
0	banco 0 e 1 (0x000h a 0x0FFh)

### *Observação*

O registrador STATUS pode ser acessado em qualquer dos 4 bancos de memória existentes no PIC.

### 5.3.2 - Registrador OPTION\_REG (0x81 e 0x181)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS2	PS0
bit7							bit0

R/W - leitura/escrita  
R - somente leitura  
O valor após o traço é o valor após reset POR.

*Figura 5.5*

O registrador OPTION\_REG é responsável pela configuração de algumas funções internas do PIC. A seguir, veremos a descrição detalhada de seus bits:

**PS0, PS1, PS2** são os bits responsáveis pela configuração do prescaler utilizado com o timer 0 ou com o watchdog. Na verdade, o prescaler nada mais é do que um contador binário, que funciona como divisor programável. A finalidade do prescaler é muito simples: dividir o sinal de entrada (clock) por um valor determinado, de forma a possibilitar a contagem de valores muito grandes, ou cuja freqüência seja muito alta. Na tabela seguinte temos os valores de pré-divisão do prescaler:

PS2	PS1	PS0	Timer0 (PSA=0)	Watchdog (PSA=1)
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

Devemos observar que o prescaler pode estar conectado ao timer 0 ou ao watchdog, daí a existência das duas colunas: a primeira para quando o prescaler estiver conectado ao timer 0 e a segunda para quando o prescaler estiver conectado ao watchdog.

**PSA** - Este bit permite selecionar se o prescaler está conectado ao timer 0 ou ao watchdog:

- 0 - O prescaler está conectado ao timer 0;
- 1 - O prescaler está conectado ao watchdog.

**T0SE** - Seleção de borda de sensibilidade do clock externo do timer 0:

- 0 - O timer 0 será incrementado na borda de subida do sinal externo aplicado em RA4/T0CKI;
- 1 - O timer 0 será incrementado na borda de descida do sinal externo aplicado em RA4/T0CKI.

**T0CS** - Seleção de entrada de clock para o timer 0:

- 0 - O clock do timer será proveniente do clock interno do sistema ( $F_{osc}/4$ );
- 1 - O clock do timer 0 será obtido externamente pelo pino RA4/T0CKI.

**INTEDG** - Seleção da borda de ativação da interrupção externa RB0/INT:

- 0 - Poderá ocorrer interrupção na borda de descida do sinal no pino RB0/INT;
- 1 - Poderá ocorrer interrupção na borda de subida do sinal no pino RB0/INT;

**RBPU** - Ativação dos resistores de pull-up internos da porta B:

- 0 - Resistores de pull-up ativados (apenas para os pinos configurados como entrada);
- 1 - Resistores de pull-up desativados.

### 5.3.3 - Registrador PCON (0x8E)

U-0	U-0	U-0	U-0	R/W-1	U-0	R/W-q	R/W-q
bit7				OSCF		POR	BOD

R/W - leitura/escrita  
R - somente leitura  
O valor após o traço é o valor após reset POR.

Figura 5.6

O registrador PCON, localizado no banco de memória 1, é utilizado para sinalização de dois estados de reset: o reset por queda de tensão de alimentação (brown-out) e o reset de inicialização (Power-on reset). Além disso, o bit 3 desse registrador controla o clock do chip quando nos modos de oscilador interno ou resistor externo:

**OSCF** - Bit de controle do modo de operação do oscilador de clock interno:

- **0** - Clock de 37Khz (somente nos modos de clock interno ou resistor externo);
- **1** - Clock de 4Mhz interno ou conforme resistor externo.

**POR** - Sinalizador de reset de inicialização:

- **0** - Houve um reset do tipo POR (reset de inicialização);
- **1** - Não aconteceu reset de inicialização.

**BOR** - Sinalizador de reset por queda de tensão de alimentação (brown-out):

- **0** - Houve reset por queda de tensão de alimentação;
- **1** - Não houve reset por queda de tensão de alimentação.

## 5.4 - Interrupções

O PIC em questão possui dez fontes possíveis de interrupção:

- ◆ Timer 0;
- ◆ Timer 1;
- ◆ Timer 2;
- ◆ Mudança no PORTB;
- ◆ Interrupção externa INT;
- ◆ Término de escrita na EEPROM;
- ◆ Comparaçao ou captura no módulo CCP;
- ◆ Recepção de dados na USART;
- ◆ Fim da transmissão de dados na USART;
- ◆ Comparador analógico.

Para manter o maior nível possível de compatibilidade com o software existente, o fabricante optou por manter a estrutura já utilizada nas arquiteturas anteriores (e que se baseava no registrador INTCON como controle de interrupções).

Desta forma, mantendo as três principais interrupções no registrador INTCON, optou-se por criar as chamadas interrupções periféricas, que são acessadas por meio de registradores independentes, mas estão subordinadas ao INTCON pelo flag PEIE. Assim, utilizam-se três registradores para o controle de interrupções:

- ◆ INTCON - para controle e sinalização das interrupções do TMR0, mudança no PORTB, interrupção externa e interrupção de periféricos;
- ◆ PIR1 - para sinalização de interrupções periféricas;
- ◆ PIE1 - para controle das interrupções periféricas.

Na figura 5.7, podemos verificar a estrutura lógica do funcionamento das interrupções no PIC 16F62x:

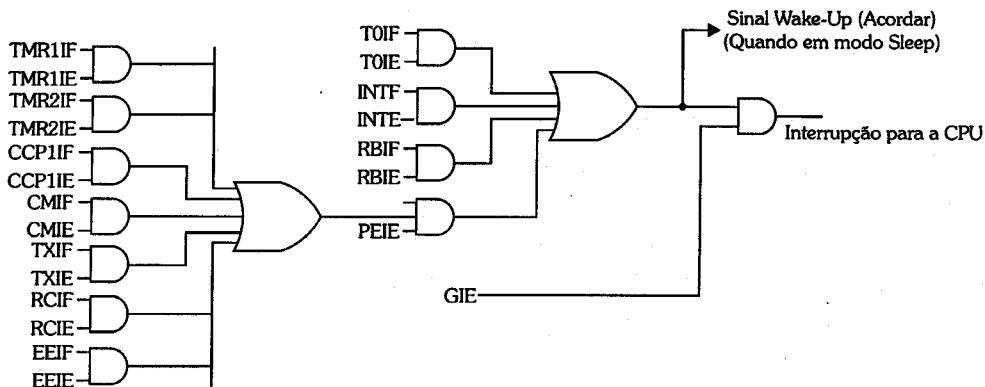


Figura 5.7

Lembrando que o flag de interrupção é setado toda vez que ocorre um evento de interrupção (por exemplo: transbordo do timer 0, transbordo do timer 1, mudança na porta B, interrupção externa INT, etc.).

Já o bit de habilitação de interrupção destina-se a controlar cada interrupção individualmente: se o bit de habilitação estiver em nível '0', a interrupção estará desativada, ou seja, não será capaz de interromper o programa nem "acordar" a CPU.

Observe que as interrupções individuais são combinadas em uma porta do tipo OR. **O sinal para acordar a CPU não depende do GIE**, mas a interrupção da CPU somente irá ocorrer se GIE = '1'.

### Observação

Os flags de interrupção (TOIF, INTF, RBIF, TMR1F, TMR2IF, CCP1IF, CMIF, EIF, TXIF e RCIF) são setados independentemente do estado dos bits de habilitação individual de interrupção. Por exemplo: se TOIE='0' (interrupção de transbordo do timer 0 desligada), ainda assim, a cada transbordo do timer 0 o TOIF será setado (e ficará setado até o programador zerá-lo), mas como TOIE está desligado, isto não provocará uma interrupção do processador.

Observe que alguns flags de interrupção (como o RBIF e o CMIF) podem ser apagados simplesmente pela leitura de um registrador específico (o PORTB no caso do RBIF e o CMCON no caso do CMIF).

Devemos também lembrar que durante o tratamento de uma interrupção, podem ser alterados valores de registradores utilizados pelo programa que foi interrompido (como o W e o STATUS, por exemplo). Neste caso é imperioso que se armazene o conteúdo dos registradores que serão alterados pela interrupção em registradores temporários e depois, seja retornado seu conteúdo original, **antes de retornar da interrupção**.

As interrupções podem também ser utilizadas para "acordar" o PIC de um modo SLEEP.

Neste caso, é necessário que, antes de entrar em modo SLEEP, o programa tenha habilitado alguma interrupção individual.

Após isso, caso aconteça um evento de interrupção cujo flag de habilitação esteja ligado, teremos um evento de wake-up (conforme podemos observar pela análise da figura 18) que fará com que a CPU retorne ao seu funcionamento normal, conforme discutido no 3.3.2.

### 5.4.1 - Registrador INTCON (0x0B)

R/W-0	R/W-X						
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit7							bit0

R/W - leitura/escrita  
 R - somente leitura  
 O valor após o traço é o valor após reset POR.

Figura 5.8

O registrador INTCON é o responsável pelo controle de interrupções no PIC. Podemos dividir o INTCON em três blocos básicos: o controle geral de interrupções (GIE), os controles individuais de interrupções (EEIE, TOIE, INTE e RBIE) e os flags sinalizadores da ocorrência da interrupção (TOIF, INTF e RBIF). Em seguida veremos explicações mais detalhadas sobre os bits do INTCON:

- GIE** - Global Interrupt Enable/Habilitação Global de Interrupções - o bit GIE é o controle geral de interrupções no PIC. Ele funciona como a chave principal de controle de interrupções:
- **0** - Se estiver desligado, não poderá ocorrer nenhuma interrupção no chip (mesmo que alguma interrupção individual esteja habilitada);
  - **1** - Se estiver ligado (e alguma das interrupções individuais também estiver habilitada), **poderá** ocorrer interrupção.

**PEIE** - Habilitação das interrupções periféricas:

- 0** - Interrupções periféricas desligadas;
- 1** - Interrupções periféricas habilitadas.

**TOIE** - Habilitação de interrupção de trasbordamento do timer 0:

- 0** - Interrupção de transbordo do timer 0 desligada;
- 1** - Interrupção de transbordo do timer 0 habilitada.

**INTE** - Habilitação de interrupção externa (RB0/INT):

- 0** - Interrupção externa desligada;
- 1** - Interrupção externa habilitada.

**RBIE** - Habilitação de interrupção por mudança na porta B (pinos RB4 a RB7):

- 0 - Interrupção por mudança na porta B desligada;
- 1 - Interrupção por mudança na porta B habilitada.

**T0IF** - flag sinalizador de transbordamento do timer 0:

- 0 - Não houve transbordamento no timer 0 (a contagem dele não superou 255);
- 1 - Houve transbordamento no timer 0 (a contagem dele superou 255) e se o GIE estiver ligado, assim como o T0IE, será gerada uma interrupção neste caso.

**INTF** - Flag sinalizador de interrupção externa (RB0/INT):

- 0 - Não houve interrupção externa no pino RB0/INT;
- 1 - Uma interrupção externa foi acionada pelo pino RB0/INT (dependendo da borda selecionada pelo bit INTEDG do registrador OPTION). Se o GIE estiver ligado, assim como o INTE, será gerada interrupção neste caso.

**RBIF** - Flag sinalizador de alteração nos pinos RB4 a RB7:

- 0 - Não houve alteração nos níveis lógicos dos pinos RB4 a RB7;
- 1 - Houve alteração em algum dos pinos RB4 a RB7. Se o GIE estiver ligado e o RBIE também, será gerada uma interrupção.

### *Observação*

Os sinalizadores de interrupção (RBIF, INTF e T0IF) são setados pelo evento gerador de interrupção, independentemente do respectivo bit de habilitação da interrupção individual (RBIE, INTE ou T0IE) estar ou não habilitado!

## 5.4.2 - Registrador PIE1 (0x8C)

R/W-0	R/W-0	R/W-0	R/W-0	U	R/W-0	R/W-0	R/W-0
EEIE	CMIE	RCIE	TXIE	-	CCP1IE	TMR2IE	TMR1IE
bit7							bit0

R/W - leitura/escrita  
R - somente leitura  
O valor após o traço é o valor após reset POR.

*Figura 5.9*

Como já foi dito, o registrador PIE1 possui a função de habilitação/desabilitação das chamadas interrupções periféricas:

**EEIE** - Habilitação de interrupção por término de escrita na EEPROM interna:

- 0 - Interrupção desabilitada;
- 1 - Interrupção habilitada.

- CMIE** - Habilitação de interrupção de mudança de estado dos comparadores analógicos:
- 0** - Interrupção desabilitada;
  - 1** - Interrupção habilitada.
- RCIE** - Habilitação de interrupção de recepção de dados na USART:
- 0** - Interrupção desabilitada;
  - 1** - Interrupção habilitada.
- TXIE** - Habilitação de interrupção de transmissão da USART:
- 0** - Interrupção desabilitada;
  - 1** - Interrupção habilitada.
- CCPIE** - Habilitação de interrupção do módulo CCP:
- 0** - Interrupção desabilitada;
  - 1** - Interrupção habilitada.
- TMR2IE** - Habilitação de interrupção de transbordo do timer 2:
- 0** - Interrupção desabilitada;
  - 1** - Interrupção habilitada.
- TMR1IE** - Habilitação de interrupção de transbordo do timer 1:
- 0** - Interrupção desabilitada;
  - 1** - Interrupção habilitada.

Observe que o bit 3 do registrador não possui nenhuma função nesse chip, porém o seu uso para outras finalidades é desencorajado pelo fabricante, tendo em vista a compatibilidade futura. Por isso, este bit deve sempre ser mantido em '0'.

Caso se deseje utilizar alguma das interrupções anteriores para "acordar" a CPU, devemos setar o bit PEIE no INTCON. Se desejarmos também que o evento de interrupção seja tratado pela CPU, devemos, além do PEIE, setar também o bit GIE.

### 5.4.3 - Registrador PIR1 (0x0C)

R/W-0	R/W-0	R-0	R-0	U	R/W-0	R/W-0	R/W-0
EEIE	CMIF	RCIF	TXIF	-	CCP1IF	TMR2IF	TMR1IF
bit7							bit0

R/W - leitura/escrita  
 R - somente leitura  
 O valor após o traço é o valor após reset POR.

Figura 5.10

O registrador PIR1 localizado no banco memória 0 abriga os flags sinalizadores de interrupções periféricas. Isto significa que cada vez que um dos eventos descritos em seguida ocorrer, o respectivo flag será setado (isto independe de o respectivo flag de habilitação no PIE1 estar ligado ou desligado):

**EEIF** - Sinalizador de término de escrita na EEPROM:

- 0** - A escrita ainda não terminou ou não foi iniciada;
- 1** - A escrita terminou.

**CMIF** - Sinalizador de mudança de estado na saída dos comparadores analógicos:

- 0** - Não houve mudança de estado nos comparadores;
- 1** - Houve mudança na saída dos comparadores.

**RCIF** - Sinalizador de recepção de caractere na USART:

- 0** - Não houve recepção de novos caracteres;
- 1** - Um novo caractere foi recebido.

**TXIF** - Sinalizador de transmissão na USART:

- 0** - Não houve transmissão na USART, ou o TSR está ocupado;
- 1** - O caractere armazenado no TXREG foi repassado para o TSR.

**CCPIF** - Sinalizador de comparação/captura no módulo CCP:

- 0** - Não houve comparação válida/captura no módulo CCP;
- 1** - Houve comparação/captura no módulo CCP.

**TMR2IF** - Sinalizador de transbordo no timer 2:

- 0** - Não houve transbordo de contagem no TMR2;
- 1** - Houve transbordo de contagem no TMR2.

**TMR1IF** - Sinalizador de transbordo no timer 1:

- 0** - Não houve transbordo de contagem no TMR1;
- 1** - Houve transbordo de contagem no TMR1.

Os eventos geradores de interrupções periféricas serão estudados um a um quando estudarmos os periféricos internos do PIC 16F62x.

### ***Observação***

Os sinalizadores de interrupção periféricas são setados pelo evento gerador de interrupção, independentemente do respectivo bit de habilitação da interrupção individual no registrador PIE1 estar ou não habilitado!

Os flags RCIF e TXIF somente podem ser lidos, mas não podem ser escritos pelo software. Para modificar o conteúdo destes flags, é necessário acessar determinados registradores da USART, conforme veremos adiante.

## 5.5 - Pinos de I/O

Como já foi dito, o PIC 16F62x possui duas portas de entrada/saída (I/O): porta A e porta B. Ambas as portas possuem oito pinos, sendo que na porta A temos sete pinos de entrada/saída e um somente de entrada; na porta B, todos os oito pinos podem ser configurados como entradas ou saídas.

Cada porta de I/O possui dois registradores que controlam suas funções: um registrador PORT e um registrador TRIS.

O registrador PORT é utilizado para acesso aos pinos do MCU: uma escrita no registrador PORT escreve nos pinos que estiverem configurados como saída.

Uma leitura do registrador PORT irá ler o nível lógico externo presente nos pinos de I/O.

Observe que mesmo estando o pino configurado como saída, ainda podemos efetuar a leitura dele, caso em que iremos ler o próprio dado de saída do pino.

O registrador TRIS é utilizado para configurar cada pino da respectiva porta como entrada ou saída. Assim, cada bit do registrador corresponde a um pino da porta. Se o bit estiver em '1', configura o pino como entrada e se estiver em '0' como saída.

### 5.5.1 - Porta A

A primeira das portas a ser estudada é a A. Além das funções de entrada/saída, encontramos também diversas outras funções multiplexadas aos pinos da porta A:

RA0/AN0	Pino de entrada/saída e entrada analógica AN0 para o módulo comparador;
RA1/AN1	Pino de entrada/saída e entrada analógica AN1 para o módulo comparador;
RA2/AN2/Vref	Pino de entrada/saída, entrada analógica AN2 ou saída de tensão de referência do módulo Vref;
RA3/AN3/CMP1	Pino de entrada/saída, entrada analógica AN3 ou saída do comparador 1;
RA4/T0CKI/CMP2	Pino de entrada/saída, entrada de clock p/timer 0 ou saída do comparador 2;
RA5/ <u>MCLR</u> /THV	Pino de entrada, reset da CPU ou tensão de programação;

- RA6/OSC2/CLKOUT Pino de entrada/saída, entrada 2 do oscilador de clock do chip ou saída de clock ( $F_{osc}/4$ );
- RA7/OSC1/CLKIN Pino de entrada/saída, entrada 1 do oscilador de clock do chip ou entrada de clock externo.

Utilizam-se dois registradores para o acesso e controle da porta A, sendo eles:

### 5.5.1.1 - Registrador PORTA (0x05)

O registrador PORTA é utilizado para a escrita/leitura dos pinos externos da porta A (RA0 a RA7).

Cada pino é acessado pelo respectivo bit do registrador. Assim, o bit 0 do registrador PORTA irá acessar o pino RA0, o bit 1 o pino RA1 e assim por diante até o bit 7 que irá acessar o pino RA7.

Observe que toda porta possui um *latch* de saída para cada pino. Desta forma, uma escrita no registrador PORTA irá na realidade escrever em cada *latch* do PORTA.

Já uma leitura do registrador PORTA irá efetivamente ler o conteúdo dos pinos da porta A.

Observe ainda que o registrador PORTA somente pode ser acessado a partir do banco de memória 0.

### 5.5.1.2 - Registrador TRISA (0x85)

O registrador TRISA é utilizado para o controle da direção de atuação de cada pino de I/O da porta A.

Se o bit do registrador TRISA estiver em nível lógico '1', o pino estará configurado como entrada.

Se o bit do registrador TRISA estiver em nível lógico '0', o pino estará configurado como saída.

Assim, se o bit 0 do TRISA estiver em nível lógico '0', significa que o pino RA0 está configurado como saída.

Observe que se fizermos uma leitura de um pino configurado como saída, vamos ler na realidade o valor armazenado no *latch* de saída do pino.

Se ao contrário escrevermos em um bit configurado como entrada, o valor será armazenado no *latch* de saída da porta A, mas não vai aparecer no pino, até que se configure o pino como saída.

O registrador TRISA é encontrado apenas no banco 1 de memória.

### 5.5.1.3 - Configuração Interna

Nas figuras seguintes, temos a estrutura interna de cada pino da porta A.

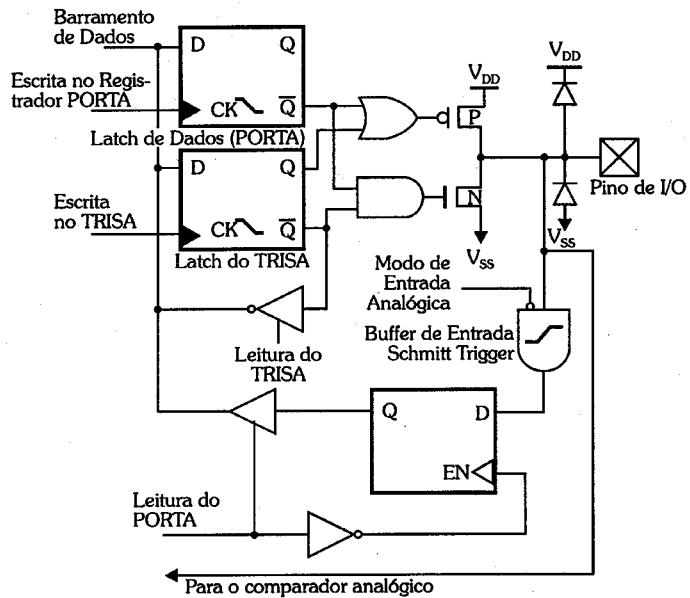


Figura 5.11 - Pinos RA0 e RA1.

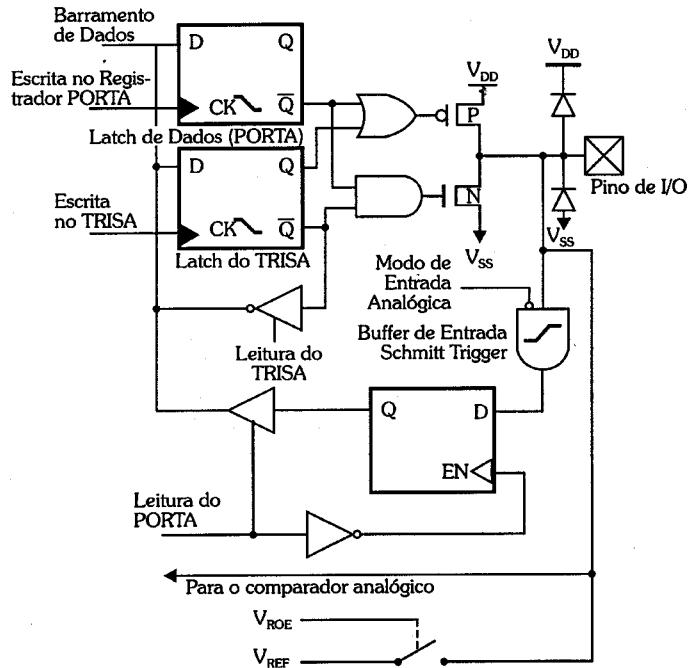


Figura 5.12 - Pino RA2.

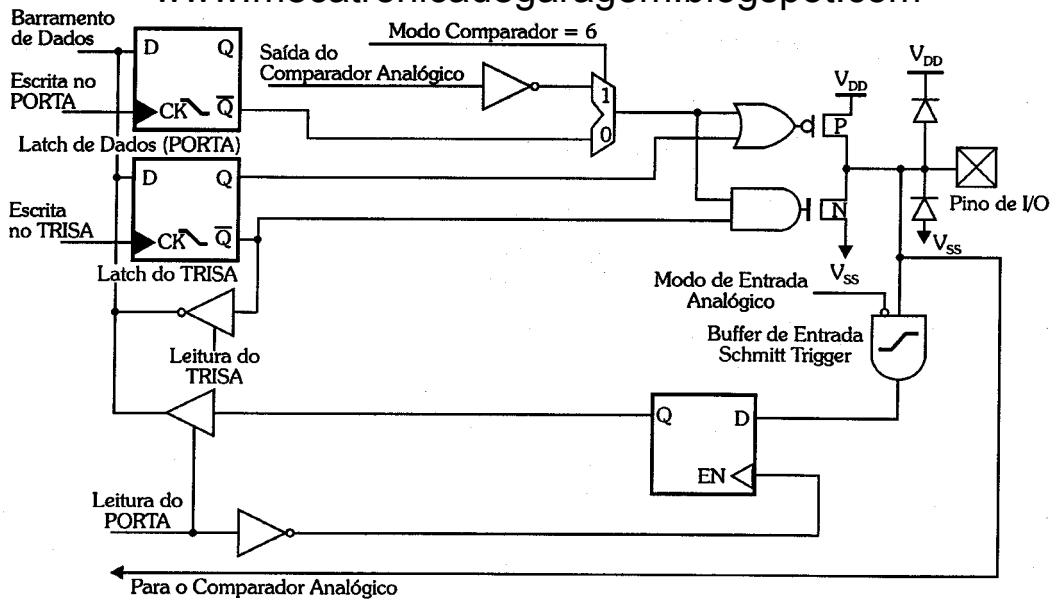


Figura 5.13 - RA3.

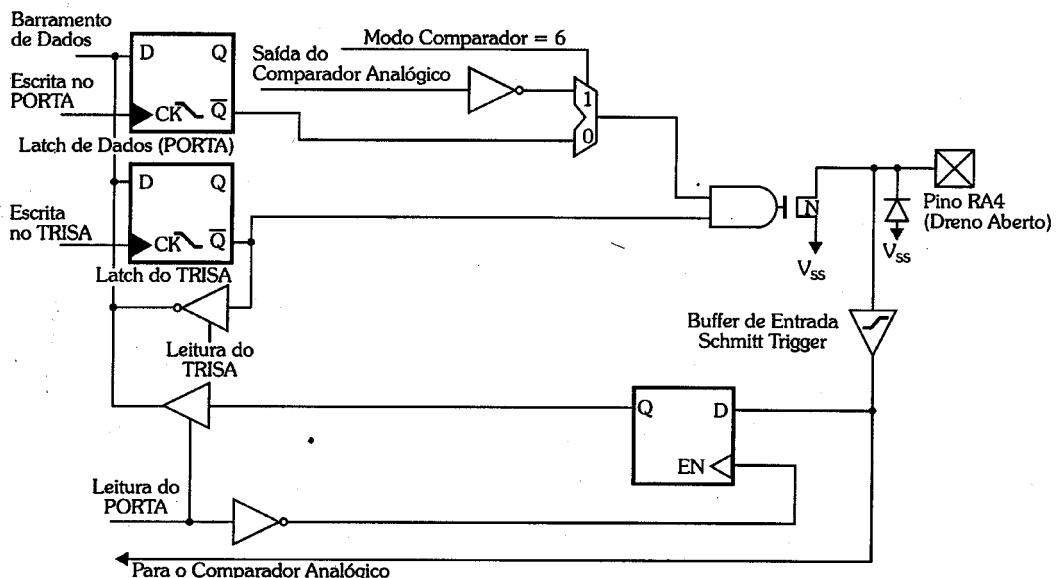


Figura 5.14 - Pino RA4.

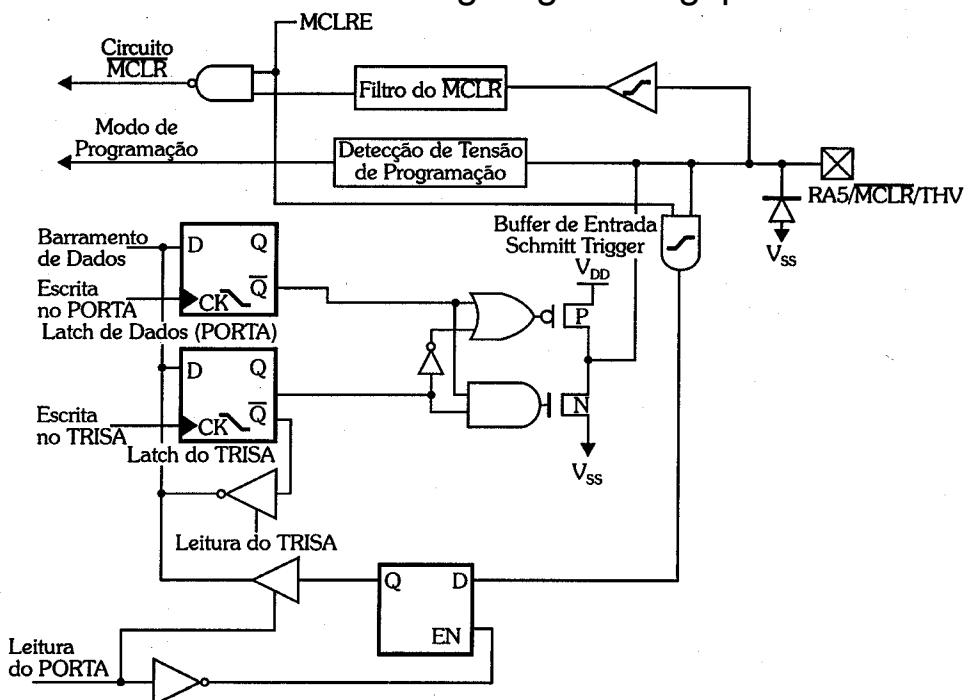


Figura 5.15 - Pino RA5.

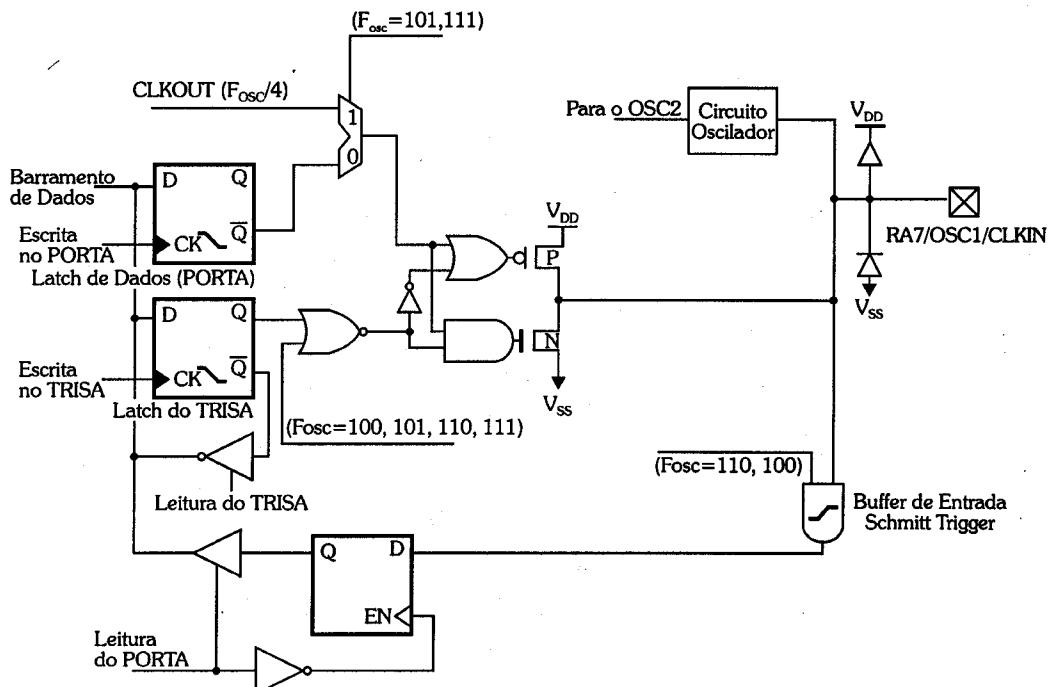
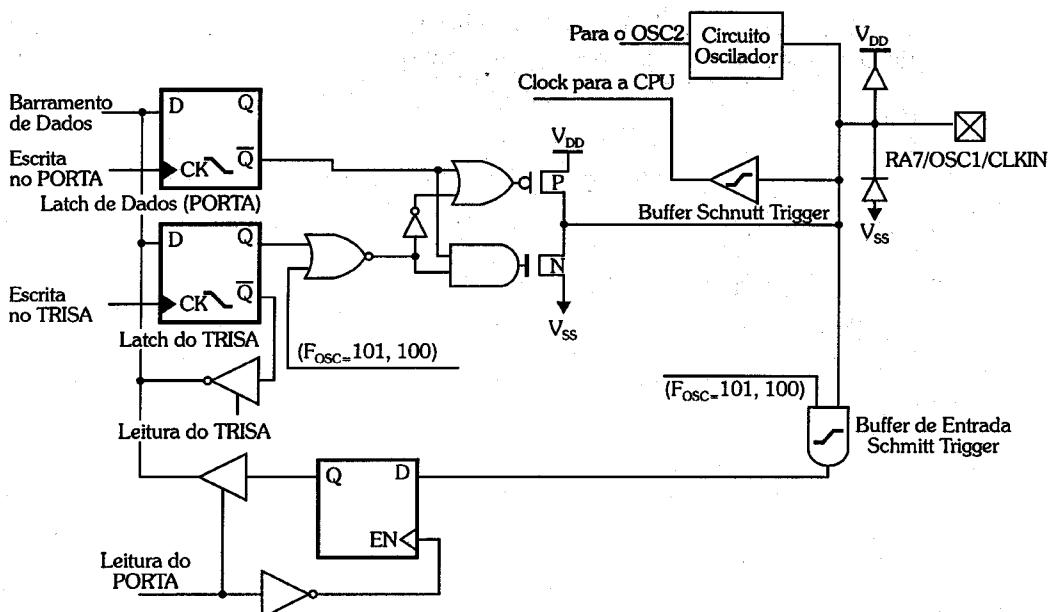


Figura 5.16 - Pino RA6.



**Figura 5.17 - Pino RA7.**

#### 5.5.1.4 - Observações sobre a Porta A

1. Dependendo do modo de funcionamento configurado para o módulo comparador e o módulo de referência de tensão, podemos utilizar ou não os pinos RA0/AN0, RA1/AN1, RA2/AN2/Vref e RA3/AN3 para I/O digital;
2. Sempre que utilizarmos os comparadores analógicos, deveremos, além de configurar o registrador CMCON, configurar também os respectivos pinos (RA0 a RA3) como entradas, ou seja, setar os respectivos bits do TRISA;
3. Uma leitura da porta A com um ou mais pinos configurados como entrada analógica fará com que o respectivo bit seja lido como um nível '0';
4. Se desejarmos utilizar as saídas dos comparadores externamente (por meio de RA3 e RA4), deveremos, além de configurar adequadamente o registrador CMCON, apagar os respectivos bits do TRISA;
5. Se for utilizada a saída de Vref externa, devemos configurar, além do registrador VRCON, o bit 2 do TRISA como entrada ('1');
6. Dependendo do modo de oscilador que utilizarmos, poderemos ter o pino RA6, ou RA6 e RA7 disponíveis para entrada/saída. Caso contrário, o respectivo bit do TRISA ficará sempre em '0', neste caso, o valor que será lido no PORTA será sempre '0';
7. O pino RA5/MCLR /THV somente poderá ser utilizado como entrada se o bit MCLRE estiver em '0' na palavra de configuração do chip.

8. Os pinos RA6 e RA7 podem ser utilizados para I/O em alguns modos de funcionamento do oscilador. Este modo é selecionado na palavra de configuração do MCU no instante em que o estamos gravando (veja página 189).

**Um nível '0' no TRISA configura o pino como saída.**

**Um nível '1' no TRISA configura o pino como entrada.**

Para acesso às funções da porta A, são utilizados os seguintes registradores:

- ◆ **PORTA** (0x05) - para leitura ou escrita de dados nos pinos da porta A;
- ◆ **TRISA** (0x85) - para configuração dos pinos da porta A como entrada ou saída;
- ◆ Os registradores de controle de periféricos: **CMCON** (0x1F) e **VRCON** (0x9F).

#### **Atenção**

Durante o funcionamento normal do CHIP, não devemos ultrapassar o valor de  $V_{DD}$  para qualquer dos pinos da portas do PIC, sob pena de danificá-lo irremediavelmente. A única exceção é o pino RA5, que é utilizado também para o modo de programação do chip (o que é feito com tensões de até 12 Volts).

### **5.5.2 - Porta B**

A porta B, tal qual a porta A, implementa diversas outras funções multiplexadas aos seus pinos:

RB0/INT	Pino de entrada/saída, entrada de interrupção externa INT;
RB1/RX/DT	Pino de entrada/saída, entrada de recepção serial da USART, entrada/saída de dados síncronos;
RB2/TX/CK	Pino de entrada/saída, saída de dados seriais da USART, clock síncrono;
RB3/CCP1	Pino de entrada/saída, entrada/saída do módulo CCP;
RB4/PGM	Pino de entrada/saída, entrada de programação por baixa tensão (LVP), interrupção por mudança de estado;
RB5	Pino de entrada/saída, interrupção por mudança de estado;
RB6/T1OSO/T1CKI	Pino de entrada/saída, entrada 1 de oscilador externo do timer 1, entrada de clock externo do timer 1, interrupção por mudança de estado;

RB7/T1OSI

Pino de entrada/saída, entrada 2 de oscilador externo do timer 1, interrupção por mudança de estado.

Outra facilidade encontrada na porta B são os resistores de pull-up internos: cada pino individualmente possui um MOSFET interno que, quando acionado, comporta-se como um resistor conectando o pino ao  $+V_{DD}$ . Esses MOSFETs são acionados pelo sinal RBPU no registrador OPTION\_REG. Observe que para que os pull-ups funcionem, é necessário que o pino esteja configurado como entrada; caso contrário, o MOSFET de pull-up daquele pino permanecerá desligado!

A corrente típica nos resistores de pull-up é de aproximadamente 200  $\mu A$ .

Como já visualizado anteriormente, encontramos na porta B duas fontes de interrupção:

- INT interrupção externa: a cada transição de '0' para '1' ou '1' para '0', teremos acionado o flag INTF indicando a ocorrência do evento de interrupção externa. A borda de subida ou descida é selecionada pelo bit INTEDG do registrador OPTION\_REG;
- RB interrupção por mudança de nível lógico nos pinos RB4 a RB7: a cada mudança de nível lógico nestes pinos, é gerado um sinal RBIF indicativo do evento. **Uma leitura do PORTB limpa o flag RBIF.**

#### ***Observação***

Somente os pinos configurados como entrada são considerados na geração do sinal RBIF.

#### **5.5.2.1 - Registrador PORTB (0x06 e 0x106)**

O registrador PORTB é utilizado para a escrita/leitura dos pinos externos da porta B (RB0 a RB7).

Cada pino é acessado pelo respectivo bit do registrador. Assim, o bit 0 do registrador PORTB irá acessar o pino RB0, o bit 1 o pino RB1 e assim por diante até o bit 7 que irá acessar o pino RB7.

Observe que toda porta possui um *latch* de saída para cada pino. Desta forma, uma escrita no registrador PORTB irá na realidade escrever em cada *latch* da porta B.

Já uma leitura do registrador PORTB irá efetivamente ler o conteúdo dos pinos da porta B.

Observe ainda que o registrador PORTB pode ser acessado a partir do banco de memória 0 e também no banco de memória 2.

#### **5.5.2.2 - Registrador TRISB (0x86 e 0x186)**

O registrador TRISB é utilizado para o controle da direção de atuação de cada pino de I/O da porta B.

Se o bit do registrador TRISB estiver em nível lógico '1', o pino estará configurado como entrada.

Se o bit do registrador TRISB estiver em nível lógico '0', o pino estará configurado como saída.

Assim, se o bit 0 do TRISB estiver em nível lógico '0', significa que o pino RB0 está configurado como saída.

Observe que se fizermos uma leitura de um pino configurado como saída, vamos ler na realidade o valor armazenado no *latch* de saída do pino.

Se ao contrário escrevermos em um bit configurado como entrada, o valor será armazenado nos *latches* de saída da porta B, mas não irão aparecer nos pinos, até que se configure os pinos como saída.

O registrador TRISB pode ser acessado a partir do banco de memória 1 e também no banco de memória 3.

### 5.5.2.3 - Configuração Interna

Nas figuras seguintes temos a estrutura interna dos pinos da porta B.

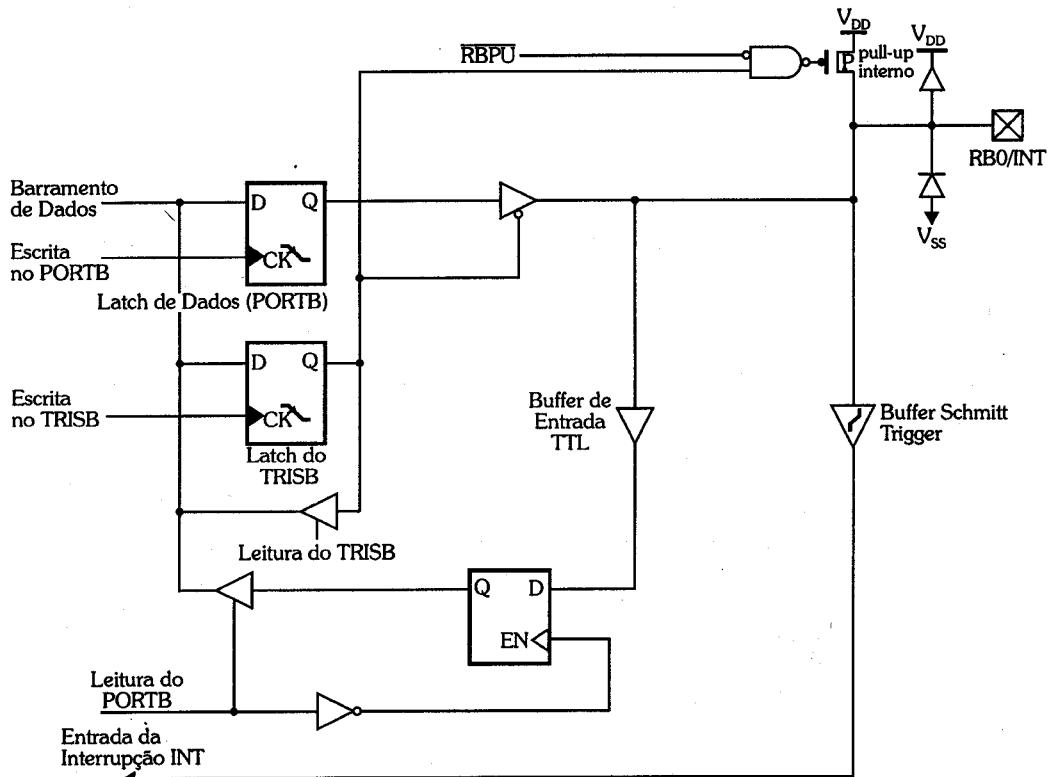


Figura 5.18 - Pino RB0.

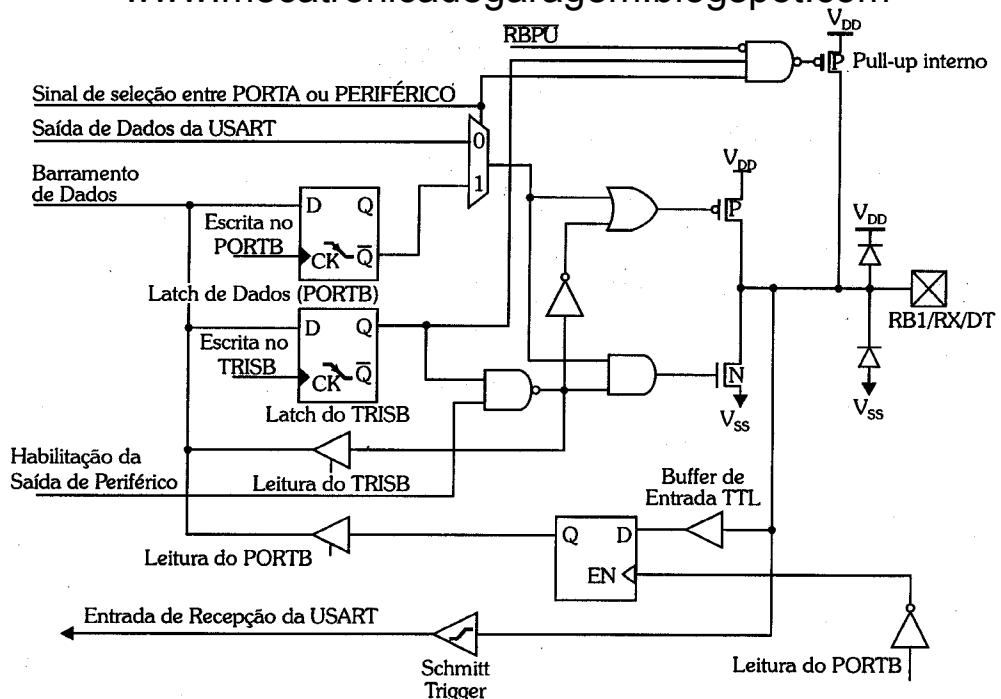


Figura 5.19 - Pino RB1.

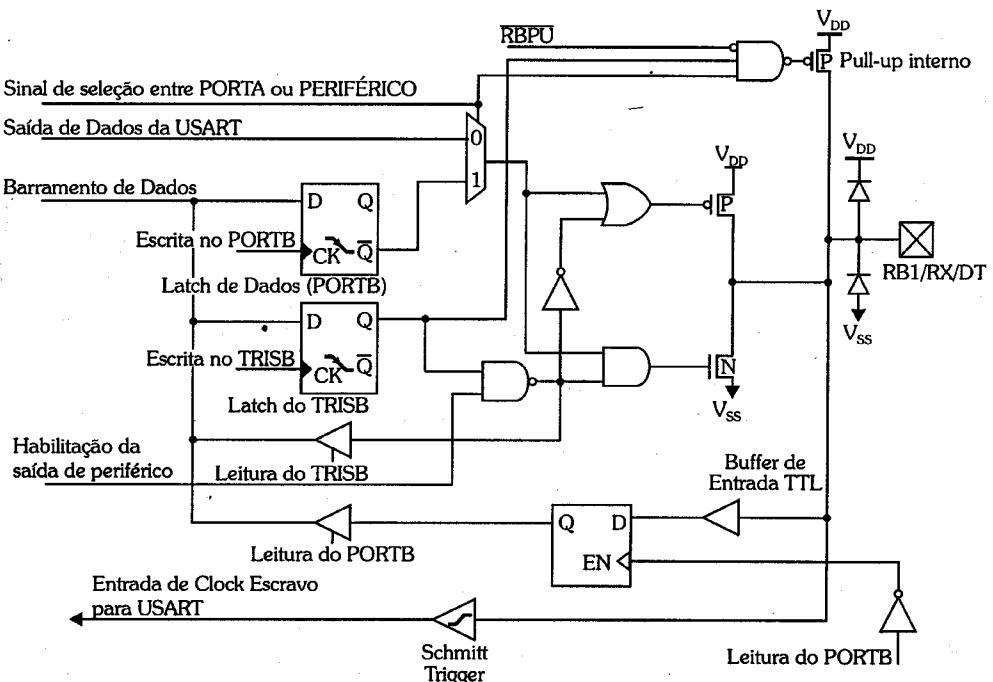


Figura 5.20 - Pino RB2.

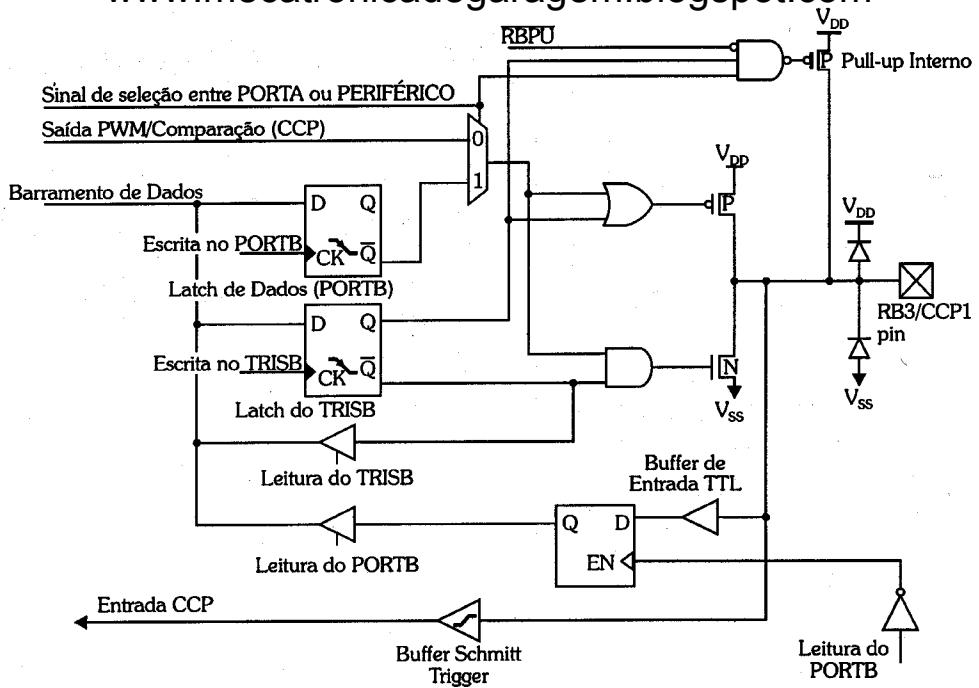


Figura 5.21 - Pino RB3.

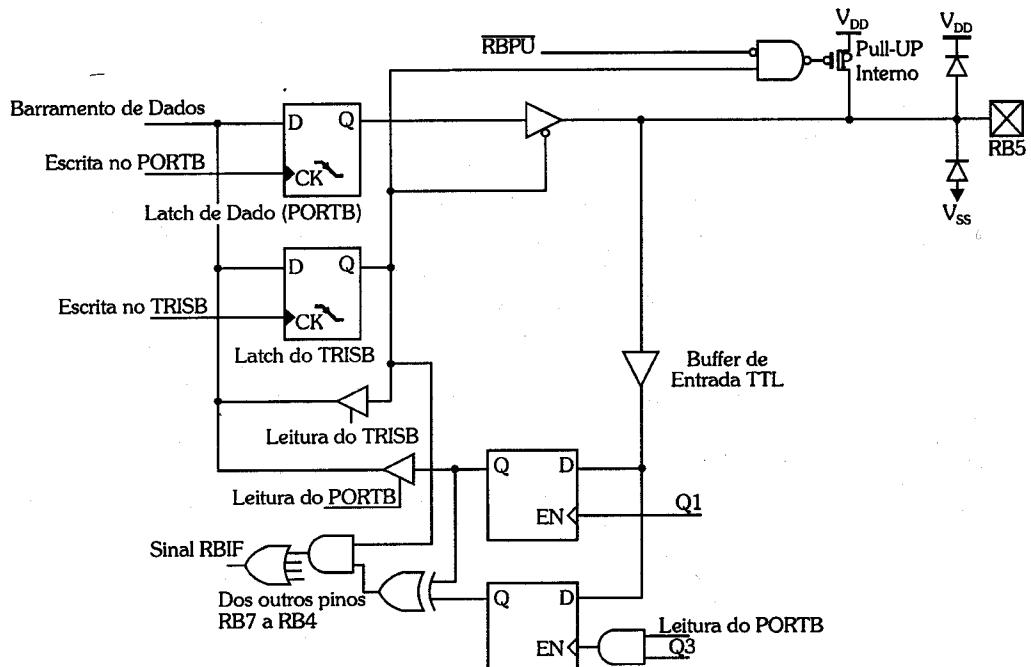


Figura 5.22 - Pino RB5.

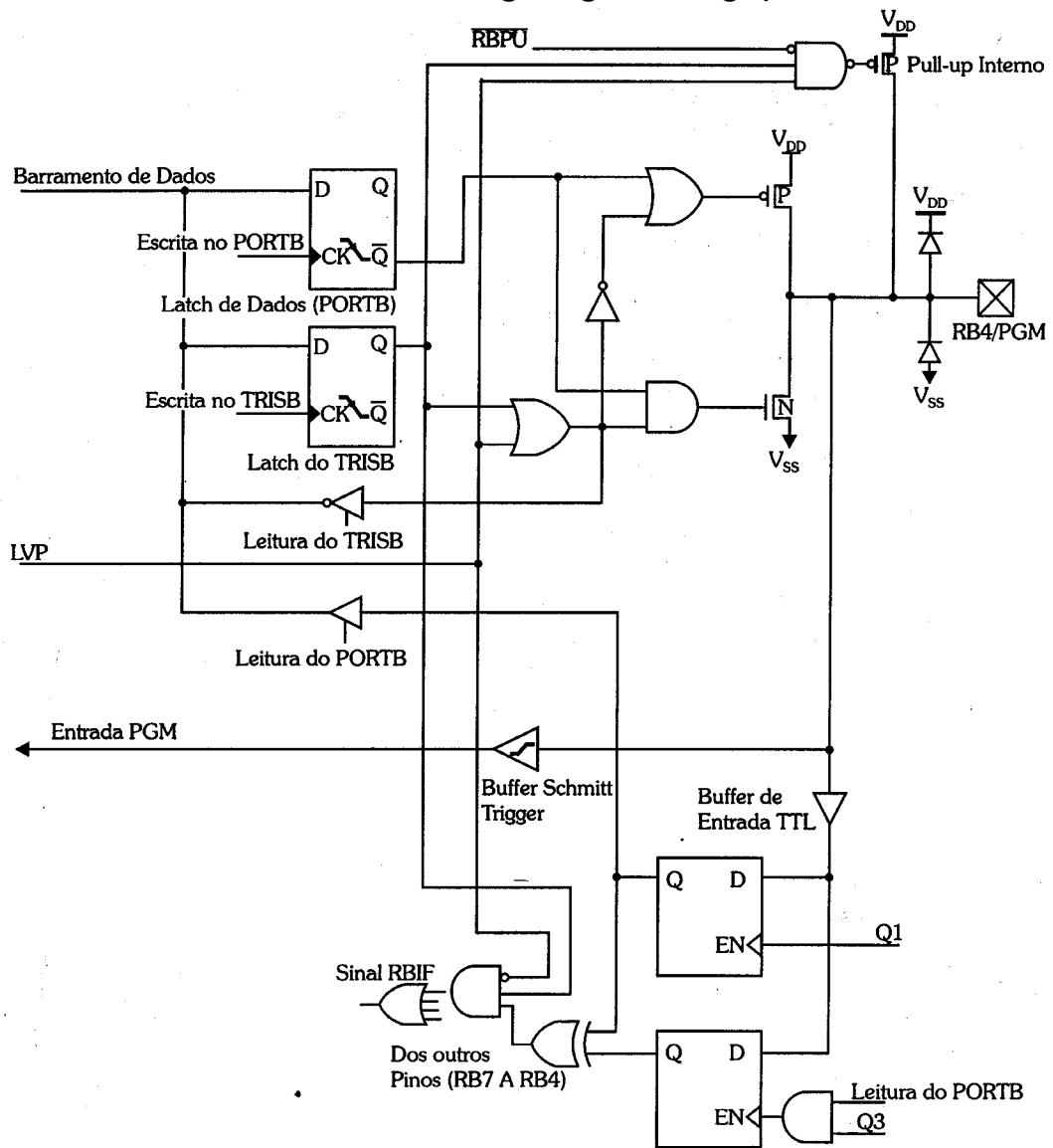


Figura 5.23 - Pino RB4.

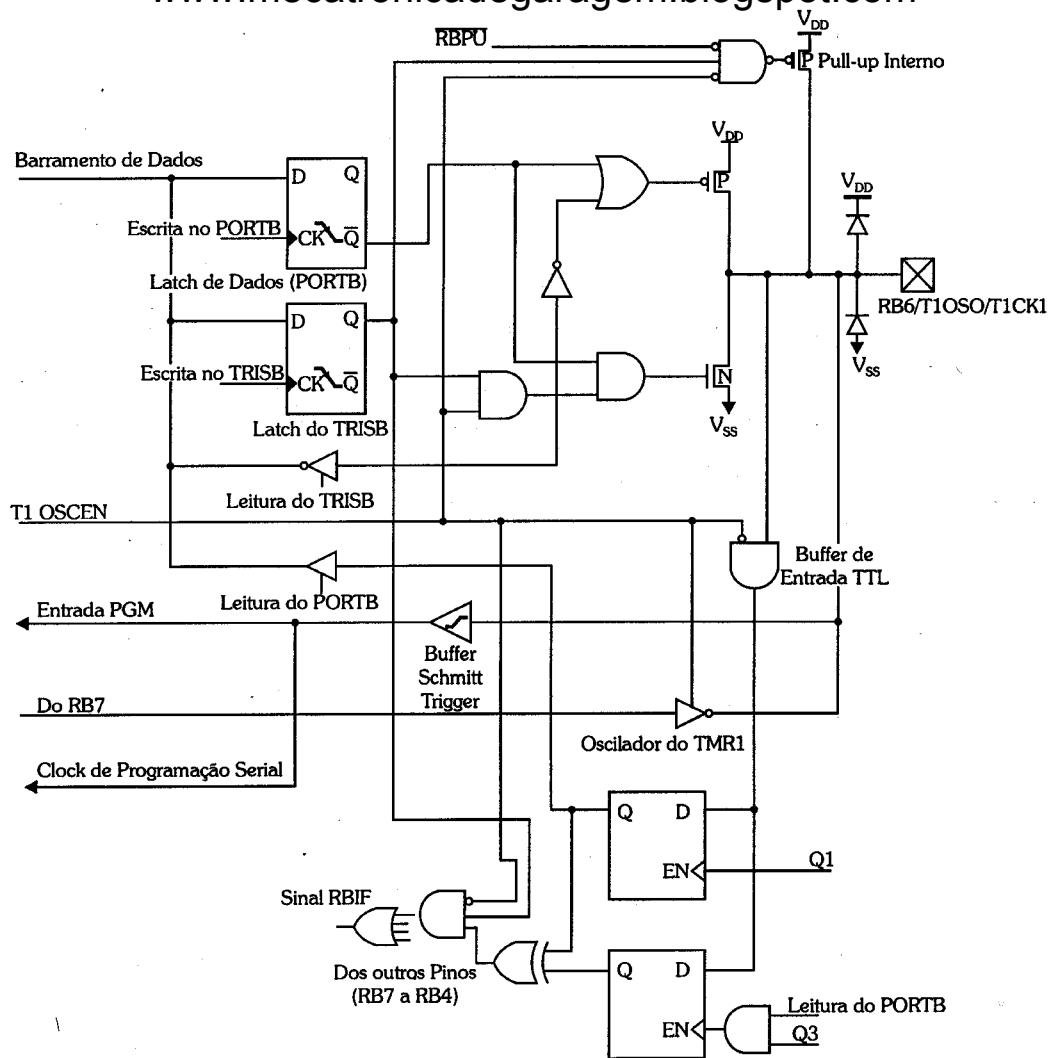
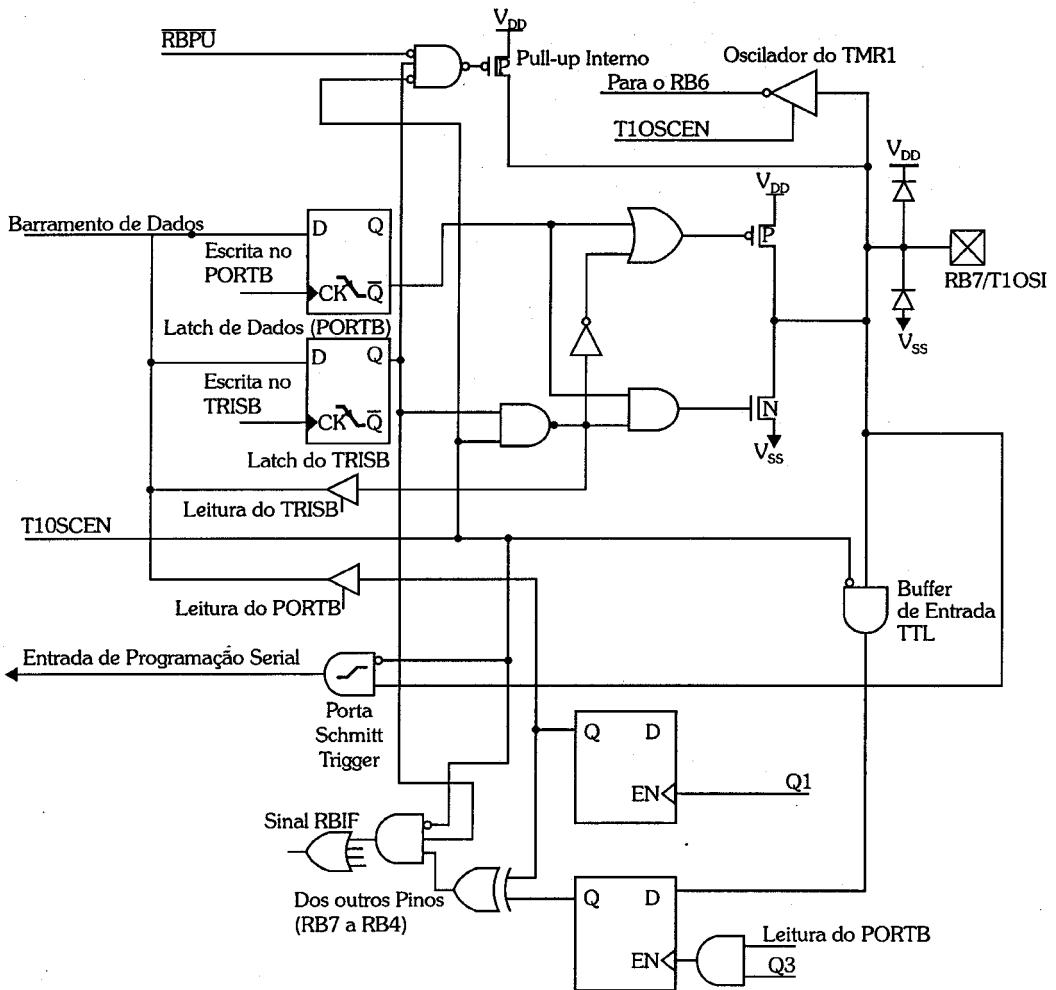


Figura 5.24 - Pino RB6.



**Figura 5.25 - Pino RB7.**

#### 5.5.2.4 - Observações sobre a Porta B

1. O registrador PORTB está localizado nos endereços 06h (banco 0) e 106h (banco 3);
2. O registrador TRISB está localizado nos endereços 86h (banco 1) e 186h (banco 3);
3. Observe que de acordo com a função desejada para o pino do chip, devemos programar o respectivo bit do TRISB. Devemos também considerar os periféricos multiplexados em cada pino, já que muitas vezes o acionamento do periférico automaticamente inibe o valor programado no TRISB;

- Se o chip for programado para aceitar o modo de programação por baixa tensão (bit LVP na palavra de configuração), o pino RB4/PGM não poderá ser utilizado para I/O, pois permanecerá dedicado como entrada de programação.

**Um nível '0' no TRISB configura o pino como saída.**

**Um nível '1' no TRISB configura o pino como entrada.**

Para acesso às funções da porta B, são utilizados os seguintes registradores:

- PORTB** (0x06 e 0x106) - para leitura ou escrita nos pinos da porta B;
- TRISB** (0x86 e 0x186) - para configuração dos pinos da porta B como entrada ou saída;
- OPTION\_REG** (0x81 e 0x181) - em que encontramos o sinal RBPU para o controle dos Pull-ups internos.

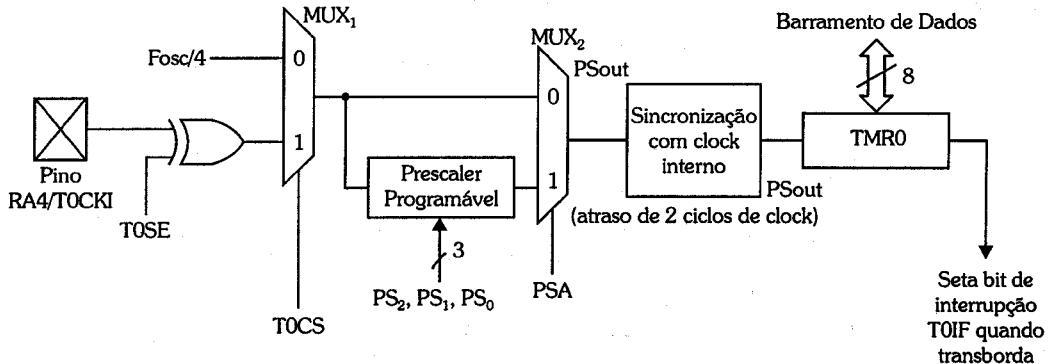
#### **Atenção**

Durante o funcionamento normal do CHIP, não devemos ultrapassar o valor de  $V_{DD}$  para qualquer dos pinos da portas do PIC, sob pena de danificá-lo irremediavelmente.

## **5.6 - Contador/Temporizador Timer 0**

A seguir, analisaremos o módulo contador/temporizador chamado de timer 0, que nada mais é do que um contador binário de 8 bits que pode ser utilizado para duas funções básicas:

- Contagem de eventos externos (quando a entrada de clock é feita por meio do pino RA4/T0CKI);
- Temporização (contagem de tempo) quando então a entrada de clock é proveniente do clock interno (o clock do sistema dividido por 4).



**Figura 5.26**

No diagrama da figura 5.26 podemos observar a estrutura interna do módulo TMR0. Primeiramente devemos prestar atenção ao MUX1, o multiplexador encarregado de selecionar a fonte de clock para o timer 0. O bit encarregado de fazer essa seleção é o T0CS (Timer 0 Clock Source - seleção de clock do timer 0), encontrado no registrador OPTION\_REG.

Pela figura anterior, podemos observar que se T0CS='1', a fonte de clock será o pino RA4/T0CKI, quando dizemos então que o timer 0 está operando como contador de eventos; caso contrário, se T0CS='0', então o clock do timer 0 virá do clock interno de sistema (clock principal/4) e dizemos que o timer 0 está operando como um contador de tempo ou temporizador.

A seguir, o sinal de clock irá passar pelo MUX2, multiplexador encarregado de selecionar entre duas opções o sinal de clock que será aplicado ao contador.

Na primeira opção, com PSA='1', o sinal de clock irá seguir diretamente do MUX1 pelo MUX2 para o timer 0.

No outro caso, com PSA='0', o sinal proveniente do MUX1 irá passar primeiramente por um prescaler e em seguida será aplicado ao contador.

O prescaler nada mais é do que um contador/divisor programável que é utilizado para reduzir a freqüência do sinal de clock aplicado ao timer 0 por um fator conhecido.

Este fator é selecionado pelos bits PS2, PS1 e PS0 do registrador OPTION\_REG, conforme a tabela seguinte:

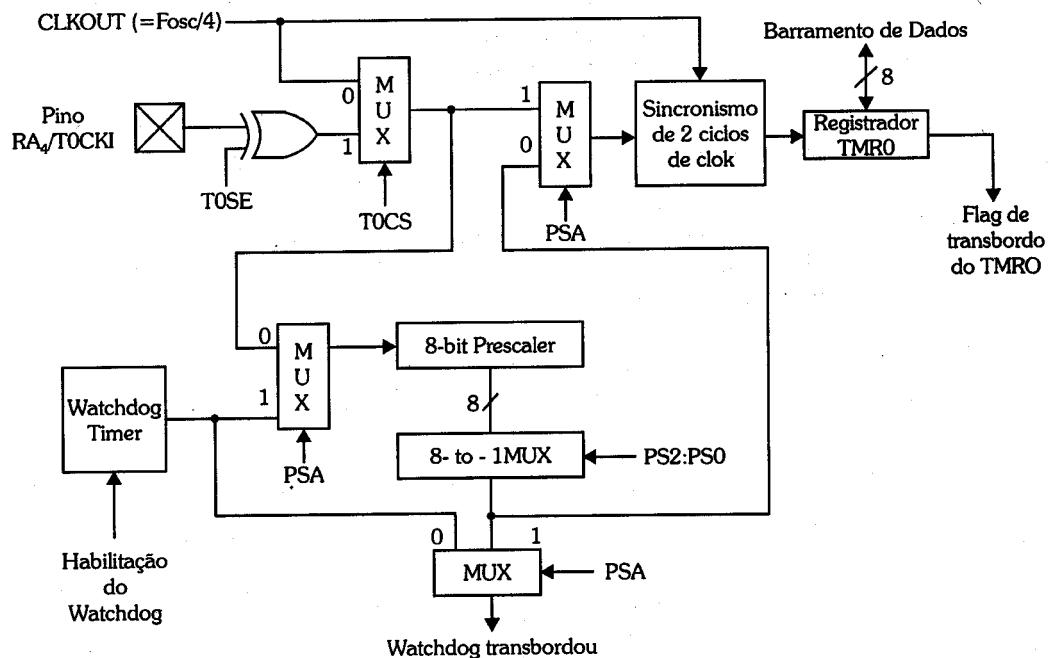
<b>PS2</b>	<b>PS1</b>	<b>PS0</b>	<b>Timer0 (PSA=0)</b>	<b>Watchdog (PSA=1)</b>
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

O valor da contagem atual do timer 0 é acessado pelo registrador TMR0. Esse registrador de 8 bits está localizado no endereço 0x01 do banco de memória 0 e também no endereço 0x101 do banco de memória 2 e pode ser lido ou escrito pelo programa do usuário. É importante observar que no caso de escrita no registrador TMR0, a contagem atual do prescaler será apagada, ou seja, a cada escrita no registrador TMR0, o prescaler volta a zero.

A cada transbordo do TMR0 (255 para 0), será ativado o sinal TOIF, sinaizando o evento de interrupção. Este sinal permanecerá ativado indefinidamente, até que o programa do usuário apague-o.

Finalmente, devemos observar que o valor atual de contagem presente no prescaler não pode ser acessado ou alterado diretamente. Além disso, o prescaler é sincronizado com os clocks internos, e por isso podem ocorrer atrasos de até dois ciclos de clock até o incremento do contador TMR0.

Na figura 5.27, podemos observar o diagrama completo do conjunto formado pelo timer 0 e pelo watchdog:



**Figura 5.27**

Observe que o prescaler estará sempre associado ao timer 0 (PSA='0') ou ao watchdog (PSA='1').

### **Observação**

Não devemos esquecer que o próprio contador do timer 0 trabalha como um divisor por 256, pois o padrão do TMR0 é iniciar a contagem a partir do 0 até 255, transbordando no próximo clock (255->0), ou seja, a cada 256 clocks é setado o flag TOIF.

### **5.6.1 - Observações sobre o Timer 0**

Algumas considerações a respeito do uso do timer 0:

- Para calcular a freqüência com que vão ocorrer as interrupções (transbordos) do timer 0, utilizamos a seguinte fórmula:

$$F_{INT} = \text{CLOCK}/\text{PRESCALER}/(256 - \text{TMR0}).$$

Em que:

*CLOCK* é o valor da freqüência de clock utilizada: se o clock for interno, teremos que o *CLOCK* será igual a  $F_{OSC}/4$ ; se o clock for externo, o valor de *CLOCK* será aquele aplicado à entrada T0CKI;

*PRESCALER* é o fator de divisão programado no prescaler. Lembre-se que se o prescaler estiver programado para ser utilizado com o watchdog, o fator de divisão que deve ser utilizado é igual a '1'.

*TMR0* é o valor inicial do registrador TMR0. Normalmente este valor é zero, mas podemos alterá-lo por software;

- Uma escrita no registrador TMR0 faz com que a contagem presente no prescaler seja ressetada. Observe que apenas a contagem atual do prescaler é apagada. O fator de divisão programado no registrador OPTION\_REG permanecerá inalterado.

Os controles do módulo Timer 0 são acessados por três registradores:

- OPTION\_REG (0x81 e 0x181) - para os sinais T0CS, T0SE, PSA, PS2, PS1 e PS0;
- TMRO (0x01 e 0x101) - para acesso à contagem do timer 0;
- INTCON (0x0B, 0x8B, 0x10B e 0x18B) - para acesso ao flag T0IF.

## 5.6.2 - Exemplos de Uso do Timer 0

### 5.6.2.1 - Clock Interno

BSF	STATUS, RP0	; MUDA PARA BANCO DE MEMÓRIA 1
CLRF	OPTION_REG	; APAGA TODOS OS BITS DO OPTION_REG
BSF	OPTION_REG, RBPU	; DESLIGA RESISTORES DE PULL-UP INTERNOS
BSF	OPTION_REG, 0	; SETA BIT PS0 NO OPTION_REG
BSF	OPTION_REG, 1	; SETA BIT PS1 NO OPTION_REG
BSF	OPTION_REG, 2	; SETA BIT PS2 NO OPTION_REG
		; CONFIGUROU-SE O PRESCALER PARA DIVIDIR
		; O SINAL DE ENTRADA POR 256
BCF	STATUS, RP0	; RETORNA PARA BANCO DE MEMÓRIA 0
PCF	INTCON, TOIF	; APAGA FLAG DE INTERRUPÇÃO DO TIMER 0
BSF	INTCON, TOIE	; HABILITA INTERRUPÇÃO DO TIMER 0
BSF	INTCON, GIE	; HABILITA INTERRUPÇÕES NO PIC

Resultado final: o timer 0 ficou configurado como temporizador, e o seu clock é igual a  $F_{OSC}/4/256$ .

### 5.6.2.2 - Clock Externo

BSF	STATUS, RP0	; MUDA PARA BANCO DE MEMÓRIA 1
MOVlw	B'10100101'	;
MOVWF	OPTION_REG	; CONFIGURA O TIMER 0 PARA CLOCK EXTERNO ; SENSÍVEL A BORDA DE SUBIDA E COM PRESCALER ; DIVIDINDO POR 64
BCF	STATUS, RP0	; RETORNA PARA O BANCO 0
BCF	INTCON, TOIF	; APAGA O FLAG DE INTERRUPÇÃO DO TIMER 0
BSF	INTCON, TOIE	; HABILITA INTERRUPÇÃO DO TIMER 0
BSF	INTCON, GIE	; HABILITA INTERRUPÇÕES NO PIC

Resultado final: o timer 0 ficou configurado como contador, e o seu clock é igual a  $F_{TOCKI} / 64$ .

### 5.6.2.3 - Clock Externo Sem Prescaler

BSF	STATUS, RP0	; MUDA PARA BANCO DE MEMÓRIA 1
MOVlw	B'10101000'	;
MOVWF	OPTION_REG	; CONFIGURA O TIMER 0 PARA CLOCK EXTERNO ; SENSÍVEL A BORDA DE SUBIDA E COM ; PRESCALER ; LIGADO AO WATCHDOG
BCF	STATUS, RP0	; RETORNA PARA O BANCO 0
BCF	INTCON, TOIF	; APAGA O FLAG DE INTERRUPÇÃO DO TIMER 0
BSF	INTCON, TOIE	; HABILITA INTERRUPÇÃO DO TIMER 0
BSF	INTCON, GIE	; HABILITA INTERRUPÇÕES NO PIC

Resultado final: o timer 0 ficou configurado como contador, e o seu clock é igual a  $F_{TOCKI} / 1$ .

### 5.6.2.4 - Alterando o Fator de Divisão do Timer 0

DEFINE	VALOR 0x01	; O VALOR DE INICIALIZAÇÃO DO TIMER 0 É ; IGUAL A 0x01
ORG	0x0000	; DIRETIVA PARA INICIAR A MONTAGEM DO ; PROGRAMA NO ENDEREÇO 0x0000
GOTO	INICIO	; PULA PARA O PONTO CHAMADO INICIO
ORG	0x0004	; MONTAGEM A PARTIR DE 0x0004
BTFS	INTCON, TOIF	; TESTA O FLAG DE INTERRUPÇÃO DO TIMER 0, ; PULA SE '1'
GOTO	FIM_INT	; SE TOIF = '0' VAI PARA O PONTO FIM_INT
MOVlw	VALOR	; COPIA O VALOR PARA O REGISTRADOR W
MOVWF	TMRO	; COPIA O VALOR (EM W) PARA O REGISTRADOR ; TMRO
...		; AQUI VAI O RESTO DO TRATAMENTO DE ; INTERRUPÇÃO DO USUÁRIO
...		; O PONTO FIM_INT ESTÁ LOCALIZADO NA PRÓXIMA ; INSTRUÇÃO
FIM_INT:		
MOVlw	B'00100000'	
MOVWF	INTCON	; APAGA TODOS OS FLAGS DO INTCON, MENOS O ; TOIE
RETFIE		; RETORNA DA INTERRUPÇÃO
INICIO:		; O PONTO INICIO ESTÁ LOCALIZADO NA PRÓXIMA ; INSTRUÇÃO

```
MOVlw      B'10000111'      ; VALOR A SER INICIALIZADO O OPTION_REG ( 
                                ; RESISTORES DE PULL-UP DESLIGADOS, TIMER 0
                                ; CLOCK INTERNO, PRESCALER 1:256 LIGADO AO
                                ; TIMER 0
BSF        STATUS, RP0       ; MUDA PARA O BANCO 1
MOVwf      OPTION_REG        ; CONFIGURA O OPTION_REG
BCF        STATUS, RP0       ; RETORNA AO BANCO 0
MOVlw      VALOR            ; INICIALIZA O
MOVwf      TMRO             ; REGISTRADOR TMRO
MOVlw      B'10100000'        ; HABILITA O GIE
MOVwf      INTCON           ; E O TOIE
...
...
...
END.
```

## 5.7 - Contador/Temporizador Timer 1

O timer 1 é um módulo contador de 16 bits capaz de operar em três modos básicos:

- ◆ Temporizador (timer) síncrono;
- ◆ Contador síncrono;
- ◆ Contador assíncrono.

Dentre as funcionalidades do TMR1 podemos destacar:

- ◆ Prescaler interno independente capaz de dividir o sinal de entrada por 1, 2, 4 ou 8;
- ◆ Capacidade de ligar/desligar a contagem do módulo por meio de um simples bit (TMR1ON);
- ◆ Possibilidade de conexão de cristal externo para clock do módulo, possibilitando a contagem mesmo com o chip em modo SLEEP (a frequência máxima nesse caso é de 200Khz);
- ◆ Capacidade de reset gerado pelo módulo CCP.

### **Nota**

O prescaler interno do módulo timer 1 é completamente independente daquele utilizado pelo timer 0 e watchdog.

Na figura 5.28 encontramos o diagrama interno do módulo TMR1.

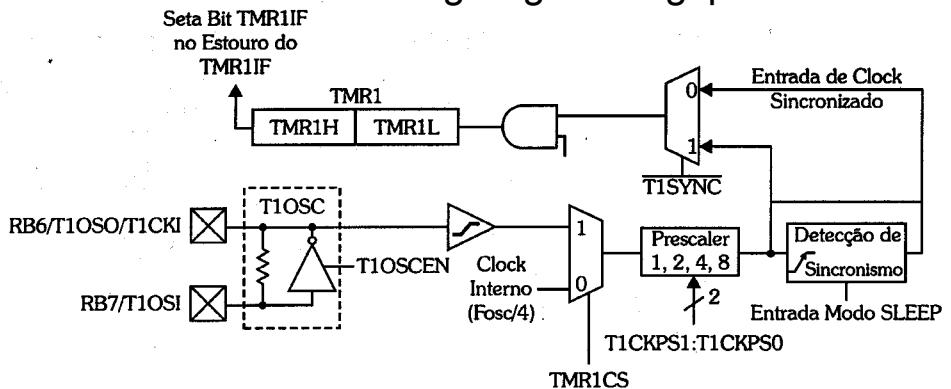


Figura 5.28

A filosofia de uso do timer 1 é a mesma do timer 0, porém destaca-se a presença de um registrador de controle destinado especificamente ao timer 1, o T1CON, além do uso de dois registradores para o armazenamento da contagem (TMR1L e TMR1H), sendo que o TMR1L acessa os 8 bits menos significativos da contagem e o TMR1H acessa os 8 bits mais significativos da contagem.

Desta forma conclui-se que um ciclo típico de contagem do timer 1 irá se iniciar em 0 (os 16 bits do contador em '0') e terminar em 65535 (o que equivale aos 16 bits do contador em '1'). O próximo pulso de clock que chegar ao contador irá gerar um estouro de contagem (pois o valor 65536 precisa de 17 bits para ser armazenado), fazendo com que o contador retorne ao valor inicial ('0000000000000000') e também setando o bit TMR1IF, sinalizando assim o estouro de contagem do módulo.

Na verdade, podemos considerar o contador do timer 1 como sendo um contador binário de 17 bits dividido da seguinte forma: os 8 primeiros bits formando o registrador TMR1L, os 8 bits seguintes formando o TMR1H e o último bit (o mais significativo) formando o TMR1IF. Observe a tabela seguinte:

Bit 17 (TMR1IF)	MSB (TMR1H)								LSB (TMR1L)								
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Lembre-se no entanto de que o bit TMR1IF somente é setado pelo timer 1 mas nunca apagado. Somente o software do usuário é que poderá providenciar o apagamento do referido bit.

Desta forma, chegamos a um fator de divisão de 1 para 65536, ou seja, 256 vezes mais que o timer 0. Este valor pode ser considerado o padrão de divisão do timer 1, mas não devemos nos esquecer que esta não é uma verdade absoluta, já que se alterarmos o valor inicial de contagem do timer 1, poderemos ter fatores de divisão desde 1 até 65536!

### 5.7.1 - Registrador T1CON (0x10)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
bit7		T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

bit0

R/W - leitura/escrita  
 R - somente leitura  
 O valor após o traço é o valor após reset POR.

Figura 5.29

- ◆ **T1CKPS1** -
- ◆ **T1CKPS0** - juntamente com o bit T1CKPS1, seleciona um dos quatro fatores de pré divisão disponíveis para o prescaler do timer 1:

T1CKPS1	T1CKPS0	Divisor
0	0	1:1
0	1	1:2
1	0	1:4
1	1	1:8

- ◆ **T1OSCEN** - Habilitação do oscilador externo do timer 1:
  - 0 - O oscilador está desabilitado, neste caso a porta inversora interna e o resistor de realimentação estão desligados, economizando energia;
  - 1 - O oscilador está habilitado.
- ◆ **T1SYNC** - Seleção de sincronização externa do timer 1:
  - 0 - Clock sincronizado com o ciclo de instruções interno;
  - 1 - clock assíncrono.
- ◆ **TMR1CS** - Seleção de clock do timer 1:
  - 0 - Clock interno (= Fosc/4);
  - 1 - Clock externo (via T1OSO/T1CKI).
- ◆ **TMR1ON** - Controle de liga/desliga da contagem do módulo timer 1:
  - 0 - Timer 1 inibido (contagem paralisada);
  - 1 - Timer1 habilitado.

#### Atenção

Os bits 7 e 6 não possuem função implementada e são lidos como '0'!

Como podemos observar, é possível escolher entre os diversos modos de operação do timer 1 simplesmente pela configuração dos bits TMR1CS e T1SYNC . A seguir, temos uma tabela demonstrando os três modos de operação do timer 1 e os respectivos bits de configuração:

TMR1CS	T1SYNC	Modo
0	0	Clock interno sincronizado
0	1	Clock interno sincronizado
1	0	Clock externo sincronizado
1	1	Clock externo não sincronizado

A diferença que existe entre os modos sincronizado e não sincronizado (assíncrono) reside no fato de que para o modo sincronizado, circuitos internos sincronizam o clock externo (após o prescaler interno do módulo) com o clock interno ( $F_{OSC}/4$ ), o que provoca um atraso entre a chegada do pulso de clock na entrada T1CKI e a sua efetiva aplicação ao contador de 16 bits.

Já no modo assíncrono, o incremento do contador é feito imediatamente, sem sincronização com o clock interno da CPU, o que, em certos casos, pode provocar efeitos colaterais que devem ser previstos. Além disso, devemos ressaltar que operando no modo assíncrono, o timer 1 pode ser utilizado inclusive quando a CPU está em modo SLEEP (desde que configurado para utilizar clock externo, já que o clock interno é desligado no modo SLEEP).

Um dos principais problemas ao operar o timer 1 em modo assíncrono consiste na leitura dos registradores TMR1L e TMR1H. Verifique a seguinte situação:

T	MSB (TMR1H)								LSB (TMR1L)							
	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
Leitura TMR1L	2	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Leitura TMR1H	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

Este exemplo demonstra que em T=2 desejava-se fazer a leitura do contador do timer 1. Para isso iniciou-se armazenando o valor de TMR1L (que naquele momento era igual a B'11111111', ou 0xFF hexadecimal). No instante T=3 continuou-se a operação, agora com a leitura de TMR1H, que neste instante é igual a B'00000001', ou '1' hexadecimal.

Se observarmos o valor final que obtivemos (0x01FF hexa) poderemos verificar que é diferente do valor que realmente pretendíamos ler, já que no instante T=2 o valor da contagem era igual a 0x00FF hexa!

Para evitar problemas como o demonstrado anteriormente, podemos nos valer de alguns artifícios:

- ◆ Parar a contagem do TMR1 (por meio do bit TMR1ON no T1CON): esta é a solução mais óbvia, porém nem todas as aplicações permitem a parada do contador;
- ◆ Utilizar uma rotina de software para verificar a possibilidade de um transbordo de TMR1L para TMR1H ter ocorrido durante a leitura deles:

```
BCF      INTCON, GIE      ; DESLIGA AS INTERRUPÇÕES
MOVF    TMR1H, W          ; LÊ TMR1H E COLOCA EM W
MOVWF   TMPH              ; ARMAZENA W NO REGISTRADOR TMPL
MOVF    TMR1L, W          ; LÊ TMR1L E COLOCA EM W
MOVWF   TMPL              ; ARMAZENA W NO REGISTRADOR TMPL
MOVEF   TMR1H, W          ; LÊ NOVAMENTE TMR1H
SUBWF   TMPH, W          ; SUBTRAI A PRIMEIRA LEITURA DA SEGUNDA
BTFSCL  STATUS, Z        ; SE IGUAL CONTINUA
GOTO    CONTINUA          ; SE O PROGRAMA CHEGAR A ESSE PONTO, É PORQUE O TMR1H FOI
                           ; INCREMENTADO DURANTE A LEITURA DOS REGISTRADORES
                           ; SE OUTRA LEITURA FOR REALIZADA AGORA, OS VALORES ESTARÃO OK
MOVEF   TMR1H, W          ; LÊ TMR1H E COLOCA EM W
MOVWF   TMPH              ; ARMAZENA W NO REGISTRADOR TMPL
MOVF    TMR1L, W          ; LÊ TMR1L E COLOCA EM W
MOVWF   TMPL              ; ARMAZENA W NO REGISTRADOR TMPL
BSF     INTCON, GIE      ; HABILITA INTERRUPÇÕES
CONTINUA:                   ; O PROGRAMA CONTINUA DAQUI
```

### 5.7.2 - Oscilador do Timer 1

Como já foi dito, o módulo timer 1 conta também com um oscilador interno capaz de utilizar cristais externos de pequeno valor (até 200 KHz).

Para utilizar o oscilador externo, devemos simplesmente setar o bit T1OSCEN do registrador T1CON.

A principal aplicação para utilizarmos o oscilador interno do timer 1 é em relógios de tempo real, e outras aplicações de medição de tempo, em que o tempo deve ser medido mesmo com o chip em modo SLEEP.

Em seguida temos uma tabela com os valores dos capacitores adequados às freqüências de cristais mais utilizadas:

Freqüência	C1	C2
32 Khz	33pF	33pF
100 Khz	15 pF	15pF
200 Khz	15pF	15pF

### 5.7.3 - Observações sobre o Timer 1

A seguir, temos algumas considerações sobre as características e funcionamento do módulo timer 1:

1. Quando se utilizar o módulo CCP, deve-se ter o cuidado de observar que em alguns modos de funcionamento ele pode gerar um sinal de reset para o timer 1. Isto deve ser considerado no caso de utilizar o timer 1 para outras funções além do CCP;
2. O timer 1 pode operar em modo SLEEP, mas somente em modo assíncrono. Isto porque no modo síncrono, o circuito de sincronização é desligado durante o SLEEP, impossibilitando então o incremento do timer;
3. Quando operando em modo assíncrono, o contador do timer 1 não se encontra sincronizado com o ciclo interno de instruções. Por isso, deve-se ter cuidado nos procedimentos de leitura dele, de forma a evitar erros na amostragem dos valores dos registradores TMR1L e TMR1H;
4. Podemos calcular a freqüência das interrupções do timer 1 com a seguinte fórmula:

$$F_{INT} = CLOCK/PRESCALER/(65536 - TMR1).$$

Em que:

**CLOCK** é o valor da freqüência de clock utilizada: se o clock for interno, teremos que o **CLOCK** será igual a  $F_{OSC}/4$ ; se o clock for externo, o valor de **CLOCK** será aquele aplicado à entrada T1CKI;

**PRESCALER** é o fator de divisão programado no prescaler;

**TMR1** é o valor inicial de 16 bits do par de registradores TMR1H e TMR1L. Normalmente este valor é zero, mas podemos alterá-lo por software.

Registradores relacionados ao módulo timer 1:

- ◆ **INTCON** (0x0B, 0x8B, 0x10B e 0x18B) - bits GIE e PEIE;
- ◆ **PIR** (0x0C) - bit TMR1IF, flag de sinalização do estouro de contagem do timer 1;
- ◆ **PIE** (0x8C) - bit TMR1IE para habilitação da interrupção;
- ◆ **TMR1L** (0x0E) - 8 bits menos significativos da contagem;
- ◆ **TMR1H** (0x0F) - 8 bits mais significativos da contagem;
- ◆ **T1CON** (0x10) - controles do timer 1.

## 5.7.4 - Exemplos de Programação do Timer 1

### 5.7.4.1 - Clock Interno e Prescaler 1:1

```
BCF      INTCON, GIE      ; DESLIGA INTERRUPÇÕES DO PIC
CLRF    T1CON             ; CONFIGURA O TIMER 1 PARA CLOCK INTERNO
                           ; PRESCALER 1:1 E CONTAGEM PARALIZADA
```

CLRF	TMR1H	; APAGA CONTAGEM
CLRF	TMR1L	; DO TIMER 1
BCF	PIR1, TMR1IF	; APAGA FLAG DE INTERRUPÇÃO DO TIMER 1
BSF	STATUS, RP0	; MUDA PARA O BANCO 1
BSF	PIE1, TMR1IE	; HABILITA INTERRUPÇÃO DO TIMER 1
BCF	STATUS, RP0	; RETORNA PARA BANCO 0
BSF	INTCON, GIE	; HABILITA INTERRUPÇÕES NO PIC
BSF	T1CON, TMR1ON	; LIGA O TIMER 1

#### 5.7.4.2 - Clock Externo em Modo Síncrono

BCF	INTCON, GIE	; DESLIGA INTERRUPÇÕES DO PIC
MOVLW	B'00010010'	; CONFIGURA O TIMER 1 PARA CLOCK EXTERNO, ; PRESCALER 1:2, MODO SÍNCRONO E CONTAGEM
MOVWF	T1CON	; PARALIZADA
CLRF	TMR1H	; APAGA CONTAGEM
CLRF	TMR1L	; DO TIMER 1
BCF	PIR1, TMR1IF	; APAGA FLAG DE INTERRUPÇÃO DO TIMER 1
BSF	STATUS, RP0	; MUDA PARA O BANCO 1
BSF	PIE1, TMR1IE	; HABILITA INTERRUPÇÃO DO TIMER 1
BCF	STATUS, RP0	; RETORNA PARA BANCO 0
BSF	INTCON, GIE	; HABILITA INTERRUPÇÕES NO PIC
BSF	T1CON, TMR1ON	; LIGA O TIMER 1

#### 5.7.4.3 - Clock Externo em Modo Assíncrono

BCF	INTCON, GIE	; DESLIGA INTERRUPÇÕES DO PIC
MOVLW	B'00110110'	; CONFIGURA O TIMER 1 PARA CLOCK EXTERNO, ; PRESCALER 1:8, MODO ASSÍNCRONO E
MOVWF	T1CON	; CONTAGEM PARALIZADA
CLRF	TMR1H	; APAGA CONTAGEM
CLRF	TMR1L	; DO TIMER 1
BCF	PIR1, TMR1IF	; APAGA FLAG DE INTERRUPÇÃO DO TIMER 1
BSF	STATUS, RP0	; MUDA PARA O BANCO 1
BSF	PIE1, TMR1IE	; HABILITA INTERRUPÇÃO DO TIMER 1
BCF	STATUS, RP0	; RETORNA PARA BANCO 0
BSF	INTCON, GIE	; HABILITA INTERRUPÇÕES NO PIC
BSF	T1CON, TMR1ON	; LIGA O TIMER 1

#### 5.7.4.4 - Clock Externo em Modo Assíncrono e Oscilador Externo Ligado

BCF	INTCON, GIE	; DESLIGA INTERRUPÇÕES DO PIC
MOVLW	B'00101110'	; CONFIGURA O TIMER 1 PARA CLOCK EXTERNO, ; PRESCALER 1:4, MODO ASSÍNCRONO, ; OSCILADOR INTERNO HABILITADO E
MOVWF	T1CON	; CONTAGEM PARALIZADA
CLRF	TMR1H	; APAGA CONTAGEM
CLRF	TMR1L	; DO TIMER 1
BCF	PIR1, TMR1IF	; APAGA FLAG DE INTERRUPÇÃO DO TIMER 1
BSF	STATUS, RP0	; MUDA PARA O BANCO 1
BSF	PIE1, TMR1IE	; HABILITA INTERRUPÇÃO DO TIMER 1
BCF	STATUS, RP0	; RETORNA PARA BANCO 0
BSF	INTCON, GIE	; HABILITA INTERRUPÇÕES NO PIC
BSF	T1CON, TMR1ON	; LIGA O TIMER 1

### 5.7.4.5 - Alterando o Fator de Divisão do Timer 1

Podemos alterar o fator de divisão do timer 1 pela simples inicialização de um valor nos registradores TMR1L e TMR1H, da mesma forma como foi visto no item 0 no estudo do timer 0:

```

ORG 0x0000          ; DIRETIVA PARA INICIAR A MONTAGEM DO PROGRAMA NO
                     ; ENDEREÇO 0x0000
GOTO    INICIO       ; PULA PARA O PONTO CHAMADO INICIO
ORG 0x0004          ; MONTAGEM A PARTIR DE 0x0004
BTFS    PIR1, TMR1IF ; TESTA O FLAG DE INTERRUPÇÃO DO TIMER 1, PULA SE
                     ; '1'
GOTO    FIM_INT      ; SE TOIF = '0' VAI PARA O PONTO FIM_INT
MOVlw   VAL_L        ; COPIA O VALOR PARA O REGISTRADOR W
MOVwf   TMR1L        ; COPIA O VALOR (EM W) PARA O REGISTRADOR TMR0
MOVlw   VAL_H        ;
MOVwf   TMR1H        ;
...
FIM_INT:           ; AQUI VAI O RESTO DO TRATAMENTO DE INTERRUPÇÃO
                     ; DO USUÁRIO
                     ; O PONTO FIM_INT ESTÁ LOCALIZADO NA PRÓXIMA
                     ; INSTRUÇÃO
MOVlw   B'01000000'  ;
MOVwf   INTCON        ; APAGA TODOS OS FLAGS DO INTCON, MENOS O PEIE
CLRF    PIR1          ; APAGA OS FLAGS DE INTERRUPÇÃO PERIFÉRICAS
RETFIE
INICIO:            ; RETORNA DA INTERRUPÇÃO
                     ; O PONTO INICIO ESTÁ LOCALIZADO NA PRÓXIMA
                     ; INSTRUÇÃO
MOVlw   B'11000000'  ; HABILITA O GIE
MOVwf   INTCON        ; E O TOIE
...
                     ; AQUI SEGUE O PROGRAMA PRINCIPAL
;
;
;
END

```

Outra forma de obter o mesmo resultado é utilizar o módulo CCP no modo de comparação. Neste modo, o par de registradores CCPR1L e CCPR1H atua como registradores de período para o timer 1, de forma idêntica aos registrador PR2 em relação ao timer 2.

Maiores detalhes sobre este assunto podem ser vistos no estudo do módulo CCP (página 147).

## 5.8 - Temporizador Timer 2

O timer 2 é um temporizador de 8 bits com princípio de funcionamento similar ao do timer 0 e as seguintes características:

- ◆ Pré-divisor (prescaler) programável de 1:1 a 1:16;
- ◆ Pós-divisor (postscaler) programável de 1:1, 1:4 ou 1:16;
- ◆ Capacidade de ligar ou desligar a contagem do módulo;
- ◆ Registrador de período (PR2).

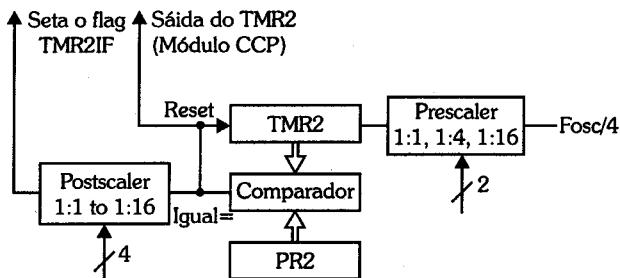
**Nota**

O prescaler interno do módulo timer 2 é completamente independente daquele utilizado pelo timer 0, timer 1 e watchdog.

A existência do módulo timer 2 serve ao propósito básico de fornecer a base de tempo para o módulo CCP1. No entanto, podemos também utilizar o timer 2 para outras aplicações, conforme veremos oportunamente.

Observe que nos dispositivos que implementam o SSP (Módulo de comunicação serial síncrona), o timer 2 é utilizado também na geração da base de tempo de transmissão/recepção do módulo. No entanto, este não é o presente caso, já que os PICs 16F62x não implementam o módulo SSP.

Na figura 5.30 podemos verificar a estrutura interna do módulo timer 2:



**Figura 5.30**

A primeira constatação que fazemos a partir da figura apresentada é que o módulo timer 2 somente pode funcionar como um temporizador, já que não possui seleção de fonte de clock (somente clock interno, Fosc/4). No entanto, podemos diminuir esta freqüência em até 16 vezes com o uso do prescaler interno do módulo.

Podemos também verificar que existem dois sinais de saída do timer 2:

- ◆ O primeiro, saindo diretamente do TMR2 reg., que nada mais é do que o sinal de transbordo do contador. Esse sinal pode servir de base de tempo para o módulo CCP;
- ◆ O segundo, saindo do Postscaler, consiste no mesmo sinal anterior, mas dividido pelo fator programado no postscaler. Esse sinal aciona o flag de interrupção do timer 2: TMR2IF.

Observe também a implementação de um registrador de período, chamado PR2, que possibilita que se altere o valor de transbordo do TMR2, ou seja: como padrão, o TMR2 vai transbordar toda vez que seu valor passar de 255 para 0, mas com o uso do PR2, armazenando um valor nele, por exemplo, 200, podemos fazer com que o TMR2 conte de 0 a 199, e ao passar a 200, o comparador detecta a coincidência e resseta o TMR2.

Desta forma, podemos facilmente programar a contagem ou divisão do timer 2 para qualquer valor entre 0 e 255.

Vejamos em seguida o registrador de controle do timer 2 TMR2CON.

### 5.8.1 - Registrador TMR2CON (12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit7							bit0

R/W - leitura/escrita  
R - somente leitura  
O valor após o traço é o valor após reset POR.

Figura 5.31

- ◆ TOUTPS3 -
- ◆ TOUTPS2 -
- ◆ TOUTPS1 -
- ◆ TOUTPS0 - Seleção do postscaler do timer 2:

TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	Divisor
0	0	0	0	1:1
0	0	0	1	1:2
0	0	1	0	1:3
0	0	1	1	1:4
0	1	0	0	1:5
0	1	0	1	1:6
0	1	1	0	1:7
0	1	1	1	1:8
1	0	0	0	1:9
1	0	0	1	1:10
1	0	1	0	1:11
1	0	1	1	1:12
1	1	0	0	1:13
1	1	0	1	1:14
1	1	1	0	1:15
1	1	1	1	1:16

- ◆ **TMR2ON** - Controle habilitação/desabilitação timer 2:  
 0 - timer 2 desativado, contagem parada;  
 1 - timer 2 habilitado, contagem correndo.
- ◆ **T2CKPS1** -
- ◆ **T2CKPS0** - Seleção de prescaler do timer 2:

T2CKPS1	T2CKPS0	Divisor
0	0	1:1
0	1	1:4
1	0	1:16
1	1	1:16

Maiores detalhes sobre o uso do timer 2 serão vistos no estudo do módulo CCP.

### 5.8.2 - Observações sobre o Timer 2

Vejamos algumas observações importantes a respeito do timer 2:

1. O timer 2 não pode operar como um contador de eventos, já que não possibilita o uso de clock externo. Sendo assim, a única fonte de clock possível para o timer 2 é o  $F_{OSC}/4$ ;
2. A implementação do postscaler permite que se utilize o timer 2 tanto na geração de sinais PWM (em conjunto com o módulo CCP), como também na função de temporizador de uso geral;
3. O prescaler e o postscaler são sempre apagados nos seguintes eventos:
  - ◆ Escrita no registrador TMR2;
  - ◆ Escrita no registrador T2CON;
  - ◆ Reset do dispositivo.
4. O módulo timer 2 não opera durante o modo SLEEP, mas retém a contagem (inclusive do prescaler e postscaler).

Registradores relacionados ao timer 2:

- ◆ **INTCON** (0x0B, 0x8B, 0x10B e 0x18B) - Bit GIE e PEIE;
- ◆ **PIR** (0x0C) - Bit TMR2IF;
- ◆ **PIE** (0x8C) - Bit TMR2IE;
- ◆ **TMR2** (0x11) - acesso à contagem do timer 2;
- ◆ **T2CON** (0x12) - Controles do timer 2;
- ◆ **PR2** (0x92) - Registrador de período do timer 2.

### 5.8.3 - Exemplo de Configuração do Timer 2

```

BCF    INTCON, GIE      ; DESLIGA INTERRUPÇÕES DO PIC
MOVLW  B'00001011'      ; CONFIGURA O TIMER 2 PARA CLOCK INTERNO,
                        ; PRESCALER 1:16, POSTSCALER 1:2
MOVWF  T2CON            ; CONTAGEM PARALIZADA
CLRF   TMR2             ; APAGA CONTAGEM DO TIMER 2

```

MOVlw	D'100'	;	O REGISTRADOR DE PERÍODO É ALTERADO
MOVWF	PR2	;	PARA O VALOR 100 DECIMAL
BCF	PIR1, TMR2IF	;	APAGA FLAG DE INTERRUPÇÃO DO TIMER 1
BSF	STATUS, RP0	;	MUDA PARA O BANCO 1
BSF	PIE1, TMR2IE	;	HABILITA INTERRUPÇÃO DO TIMER 1
BCF	STATUS, RP0	;	RETORNA PARA BANCO 0
BSF	INTCON, GIE	;	HABILITA INTERRUPÇÕES NO PIC
BSF	T2CON, TMR2ON	;	LIGA O TIMER 1

## 5.9 - Módulo CCP

O módulo CCP pode ser utilizado para três funções básicas:

- ◆ **Captura** do período de sinais PWM/pulsos - ou seja, neste modo, o módulo CCP efetua a medição do período de um sinal aplicado na entrada CCP1;
- ◆ **Comparação** - neste modo, o valor do registrador de 16 bits do módulo CCP é comparado com o valor do timer 1, e quando ocorre uma coincidência, pode-se setar o pino CCP1, ressetar o pino CCP1, ressetar o TMR1, ou gerar uma interrupção TMR1IF. Uma das utilidades do modo de comparação é a geração de pulsos de largura controlada por software;
- ◆ **Geração de sinal PWM** - neste modo, o módulo CCP1, juntamente com o timer 2, gera um sinal PWM saindo pelo pino CCP1.

Cada um dos três modos de operação descritos acima será visto em maiores detalhes mais adiante.

Por ora vejamos os registradores envolvidos na operação do módulo.

### 5.9.1 - Registrador CCP1CON (0x17)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0

bit7

bit0

R/W - leitura/escrita  
 R - somente leitura  
 O valor após o traço é o valor após reset POR.

Figura 5.32

Registrador de controle do módulo CCP:

- ◆ CCP1X -
- ◆ CCP1Y - Bits menos significativos do registrador de ciclo ativo quando em modo PWM. Quando o módulo CCP está operando em modo de captura ou comparação, esses bits não têm significado.
- ◆ CCP1M3-
- ◆ CCP1M2-
- ◆ CCP1M1-
- ◆ CCP1M0- seleção do modo de operação do módulo CCP:

<b>CCP1M3</b>	<b>CCP1M2</b>	<b>CCP1M1</b>	<b>CCP1M0</b>	<b>MODO</b>
0	0	0	0	Módulo CCP desligado, reset CCP
0	1	0	0	Captura a cada borda de descida
0	1	0	1	Captura a cada borda de subida
0	1	1	0	Captura a cada 4ª. borda de subida
0	1	1	1	Captura a cada 16ª. borda de subida
1	0	0	0	Comparação, pino CCP1 inicia em 0, em caso de coincidência CCP1 vai a '1'
1	0	0	1	Comparação, pino CCP1 inicia em 1, em caso de coincidência CCP1 vai a '0'
1	0	1	0	Comparação, em caso de coincidência gera interrupção CCP1IF
1	0	1	1	Comparação, em caso de coincidência reseta TMR1 e gera interrupção CCP1IF
1	1	X	X	Modo PWM

Os bits 7 e 6 não estão implementados e são lidos como '0'.

### 5.9.2 - Registrador CCPR1L (0x15)

O registrador CCPR1L é utilizado para armazenar os 8 bits menos significativos a serem utilizados na captura/comparação.

No modo PWM é utilizado para armazenar os 8 bits menos significativos utilizados na determinação do ciclo ativo do sinal.

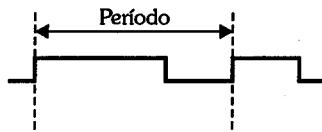
### 5.9.3 - Registrador CCPR1H (0x16)

O registrador CCPR1H é utilizado para armazenar os 8 bits mais significativos a serem utilizados nos modos de captura e comparação de sinais.

No modo de geração de sinais PWM, é utilizado como registrador escravo, no qual é copiado o conteúdo do registrador CCPR1L. Isto permite uma operação livre de ruídos e interferência que poderiam ser introduzidas no sinal caso fosse utilizado o registrador CCPR1L diretamente.

### 5.9.4 - Modo de Captura

No modo de captura, o período de um sinal aplicado à entrada CCP1 é capturado (medido), sendo o resultado armazenado no par de registradores CCPR1L e CCPR1H.



**Figura 5.33**

Temos quatro modos de captura disponíveis:

**1. Captura a cada borda de subida do sinal**

Neste modo, o período do sinal é capturado a cada borda de subida (transição de '0' para '1') do sinal aplicado ao pino CCP1(RB3).

**2. Captura a cada borda de descida do sinal**

Neste modo, o período do sinal é capturado a cada borda de descida (transição de '1' para '0') do sinal aplicado ao pino CCP1(RB3).

**3. Captura a cada 4<sup>a</sup>. borda de subida do sinal**

Neste modo, o período do sinal é capturado a cada quatro bordas de subida do sinal aplicado ao pino CCP1(RB3). Isto significa que o sinal capturado será equivalente a quatro períodos do sinal de entrada. Isto nos permite uma maior precisão na medição do sinal.

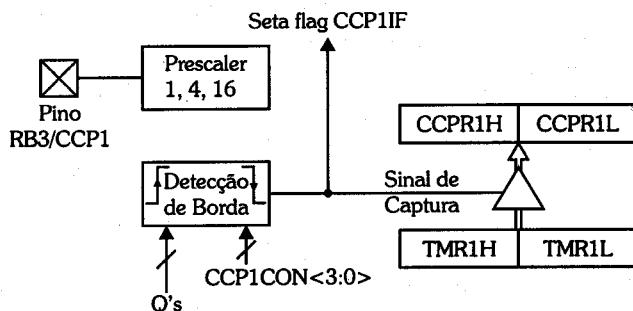
**4. Captura a cada 16<sup>a</sup>. borda de subida do sinal**

Neste modo, o período do sinal é capturado a cada dezesseis bordas de subida do sinal aplicado ao pino CCP1(RB3). Isto significa que o sinal capturado será equivalente a dezesseis períodos do sinal de entrada. Isto nos permite uma precisão ainda maior que a do modo 3 na medição do sinal.

Os modos são selecionados pelos seguintes bits do registrador CCP1CON:

CCP1M3	CCP1M2	CCP1M1	CCP1M0	Modo de Captura
0	1	0	0	1
0	1	0	1	2
0	1	1	0	3
0	1	1	1	4

Na figura 5.34 temos a estrutura interna do módulo CCP quando operando em modo de captura.



**Figura 5.34**

O princípio de funcionamento é simples.

Cada vez que o sinal de entrada aplicado ao pino CCP1 gera uma borda (de subida ou de descida, dependendo do modo de operação do módulo), é gerado um sinal de captura que copia a contagem atual do timer 1(**TMR1L** e **TMR1H**) no par de registradores **CCPR1L** e **CCPR1H**. Além disso, o sinal de captura aciona também o flag de interrupção do módulo CCP1 (CCP1IF), fazendo com que, caso as interrupções estejam habilitadas (CCP1IE='1', PEIE='1' e GIE='1'), tenhamos uma interrupção do programa em execução.

Desta forma, considerando, por exemplo, que o clock do PIC seja de 4Mhz, e o prescaler (do timer 1) esteja configurado para divisão 1:1, teremos uma freqüência de operação de 1Mhz, ou seja, cada incremento do timer 1 equivalerá a 1 $\mu$ s. Desta forma, se após uma captura, o valor armazenado no par de registradores CCPR1L:CCPR1H for igual a 1000 decimal, implicará em que o sinal capturado possui um período de 1000 $\mu$ s ou 1ms.

#### 5.9.4.1 - Observações sobre o Modo de Captura

Algumas considerações a respeito do modo de captura do CCP:

1. O pino CCP1(RB3) deve estar programado como entrada (TRISB bit 3 = '1');
2. É necessário programar o timer 1 para trabalhar como temporizador ou contador síncrono (o fabricante não garante o sucesso na captura com o timer 1 operando como contador assíncrono);
3. O sinal de captura não limpa o conteúdo dos registradores TMR1L e TMR1H, o que possibilita que o timer 1 seja utilizado para outras aplicações além da captura de sinal. No entanto, devido ao fato de não ser ressetado, devemos lembrar que o período do sinal medido será então a diferença entre duas capturas consecutivas;
4. O prescaler é programado pela escolha dos modos de captura disponíveis (bits CCP1M3 a CCP1M0 no registrador CCP1CON). No entanto, deve-se lembrar que a troca de modo de captura pode gerar um sinal de interrupção CCP1IF indesejado. Para evitar este efeito indesejado, deve-se, antes de mudar o modo de captura, proceder ao reset do módulo CCP, o que é feito comutando o modo de operação para '0000'. Além disso, é importante também que se desabilite a interrupção CCP1, o que é feito apagando o bit CCP1IE no registrador PIE1;
5. Quando em modo SLEEP, devemos ter em mente que o timer 1 irá cessar a operação (seja pela ausência do sinal de clock interno, seja pela parada do seu prescaler). Desta forma, quando um evento de captura ocorrer e se o bit CCP1IE estiver habilitado, a CPU irá acordar, mas o conteúdo do timer 1 não será capturado;

6. É possível expandir a precisão da captura para além dos 16 bits disponíveis no hardware. Para tanto, basta o aproveitamento da interrupção de transbordo do timer 1 (TMR1IF) para o incremento de uma (ou mais) variável no programa.

Registradores envolvidos na operação do módulo CCP em modo de captura:

- ◆ **CCP1CON** (0x17) - controles do módulo CCP;
- ◆ **CCPR1L** (0x15) - armazena os 8 bits LSB do período do sinal capturado;
- ◆ **CCPR1H** (0x16) - armazena os 8 bits MSB do período do sinal capturado;
- ◆ **TMR1L** (0x0E) - 8 bits inferiores do timer 1 utilizado na medição do sinal;
- ◆ **TMR1H** (0x0F) - 8 bits superiores do timer 1 utilizado na medição do sinal;
- ◆ **T1CON** (0x10) - controles do timer 1;
- ◆ **PIR1** (0x0C) - utilizado para o flag de interrupção do módulo CCP (CCP1IF);
- ◆ **PIE1** (0x8C) - utilizado para o bit de habilitação da interrupção CCP1IF;
- ◆ **INTCON** (0x0B, 0x8B, 0x10B e 0x18B) - utilizado para habilitação das interrupções periféricas (PEIE) e globais (GIE).

#### 5.9.4.2 - Exemplo de Programação do CCP no Modo Captura

```

ORG      0x0000      ; DIRETIVA PARA MONTAGEM DO PROGRAMA A
                ; PARTIR DO ENDEREÇO 0x0000
GOTO    INICIO       ; PULA PARA O INÍCIO DO PROGRAMA
ORG      0x0004      ; MONTAGEM A PARTIR DO ENDEREÇO 0x0004
BTFFS   PIR1, CCP1IF ; TESTA A INTERRUPÇÃO DO MÓDULO CCP1
RETIE   PIR1, CCP1IF ; SE CCP1IF='0' RETORNA DA INTERRUPÇÃO
BCF     PIR1, CCP1IF ; APAGA O FLAG CCP1IF
...
...
...
RETIE
INICIO:
CLRF   INTCON       ; LABEL INDICANDO O PONTO CHAMADO INICIO
CLRF   TMR1L        ; INIBE TODAS AS INTERRUPÇÕES
CLRF   TMR1H        ; APAGA O CONTEÚDO DO TMR1L
CLRF   PIR1         ; APAGA O CONTEÚDO DO TMR1H
CLRF   CCP1CON      ; INIBE TODAS AS INTERRUPÇÕES PERIFÉRICAS
MOVLW  'B'00000101' ; CONFIGURA O MÓDULO CCP1 PARA MODO
MOVWF  CCP1CON      ; DE CAPTURA A CADA BORDA DE SUBIDA
CLRF   T1CON        ; DESLIGA O TIMER 1
BSF    STATUS, RP0   ; SELECCIONA O BANCO DE MEMÓRIA 1
CLRF   PIE1          ; DESABILITA TODAS AS INTERRUPÇÕES PERIFÉRICAS
BSF    PIE1, CCP1IE  ; HABILITA INTERRUPÇÃO DO MÓDULO CCP1
BSF    TRISB, 3      ; CONFIGURA O PINO 3 DO PORT B COMO ENTRADA
BCF    STATUS, RP0   ; RETORNA PARA O BANCO DE MEMÓRIA 0
BSF    T1CON, TMR1ON ; INICIA CONTAGEM DO TIMER 1
BSF    INTCON, PEIE  ; HABILITA INTERRUPÇÕES PERIFÉRICAS
BSF    INTCON, GIE   ; HABILITA INTERRUPÇÕES GLOBAIS
...
...
...
END

```

## 5.9.5 - Modo de Comparação

No modo de comparação, a contagem do timer 1 é constantemente comparada com o valor armazenado no par de registradores CCPR1L e CCPR1H. No caso de coincidência, um dos seguintes eventos é disparado:

- O pino CCP1 é ressetado e é gerado um sinal de interrupção CCP1IF

Neste modo, o pino CCP1(RB3) é inicializado em nível lógico '0' e após uma comparação verdadeira é colocado em nível '1';

- O pino CCP1 é setado e é gerado um sinal de interrupção CCP1IF

Neste modo, o pino CCP1(RB3) é inicializado em nível lógico '1' e após uma comparação verdadeira é colocado em nível '0';

- O pino CCP1 não é alterado e é gerado um sinal de interrupção CCP1IF

Neste modo, uma coincidência irá setar o flag CCP1IF em '1', podendo ocorrer uma interrupção na CPU se o bit CCP1IE estiver habilitado. O estado do pino CCP1 não é alterado;

- O pino CCP1 não é alterado, o timer 1 é ressetado e é gerado um sinal de interrupção CCP1IF

Neste modo, uma coincidência irá setar o flag de interrupção (tal qual no modo 3) e o timer 1 será ressetado (**TMR1L** e **TMR1H** = 0). Neste modo o estado do pino CCP1 também não é alterado. Observe que apesar de o timer 1 ser ressetado, não é gerada uma interrupção de transbordo do timer 1 (TMR1IF).

O modo de comparação é selecionado no registrador CCP1CON:

CCP1M3	CCP1M2	CCP1M1	CCP1M0	Modo de Comparação
1	0	0	0	1
1	0	0	1	2
1	0	1	0	3
1	0	1	1	4

Na figura seguinte temos a estrutura interna do módulo CCP operando em modo de comparação:

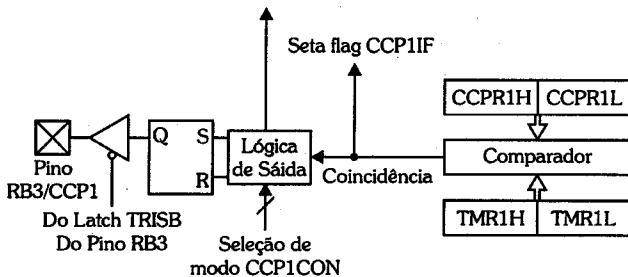


Figura 5.35

O funcionamento no modo de comparação é muito simples: o valor da contagem do timer 1 é constantemente comparado com o valor do par de registradores CCPR1L e CCPR1H. No caso de coincidência, é disparado os eventos correspondentes ao modo selecionado.

### 5.9.5.1 - Observações sobre o CCP em Modo de Comparação

Alguns aspectos a serem considerados quando da utilização do módulo CCP em modo de comparação:

1. Devemos configurar o pino RB3/CCP1 como saída (TRISB, bit 3 = '0');
2. É necessário programar o timer 1 para trabalhar como timer ou contador síncrono (o fabricante não garante o sucesso da comparação com o timer 1 operando como contador assíncrono);
3. Uma vez selecionado o modo de comparação, o dado presente no pino RB3/CCP1 será aquele relativo ao funcionamento do CCP1 (e não o dado armazenado pelo usuário no PortB bit 3);
4. Em modo SLEEP, ocorre a parada do timer 1 (seja pela ausência do clock interno ( $F_{osc}/4$ ), seja pela parada do circuito de sincronização). Assim, o estado do módulo permanecerá inalterado e da mesma forma o estado do pino RB3/CCP1;
5. Ao apagar-se o registrador CCP1CON, o estado do pino CCP1 será forçado ao estado inativo do respectivo modo, ou seja no modo 1 a saída será forçada em '0' e no modo 1 a saída será forçada em '1';
6. Lembre-se que no modo 4, o reset do timer 1 não gera uma interrupção TMR1IF.

#### Atenção

Conforme publicado pelo fabricante do dispositivo no documento de errata DS80073E, existe um BUG interno no módulo CCP1 que se manifesta nos modos comparadores 1 e 2: nas aplicações onde o modo de comparação é alterado de 1 para 2 ou de 2 para 1 a cada comparação ocorrida, a partir da terceira comparação o estado do pino CCP1 terá um comportamento inverso ao esperado. Este comportamento é errôneo e pode provocar falhas inesperadas nas aplicações.

Para solucionar este problema, pode-se adotar dois procedimentos:

1. Utilizar o modo 3 de comparação e modificar o estado do pino CCP1 por software;
2. Ressetar o módulo CCP1 a cada uma ou duas comparações realizadas.

### 5.9.5.2 - Exemplo de Configuração para Modo de Comparação

```

    ...                                ; desliga módulo CCP
CLRF    CCP1CON                      ; apaga byte superior do timer 1
CLRF    TMR1H                         ; apaga byte inferior do timer 1
CLRF    TMR1L                         ; desliga as interrupções
CLRF    INTCON                        ; muda para o banco 1
BSF     STATUS, RP0                   ; configura pino RB3 para saída
BCF     TRISB, 3                     ; desabilita interrupções periféricas
CLRF    PIE1                           ; muda para o banco 0
BCF     STATUS, RP0                   ; apaga flags de interrupção periférica
CLRF    PIR1                           ; configura o módulo CCP para o modo de
MOVLW   0x08                          ; comparação 1 (ativa pino CCP1 quando a
MOVWF   CCP1CON                      ; comparação ocorrer com sucesso)
BSF     T1CON,TMR1ON                 ; liga o timer 1

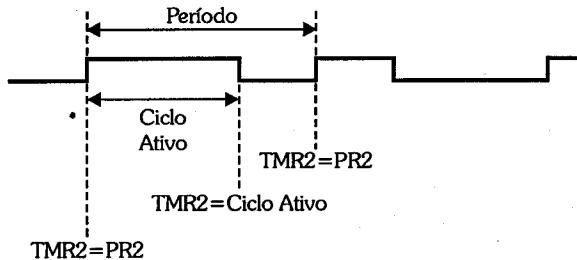
EVENTO_DE_COMPARACAO:
    BTFS   PIR1,CCP1IF                ; testa para ver se ocorreu uma comparação
    GOTO   EVENTO_DE_COMPARACAO      ; se não, retorna para a instrução acima
; se o programa chegar a este ponto é sinal de que ocorreu uma comparação bem
; sucedida
    BCF    PIR1,CCP1IF                ; apaga flag de interrupção CCP1IF

```

### 5.9.6 - Modo PWM

O último modo de operação do módulo CCP é o de geração de sinal PWM. O sinal PWM consiste num sinal de frequência fixa em que é variado o ciclo ativo da onda. É muito utilizado no controle de potência (fontes chaveadas, controles de velocidade, etc.), além de servocomandos, etc.

Apesar de ser possível (e de fato relativamente simples) gerar um sinal PWM por meio de software, a disponibilidade de hardware especializado para tanto é uma opção que não deve ser descartada. Mesmo porque o hardware específico permite uma menor ocupação do software com o controle PWM.



*Figura 5.36*

Na figura 5.37 temos um diagrama da estrutura interna envolvida na geração de sinal PWM.

Registradores de Ciclo

CCP1CON&lt;5:4&gt;

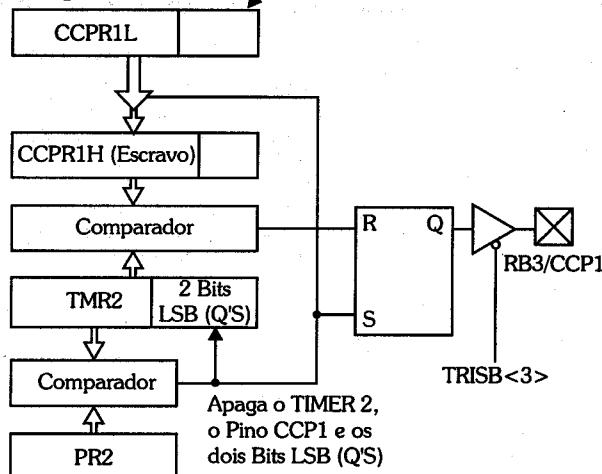


Figura 5.37

O princípio de funcionamento é o seguinte:

A base de tempo do sinal é gerada pelo timer 2. Isto significa que o período (e consequentemente a freqüência) do sinal é determinado pela programação do timer 2.

Observe na figura 5.36 que a onda (sinal PWM) inicia-se quando  $TMR2 = PR2$  (lembre-se que o PR2 é o registrador de período do timer 2). Neste momento o timer 2 volta a zero e o flip-flop de saída visto na figura 5.37 é setado, fazendo com que a saída RB3/CCP1 vá a nível lógico '1'.

Neste mesmo instante o conteúdo do registrador CCPR1L concatenado com os dois bits CCP1X e CCP1Y do CCP1CON é lido e armazenado num registrador escravo formado por CCPR1H e mais dois flip-flops internos.

Observe que o flip-flop de saída irá permanecer setado até que o valor da contagem do timer 2 seja igual ao valor armazenado em CCPR1H (que é o mesmo armazenado em CCPR1L). Neste momento, o comparador irá gerar um sinal de reset no flip-flop de saída, fazendo com que o pino RB3/CCP1 fique em nível baixo até que ocorra um novo reinício no timer 2.

Os dois bits adicionais que aparecem junto ao registrador TMR2 são obtidos internamente do prescaler do timer 2.

Observe que a largura de dados vista na figura 42 é de 10 bits. Daí temos que a resolução máxima em modo PWM é de 10 bits. No entanto esta resolução pode ser diminuída em prol do aumento da freqüência máxima do sinal PWM de saída.

O período em segundos do sinal PWM é dado pela seguinte fórmula:

$$T_{PWM} = [(PR2)+1] * 4 * T_{osc} * (\text{Prescaler TMR2})$$

O ciclo ativo é dado por:

$$T_{CICLO\ ATIVO} = (CCPR1L+CCP1X+CCP1Y)*T_{osc} *(\text{Prescaler TMR2})$$

Observe que o valor  $(CCPR1L+CCP1X+CCP1Y)$  refere-se ao valor decimal correspondente aos 10 bits resultantes da associação dos 8 bits de CCPR1L mais 1 bit de CCP1X mais 1 bit de CCP1Y, iniciando do MSB. Veja os exemplos:

CCPR1L	CCP1X	CCP1Y	Valor decimal
00000000	0	0	0
00000000	0	1	1
00000001	0	0	4
11111111	1	0	1022
11111111	1	1	1023

A resolução máxima em bits para uma determinada freqüência de PWM é dada pela fórmula:

$$\text{Resolução(BITS)} = \frac{\log(\frac{F_{osc}}{F_{PWM}})}{\log(2)}$$

Devemos ainda fazer algumas considerações sobre a operação em modo PWM:

Se  $(CCPR1L+CCP1X+CCP1Y) > PR2*4$ , teremos um ciclo ativo de 100%;

Se  $(CCPR1L+CCP1X+CCP1Y) = 0$ , teremos um ciclo ativo de 0%.

### 5.9.6.1 - Exemplo de Configuração do CCP em modo PWM

```

    ...
    CLRF      CCP1CON      ; reseta módulo CCP
    CLRF      TMR2          ; apaga a contagem do timer 2
    MOVLW    0x7F          ; configura o registrador de período (PR2)
    MOVWF    PR2           ; para contagem até 127 (divide por 128)
    MOVLW    0x1F          ; configura o CCPR1L para aproximadamente
    MOVWF    CCPR1L         ; 25% de ciclo ativo
    CLRF      INTCON        ; desliga interrupções
    BSF       STATUS, RP0   ; muda para banco 1
    BCF       TRISB, 3      ; configura pino RB3 como saída
    CLRF      PIE1           ; desabilita interrupções periféricas
    BCF       STATUS, RP0   ; volta para banco 0
    CLRF      PIR1           ; apaga flags de interrupções periféricas
    MOVLW    0x2C          ; configura o módulo CCP para modo PWM com
    MOVWF    CCP1CON        ; os bits LSB em '1' (CCP1X e CCP1Y)
    CLRF      T2CON          ; configura pre e postscaler do tmr2 para 1:1
    BSF       T2CON, TMR2ON  ; liga timer 2

```

; a partir de agora o módulo CCP está configurado e gerando um sinal PWM com  
; ciclo ativo de 25% e frequência de aproximadamente 7812 Hz

## 5.10 - Módulo Comparador Analógico

O módulo comparador analógico consiste em um conjunto de dois comparadores analógicos internos, que podem ter suas entradas associadas aos pinos do MCU para fazer comparações de tensões analógicas externas entre si ou com a fonte de referência interna do chip.

A adição do módulo comparador permite a implementação de diversos circuitos, inclusive conversores A/D, como o de rampa (utilizando a fonte de referência interna) ou conversores do tipo Delta-Sigma.

O módulo possui apenas um registrador: CMCON responsável pela configuração e controle dos comparadores internos.

### 5.10.1 - Registrador CMCON (0x1F)

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit7							bit0

R/W - leitura/escrita  
 R - somente leitura  
 O valor após o traço é o valor após reset POR

Figura 5.38

**C2OUT** - Bit indicativo do estado da saída do comparador 2:

Caso C2INV = '0':

**0** - Entrada Vin+ < Vin-

**1** - Entrada Vin+ > Vin-

Caso C2INV = '1':

**0** - Entrada Vin+ > Vin-

**1** - Entrada Vin+ < Vin-

**C1OUT** - Bit indicativo do estado da saída do comparador 1:

Caso C1INV = '0':

**0** - Entrada Vin+ < Vin-

**1** - Entrada Vin+ > Vin-

Caso C1INV = '1':

**0** - Entrada Vin+ > Vin-

**1** - Entrada Vin+ < Vin-

**C2INV** - seleção de inversão do sinal de saída do comparador 2:

**0** - Saída não invertida

**1** - Saída invertida

**C2INV** - seleção de inversão do sinal de saída do comparador 2:

**0** - Saída não invertida

**1** - Saída invertida

**CIS** - Seleção de entrada dos comparadores nos modos 1 e 2:

**CM2**

**CM1**

**CM0** - Seleção do modo de operação dos comparadores:

<b>CM2</b>	<b>CM1</b>	<b>CM0</b>	<b>Modo</b>
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

A operação do módulo comparador é muito simples: todos os parâmetros de configuração e operação dele estão localizados no registrador CMCON.

A saída dos comparadores pode ser lida pelos sinais CM1OUT e CM2OUT, ou pelos pinos do MCU (dependendo do modo de operação do módulo).

O sinal de interrupção CMIF é gerado pelo módulo sempre que ocorre uma mudança nas saídas dos comparadores.

O flag de interrupção CMIF pode ser apagado de duas formas: pela leitura do registrador CMCON ou apagando-se diretamente o flag CMIF no registrador PIR1.

## 5.10.2 - Modo 0 - Comparadores em Reset

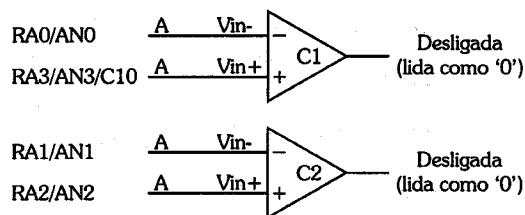


Figura 5.39

O modo 0 ( $CM2='0'$ ,  $CM1='0'$  e  $CM0='0'$ ) é o padrão dos comparadores. Isto significa que após ligarmos o PIC ou após um reset, este é o modo inicial de configuração dos comparadores.

Neste modo, ambas as saídas estão desligadas, mas os pinos RA0, RA1, RA2 e RA3 estão conectados às entradas dos comparadores.

Isto significa que no modo 0 os pinos RA0 a RA3 estão configurados como **entradas analógicas**.

A leitura dos bits C1OUT e C2OUT neste modo resulta sempre em '0'.

## 5.10.3 - Modo 1 - Dois Comparadores com Três Entradas Multiplexadas

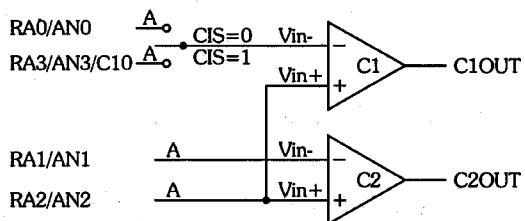


Figura 5.40

No modo 1 ( $CM2='0'$ ,  $CM1='0'$  e  $CM0='1'$ ) temos 2 entradas multiplexadas conectadas à entrada inversora de C1.

Neste modo, ambos os comparadores compartilham a mesma referência externa conectada à entrada RA2/AN2.

Podemos selecionar entre duas opções para a entrada inversora de C1:

Conectada ao pino RA0/AN0 (bit CIS em '0');

Conectada ao pino RA3/AN3/C10 (bit CIS em '1').

O sinal de saída dos comparadores (C1OUT e C2OUT) bem como o bit CIS podem ser encontrados no registrador CMCON.

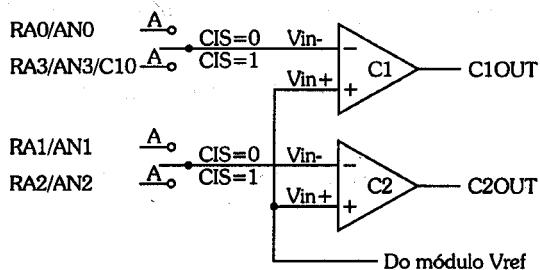
**Atenção**

Conforme publicado pelo fabricante do dispositivo no documento de errata DS80073E, existe um BUG interno no módulo comparador analógico quando configurado para o modo 1. Neste modo, se o bit CIS estiver em nível '1', selecionando a entrada RA3/AN3, o comparador 1 pode apresentar um funcionamento errático. Não há solução para este problema e portanto não devemos utilizar o modo 1 em conjunto com o bit CIS em '1'.

### **5.10.4 - Modo 2 - Dois Comparadores com Quatro Entradas Multiplexadas**

No modo 2 ( $CM2=0'$ ,  $CM1='1'$  e  $CM0='0'$ ) os comparadores são configurados com a entrada não inversora conectada ao módulo de referência de tensão interna (VREF).

As entradas inversoras de cada comparador são multiplexadas cada uma entre duas possibilidades:



**Figura 5.41**

A entrada inversora de C1 entre o pino RA0/AN0 ( $CIS='0'$ ) ou o pino RA3/AN3/C10 ( $CIS='1'$ ).

A entrada inversora de C2 entre o pino RA1/AN1 ( $CIS='0'$ ) ou o pino RA2/AN2 ( $CIS='1'$ ).

O sinal de saída dos comparadores (C1OUT e C2OUT), bem como o bit CIS, pode ser encontrado no registrador CMCON.

### **5.10.5 - Modo 3 - Dois Comparadores com uma Referência Externa Comum**

No modo 3 ( $CM2=0'$ ,  $CM1='1'$  e  $CM0='1'$ ) temos uma configuração na qual os dois comparadores possuem uma referência de tensão externa comum e conectada ao pino RA2/AN2.

Neste modo, o pino RA3/AN3/C10 não é conectado aos comparadores e pode ser utilizado para I/O digital.

O sinal de saída dos comparadores (C1OUT e C2OUT), bem como o bit CIS, pode ser encontrado no registrador CMCON.

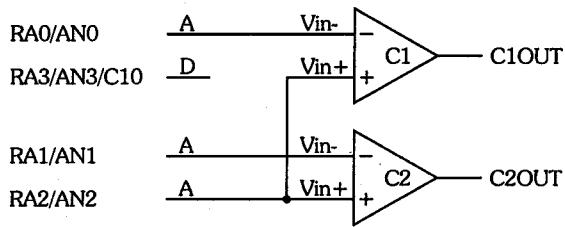


Figura 5.42

### 5.10.6 - Modo 4 - Dois Comparadores Independentes

No modo 4 (CM2='1', CM1='0' e CM0='0') os dois comparadores podem ser utilizados independentemente.

Neste modo, as quatro entradas RA0 a RA3 são configuradas como analógicas.

O sinal de saída dos comparadores (C1OUT e C2OUT), bem como o bit CIS, pode ser encontrado no registrador CMCON.

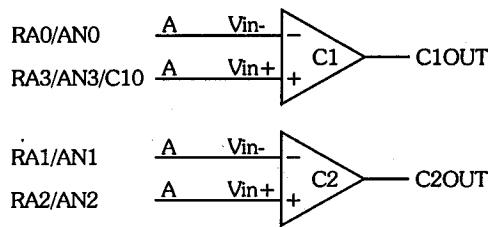


Figura 5.43

### 5.10.7 - Modo 5 - Um Comparador

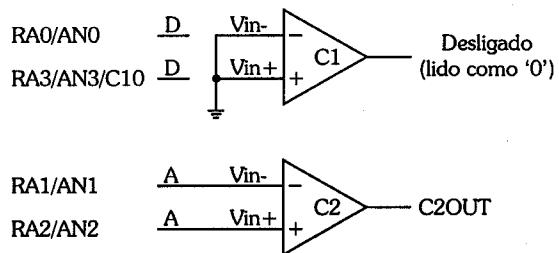


Figura 5.44

No modo 5 (CM2='1', CM1='0' e CM0='1'), apenas o comparador C2 está em funcionamento.

O comparador C1 permanece desligado. Os pinos RA0/AN0 e RA3/AN3/C10 podem ser utilizados para I/O digital.

O sinal C1OUT é lido sempre como '0'.

O sinal de saída dos comparadores (C1OUT e C2OUT), bem como o bit CIS pode ser encontrado no registrador CMCON.

### 5.10.8 - Modo 6 - Dois Comparadores com Referência Comum e Saídas Externas

No modo 6 (CM2='1', CM1='1' e CM0='0') temos dois comparadores compartilhando uma mesma referência externa comum (RA2/AN2).

As saídas C1OUT e C2OUT encontram-se disponíveis externamente nos pinos RA3/AN3/C10 e RA4/T0CKI/C20 respectivamente.

O sinal de saída dos comparadores (C1OUT e C2OUT), bem como o bit CIS pode ser encontrado no registrador CMCON.

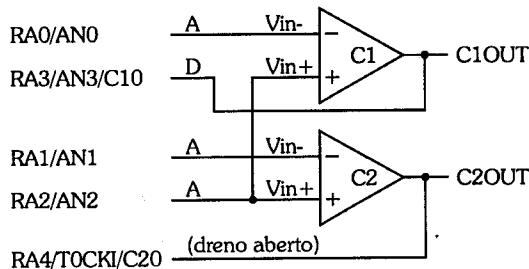


Figura 5.45

### 5.10.9 - Modo 7 - Comparadores Desligados

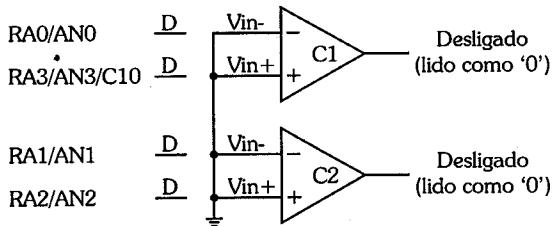


Figura 5.46

No modo 7 (CM2='1', CM1='1' e CM0='1') o módulo comparador está totalmente desligado.

Os pinos RA0 a RA3 podem ser utilizados para I/O digital.

As saídas também estão desligadas (C1OUT e C2OUT) e são lidas como '0'.

### **Observação**

Este é o modo que garante o menor consumo para o módulo comparador, sendo ideal para ser utilizado quando em modo SLEEP.

## **5.10.10 - Observações sobre o Módulo Comparador**

A seguir, temos alguns cuidados a serem tomados ao utilizar os comparadores internos:

1. Sempre desabilitar a interrupção dos comparadores (CMIE) antes de mudar o seu modo de operação;
2. Não ultrapassar os limites mínimo e máximo de alimentação nos pinos do chip (ou seja, não devemos ultrapassar  $V_{DD}$  nem  $V_{SS}$ );
3. Os comparadores podem operar durante o modo SLEEP, podendo inclusive ser utilizados para acordar a CPU, desde que a sua interrupção esteja habilitada (CMIE='1');
4. Após uma interrupção dos comparadores, a rotina de tratamento de interrupção deve garantir que o flag de interrupção CMIF esteja em '0', de forma a impedir que o programa entre em um "loop infinito";
5. Devemos ter em mente que os comparadores internos não são ideais e possuem diversos parâmetros que devem ser considerados no projeto, tais como: tempo de resposta típico da ordem de 150ns, CMRR mínima de 55db e tensão de offset de entrada tipicamente em torno de 5mV;
6. Colocando os bits CM2, CM1 e CM0 em '1', teremos o desligamento dos comparadores. Observe que este não é o modo padrão após o POR (Reset de Power-On). No entanto, este modo é o que garante o menor consumo;
7. Sempre que é selecionado um novo modo de funcionamento para os comparadores, deve-se aguardar um mínimo de 10 $\mu$ s para a estabilização da saída deles;
8. Ao utilizar os comparadores no modo 1, devemos observar a ressalva de não utilizar o bit CIS em '1', conforme a nota da página 160.

## **5.10.11 - Exemplo de Configuração do Comparador Analógico**

```
...  
MOVlw    0x04          ; inicializa o módulo comparador no  
MOVwf    CMCON         ; modo 4  
BSF      STATUS,RP0     ; muda para o banco 1  
MOVlw    0x0F          ; configura os pinos RA0 até RA3 como  
MOVwf    TRISA         ; entradas
```

BCF	STATUS, RP0	; muda para o banco 0
CALL	ESPERA10	; chama subrotina de atraso de tempo
MOVF	CΜCON, F	; lê o comparador
BCF	PIR1, CMIF	; apaga flag de interrupção do comparador
BSF	STATUS, RP0	; muda para o banco 1
BSF	PIE1, CMIE	; habilita interrupção do comparador
BCF	STATUS, RP0	; muda para o banco 0
BSF	INTCON, PEIE	; habilita interrupções periféricas
BSF	INTCON, GIE	; habilita interrupções globais
...		

## 5.11 - Referência Interna de Tensão

O módulo de referência interna de tensão ( $V_{REF}$ ) constitui-se basicamente num complemento ao módulo comparador analógico. Sua função básica é fornecer uma referência programável de tensão para uso com os comparadores analógicos já analisados.

Na figura 5.47 podemos observar o diagrama em blocos do módulo de referência de tensão. Observe que a estrutura é baseada em uma configuração de conversor D/A conhecida como "escada resistiva". Composta por um total de 16 resistores, cada "degrau" dessa "escada" fornece um nível de tensão proporcional ao ponto em que está localizado. Esses "degraus" são então multiplexados por um MUX analógico (controlado pelos bits VR0, VR1, VR2 e VR3) em cuja saída teremos a referência de tensão selecionada pelo MUX.

O valor típico de R na figura seguinte é de  $2K\Omega$ .

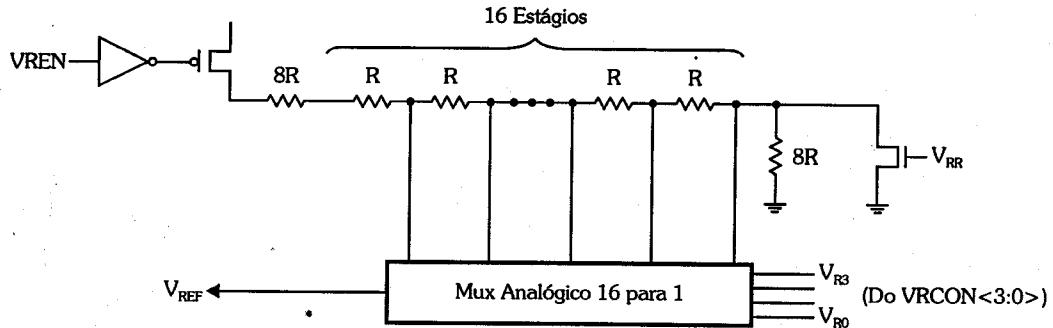


Figura 5.47

O módulo é controlado pelo registrador VRCON que veremos em seguida:

### 5.11.1 - Registrador VRCON (0x9F)

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
V <sub>REN</sub>	V <sub>ROE</sub>	V <sub>RR</sub>		V <sub>R3</sub>	V <sub>R2</sub>	V <sub>R1</sub>	V <sub>R0</sub>
bit7							bit0

R/W - leitura/escrita  
R - somente leitura  
O valor após o traço é o valor após reset POR.

Figura 5.48

**VREN** - Ativação/desativação do módulo VREF:

0 - Módulo desligado

1 - Módulo ligado

**VROE** - Conexão da saída VREF ao pino RA2:

0 - Saída VREF não conectada ao pino RA2

1 - Saída VREF conectada ao pino RA2

**VRR** - Seleção de escala de tensão0 - Grande escala (de 0,25VDD até 0,75VDD), incrementos de VDD/32. Neste caso a tensão VREF será igual a  $(VR<3:0>/24) \times VDD$ 1 - Pequena escala (de 0V até 0,75VDD), incrementos de VDD/24. Neste caso a tensão VREF será igual a  $(VDD \times 1/4) + (VR<3:0>/32) \times VDD$ **VR3****VR2****VR1****VR0** - Seleção do nível de tensão de saída do módulo. Veja a tabela em seguida para valores típicos para a saída VREF. Observe que os valores desta tabela são aproximados e são válidos para uma tensão  $V_{DD} = 5V$ .

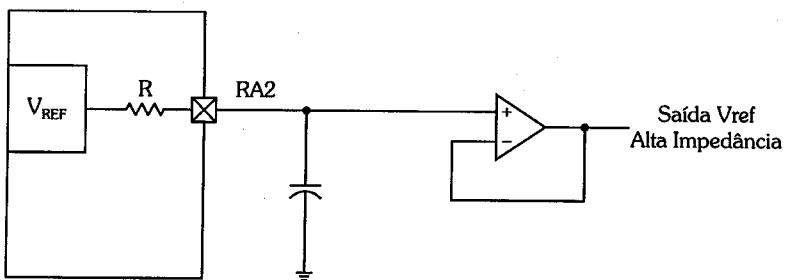
VR3:VR0	V <sub>REF</sub>	
	VRR=1	VRR=0
0000	0.00V	1.25V
0001	0.21V	1.41V
0010	0.42V	1.56V
0011	0.63V	1.72V
0100	0.83V	1.88V
0101	1.04V	2.03V
0110	1.25V	2.19V
0111	1.46V	2.34V
1000	1.67V	2.50V
1001	1.88V	2.66V
1010	2.08V	2.81V
1011	2.29V	2.97V
1100	2.50V	3.13V
1101	2.71V	3.28V
1110	2.92V	3.44V
1111	3.13V	3.59V

*Figura 5.49*

## 5.11.2 - Observações sobre a Referência Interna de Tensão

Devemos ainda fazer algumas considerações sobre a utilização do módulo VREF:

1. O modo padrão do módulo VREF é desligado VRCON='0';
2. O bit 4 do registrador VRCON não é implementado e é lido como '0';
3. Durante o modo SLEEP, o módulo VREF continua funcionando normalmente (a não ser que seja desligado, é claro);
4. O módulo VREF pode ser utilizado como um conversor D/A de baixa resolução, no entanto devemos lembrar que a saída VREF possui pequena capacidade de corrente, sendo por isso necessária a utilização de um buffer analógico para aumentar a capacidade de corrente de saída do módulo;



**Figura 5.50**

Note que o valor da resistência interna R vai depender do modo configurado para a operação do módulo.

5. O valor a ser carregado em VR3 a VR0 pode ser obtido pelas seguintes fórmulas:

$$\text{Se } VRR = '1', \text{ VR3 : VR0} = \frac{V_{REF}}{V_{DD}} \times 24$$

$$\text{Se } VRR = '0', \text{ VR3 : VR0} = \frac{V_{REF} - V_{DD}/4}{V_{DD}} \times 32$$

## 5.11.3 - Exemplo de Configuração da Referência Interna

```

MOVLW 0xA6      ; ...
BSF    STATUS, RP0 ; seleciona o banco 1
MOVWF VRCON     ; configura o Vref para 2,19 Volts
BCF    STATUS, RP0 ; seleciona o banco 0
CALL   ATRASO     ; chama subrotina de atraso (no mínimo 10µs)

```

## 5.12 - USART

A USART (Interface Serial Universal síncrona/assíncrona), também chamada de SCI (Interface de Comunicação Serial), é um dispositivo interno utilizado para fazer a comunicação serial com elementos externos ao chip, tais como: computadores, modems, terminais, memórias, conversores A/D e D/A, etc.

O coração da USART é composto de dois registradores de deslocamento, responsáveis pela conversão paralelo/serial (transmissão), chamado internamente de TSR, e serial/paralelo (recepção), chamado internamente de RSR.

A existência de dois registradores de deslocamento possibilita o funcionamento Full-Duplex, ou seja, a transmissão e recepção de dados de forma simultânea. No entanto, no modo síncrono somente podemos trabalhar no modo Half-duplex (a transmissão e a recepção são feitas alternadamente).

Encontramos também na USART um gerador interno de clock para os registradores de deslocamento. Esse gerador é chamado BRG (Baud Rate Generator - Gerador de Baud Rate) e consiste em um divisor cujo clock provém indiretamente do clock de sistema ( $F_{OSC}$ ).

Os sinais transmitidos/recebidos pela USART fluem por dois pinos externos do chip: RB1 e RB2.

O pino RB1 atua como entrada (recepção de dados) quando a USART está trabalhando em modo assíncrono, e como entrada/saída de dados quando a USART está trabalhando em modo síncrono.

Já o pino RB2 atua como saída (dados transmitidos) quando a USART está trabalhando em modo assíncrono, como saída de clock quando a USART está trabalhando em modo síncrono com clock interno e como entrada quando a USART está trabalhando em modo síncrono com clock externo.

Além da transmissão/recepção padrão de 8 bits, o módulo USART permite ainda um outro modo de transmissão/recepção de 9 bits, que permite o funcionamento da USART em duas outras modalidades:

1. Utilização do nono bit para a função de detecção de paridade, permitindo assim um controle de erros de transmissão de baixo nível;
2. Utilização do nono bit para a função de controle de endereçamento. Neste modo, o nono bit funciona como um bit de controle de endereçamento, indicando se o dado transmitido/recebido é um caractere ou um endereço de dispositivo.

A utilização do nono bit para a função de controle de endereçamento permite que diversos dispositivos compartilhem um mesmo barramento serial, cada qual com seu próprio endereço. Desta forma, é possível ao dispositivo mestre comunicar-se com vários dispositivos escravos, apenas selecionando previamente o endereço de cada um.

Observe que o modo de endereçamento somente está disponível no modo assíncrono de comunicação.

Assim, é possível configurar a USART para trabalhar em quatro modos, a saber:

- ◆ Assíncrono Full-duplex sem detecção de endereços;
- ◆ Assíncrona Full-duplex com detecção de endereços;
- ◆ Síncrono Half-duplex com clock interno (modo mestre);
- ◆ Síncrono Half-duplex com clock externo (modo escravo).

Para o controle e configuração da USART utilizamos cinco registradores específicos:

- ◆ **TXSTA** (0x98) - utilizado para controle e configuração de transmissão da USART;
- ◆ **RCSTA** (0x18) - utilizado para controle e configuração de recepção da USART;
- ◆ **SPBRG** (0x99) - utilizado para configurar o clock da USART;
- ◆ **TXREG** (0x19) - registrador de transmissão de dados (armazena o dado a ser transmitido);
- ◆ **RCREG** (0x1A) - registrador de recepção de dados (armazena o dado recebido).

Obs.: O registrador RCREG é na realidade constituído por uma memória FIFO de duas posições.

A seguir, estudaremos a estrutura interna de transmissão e recepção da USART, seus registradores e modos de operação em maiores detalhes.

### 5.12.1 - Estrutura de Transmissão da USART

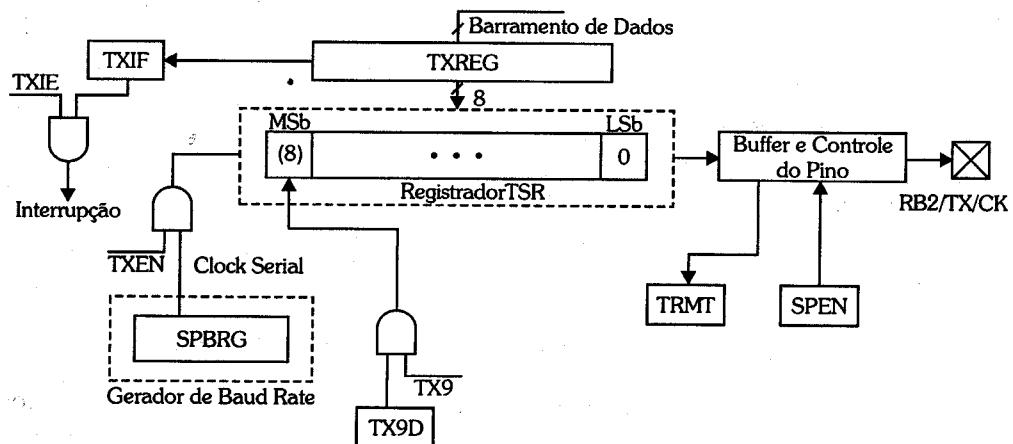


Figura 5.51

A figura 5.51 apresenta o diagrama em blocos simplificado da estrutura interna de transmissão da USART. Nele podemos destacar:

- ◆ BRG - Gerador de Baud Rate - responsável pela geração de clock para comunicação (esse módulo é compartilhado com a estrutura de recepção da USART);
- ◆ TSR - Registrador de deslocamento de transmissão - responsável pela conversão da informação paralela vinda do registrador TXREG em informação serial;
- ◆ TXREG - Registrador de dados transmitidos - armazena o dado a ser transmitido;
- ◆ Lógica de apoio - responsável pelo controle de interrupção, habilitação da USART, sinalização de estado e modo de 9 bits.

### 5.12.2 - Estrutura de Recepção da USART

Na figura 5.52 temos a estrutura interna de recepção da USART. Nela podemos verificar a interação entre os diversos componentes internos da USART.

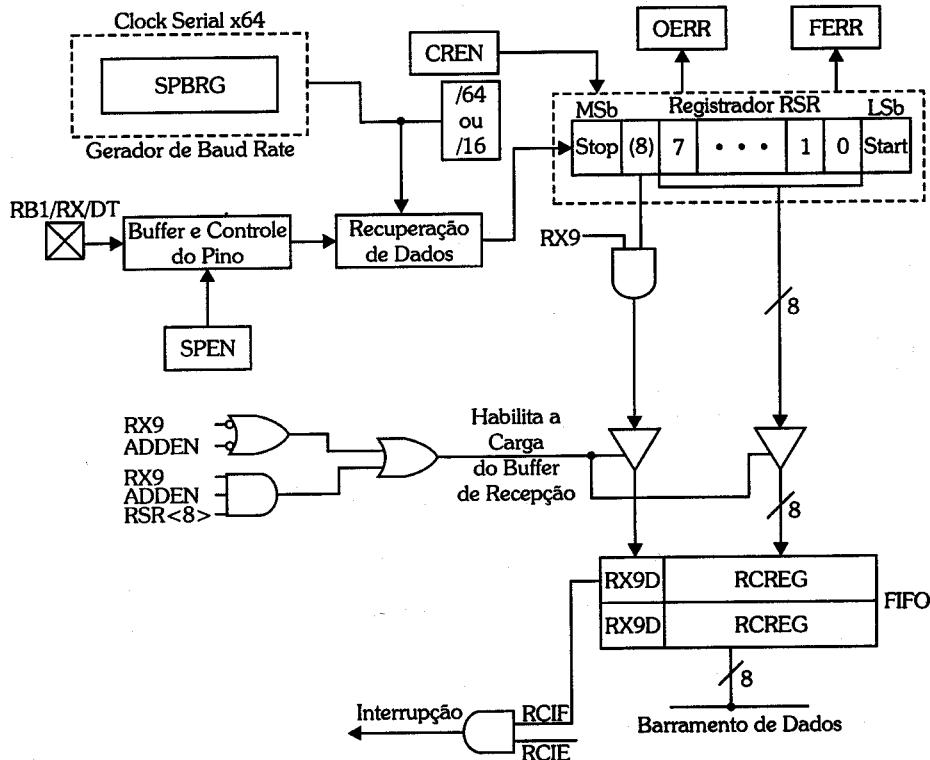


Figura 5.52

Como principais elementos, podemos destacar:

- ◆ O BRG - Gerador de Baud Rate - responsável pela geração de clock de comunicação (compartilhado com a estrutura de transmissão);
- ◆ RSR - Registrador de descolamento de recepção - responsável pela conversão da informação serial recebida em informação paralela;
- ◆ RCREG - Registrador de dados recebidos - estrutura FIFO de dois estágios destinada a armazenar a informação recebida do RSR;
- ◆ Lógica de apoio - responsável pela amostragem do sinal serial de entrada, controle de endereçamento, detecção de erros, seleção de modo 9 bits, sinalização de interrupção e habilitação da USART.

A seguir, estudaremos os registradores envolvidos na operação da USART, bem como os seus modos e funcionamento.

### 5.12.3 - Registrador TXSTA (98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0
CSRC	TX9	TXEN	SYNC		BRGH	TRMT	TX9D

bit7

bit0

R/W - leitura/escrita  
 R - somente leitura  
 O valor após o traço é o valor após reset POR.

*Figura 5.53*

**CSRC** - Seleção de fonte de clock síncrono:

- 0** - Clock externo (modo escravo)
- 1** - Clock interno (modo mestre)

#### *Observação*

No modo assíncrono este bit não é utilizado.

**TX9** - Habilitação de modo de transmissão de 9 bits:

- 0** - Seleciona modo de transmissão de 8 bits
- 1** - Seleciona modo de transmissão de 9 bits

**TXEN** - Habilitação do transmissor:

- 0** - Transmissor desabilitado
- 1** - Transmissor habilitado

**SYNC** - Seleção de modo de operação da USART:

- 0** - Seleciona modo assíncrono de transmissão
- 1** - Seleciona modo síncrono de transmissão

**BRGH** - Seleção de modo de clock de alta velocidade:

- 0** - Modo de baixa velocidade
- 1** - Modo de alta velocidade

#### *Observação*

Este bit somente tem função no modo assíncrono. No modo síncrono ele não é utilizado.

**TRMT** - Bit indicador do estado do registrador de deslocamento (transmissão):

- 0** - Registrador de transmissão (TSR) cheio
- 1** - Registrador de transmissão (TSR) vazio

**TX9D** - Nonº bit de dados (usado apenas no modo de 9 bits, pode ser utilizado para gerar informação de paridade (detecção de erros de transmissão).

#### *Observação*

O bit 3 não está implementado e é lido como '0'.

### 5.12.4 - Registrador RCSTA (18h)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit7							bit0

R/W - leitura/escrita  
R - somente leitura  
O valor após o traço é o valor após reset POR.

*Figura 5.54*

**SPEN** - Habilitação da USART:

- 0** - USART desabilitada (pinos RB1/RB2 podem ser utilizados p/I/O)
- 1** - USART habilitada (pinos RB1/RB2 utilizados p/USART)

**RX9** - Habilitação de recepção de 9 bits:

- 0** - Seleciona modo de recepção de 8 bits
- 1** - Seleciona modo de recepção de 9 bits

**SREN** - habilitação de recepção única:

- 0** - Desabilita recepção única
- 1** - Habilita recepção única

**Observação**

Após a recepção de um caractere este bit volta a '0'.

**CREN** - Habilitação de recepção contínua:

- 0** - Desabilita recepção contínua
- 1** - Habilita recepção contínua

**ADDEN** - Habilitação de detecção de endereço:

- 0** - Desabilita a detecção de endereço. O 9º bit pode ser utilizado para paridade.
- 1** - Habilita a detecção de endereço. A USART permanece aguardando a chegada de um endereço.

**Observação**

A detecção de endereços não deve ser utilizada no modo síncrono.

**FERR** - Indicador de erro de *Frame* (quadro):

- 0** - Nenhum erro de frame
- 1** - Erro de frame (Bit de STOP não detectado ou detectado fora de hora)

**OERR** - Indicador de erro de *overrun* (sobreposição):

- 0** - Nenhum erro de overrun
- 1** - Erro de overrun, três caracteres foram recebidos sem terem sido lidos pela CPU.

**RX9D** - Nono bit dos dados recebidos pela USART, pode ser utilizado para verificar a paridade, desde que o transmissor esteja configurado de acordo.

## 5.12.5 - Registrador SPBRG (0x99)

O registrador SPBRG é o responsável pelo controle do gerador de baud rate interno da USART.

Por meio do valor colocado nesse registrador, altera-se o fator de divisão do BRG, resultando em diferentes freqüências de clock para a USART.

A tabela seguinte demonstra as fórmulas de cálculo do Baud Rate para cada modo possível:  $F_{OSC}$

SYNC	Modo	$BRGH='0'$	$BRGH='1'$
0	Assíncrono	Baud Rate = $F_{OSC}/[64(X+1)]$	Baud Rate = $F_{OSC}/[16(X+1)]$
1	Síncrono	Baud Rate = $F_{OSC}/[4(X+1)]$	N/A

### Observação

Na tabela acima, X é o valor que deve ser armazenado no registrador SPBRG.

Vejamos um exemplo: digamos que desejemos utilizar uma velocidade de 19.200 Bps assíncrona num circuito com o PIC trabalhando com clock de 20 Mhz. Teremos de calcular o valor a ser programado no registrador SPBRG. Teríamos então duas alternativas:

$$\text{Com } BRGH='0' \Rightarrow 19200 = 20.000.000,00 / [64(X+1)]$$

$$X = 15,28 \text{ arredondando, } X = 15 \Rightarrow SPBRG = 15$$

Reaplicando a fórmula com  $X=15$ , teremos:

$$\text{Velocidade} = 20.000.000,00 / [64(15+1)]$$

$$\text{Velocidade} = 19.531,25 \text{ Bps}$$

O que nos dá um erro aproximado de:

$$\text{Erro}(\%) = ((19.531,25 - 19.200) / 19.200) * 100 = 1,7\%$$

$$\text{Com } BRGH='1' \Rightarrow 19200 = 20.000.000,00 / [16(X+1)]$$

$$X = 64,1 \text{ arredondando, } X = 64 \Rightarrow SPBRG = 64$$

$$\text{A velocidade real será igual a} \\ 20.000.000,00 / [16(64+1)] = 19.230,77$$

Resultando num erro de 0,16 %.

Veja então que no presente caso, é muito mais interessante utilizar o BRGH em '1', pois o erro será muito menor e, consequentemente, a velocidade muito mais próxima da desejada.

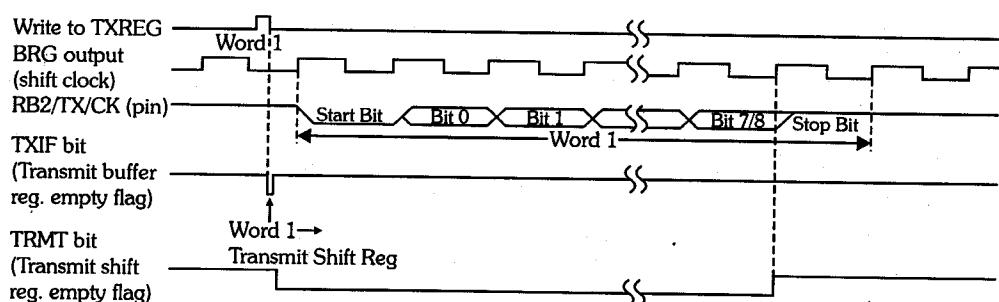
No apêndice B temos tabelas com valores previamente calculados para o registrador SPBRG em função de diversos clocks e velocidades.

### 5.12.6 - Transmissão no Modo Assíncrono

A transmissão serial assíncrona caracteriza-se pela ausência de uma linha de sincronização entre o elemento transmissor e o elemento receptor.

Como não há sincronização de clock entre os elementos, utilizam-se velocidades padronizadas que devem ser seguidas por cada elemento de comunicação. Assim, uma vez que dois elementos estejam operando com uma mesma velocidade de comunicação, utilizam-se bits de START (início) e STOP (fim) para sinalizar o início e o fim de uma transmissão.

Na figura 5.55 temos um diagrama da transmissão assíncrona da USART.



**Figura 5.55**

Para realizarmos a transmissão de dados no modo assíncrono, devemos observar os seguintes passos:

1. Proceder à inicialização do gerador de Baud Rate, selecionando o modo de alta ou baixa velocidade por meio do bit BRGH (regidor TXSTA) e escrevendo o valor correspondente à velocidade desejada no registrador SPBRG;
2. Selecionar o modo assíncrono (SYNC='0', regidor TXSTA) e habilitar a USART (SPEN='1', regidor RCSTA). Obs.: Depois de habilitada a USART, os pinos RB1 e RB2 passam a funcionar controlados pela USART.
3. Se o programador necessitar do uso da interrupção de transmissão deve habilitar os bits TXIE, PEIE e GIE;
4. Se for desejado utilizar o modo de 9 bits de transmissão, devemos setar o bit TX9 no regidor TXSTA;

5. Devemos habilitar a transmissão por meio do bit TXEN (registrator TXSTA);
6. Se foi programado o modo de 9 bits, o nono bit deve ser carregado no bit TX9D no registrador TXSTA;
7. Finalmente, carrega-se o dado a ser transmitido no registrador TXREG. A transmissão tem início imediatamente;
8. Uma vez que o dado armazenado no TXREG é copiado para o TSR, o flag de interrupção TXIF é setado. Repare que este flag não pode ser apagado por software. A única forma de se apagar o flag é armazenar um novo dado no registrador TXREG.

#### **5.12.6.1 - Observações sobre a Transmissão Assíncrona**

Algumas observações sobre o funcionamento da USART no modo de transmissão assíncrona:

- ◆ Muito cuidado deve ser tomado ao utilizar a interrupção de transmissão da USART: ela será ativada cada vez que o dado presente no registrador TXREG for passado para o TSR. Desta forma, se for desejado o uso de interrupção, a rotina de tratamento da interrupção TXIF é que deverá providenciar a carga de um novo dado a ser transmitido no registrador TXREG;
- ◆ O flag TXIF é somente para leitura. Ele somente pode ser apagado colocando um novo dado no registrador TXREG;
- ◆ O bit TRMT sinaliza que o dado presente no registrador TSR foi totalmente transmitido e agora o TSR está vazio, aguardando um novo dado a ser transmitido;
- ◆ Se o bit TXEN for apagado durante uma transmissão, ela será abortada, ressetando o módulo de transmissão;
- ◆ O pino RB2/TX/CK irá permanecer em alta impedância quando não houver transmissão em andamento;
- ◆ Na transmissão em modo de 9 bits, devemos sempre armazenar primeiramente o nono bit no registrador TXSTA e então os restantes 8 bits no registrador TXREG. Isto porque se o TSR estiver vazio, uma escrita no TXREG provoca imediatamente o início de uma transmissão.

#### **5.12.7 - Recepção no Modo Assíncrono**

Para utilizarmos o módulo de recepção serial da USART, devemos observar os seguintes passos:

1. Proceder à inicialização do gerador de Baud Rate, selecionando o modo de alta ou baixa velocidade por meio do bit BRGH (registrator TXSTA) e

escrevendo o valor correspondente à velocidade desejada no registrador SPBRG;

2. Selecionar o modo assíncrono (SYNC='0', registrador TXSTA) e habilitar a USART (SPEN='1', registrador RCSTA).

### ***Observação***

Depois de habilitada a USART os pinos RB1 e RB2 passam a funcionar controlados pela USART.

3. Se o programador necessitar do uso da interrupção de transmissão deve habilitar os bits RCIE, PEIE e GIE;
4. Se for desejado utilizar o modo de 9 bits de transmissão, devemos setar o bit RX9 no registrador RCSTA;
5. Devemos habilitar a transmissão, setando o bit CREN (registrador RCSTA);
6. Se a interrupção RCIE estiver habilitada, devemos esperar o acontecimento de uma interrupção RCIF, notificando o programa de que foi recebido um caractere. Se a interrupção RCIE estiver desabilitada, devemos constantemente verificar o flag RCIF para detectar se houve a recepção de um caractere;
7. Efetuar a leitura do registrador RCSTA para verificar se houve algum erro na recepção. Se estiver habilitada a recepção de 9 bits, podemos também imediatamente ler o nono bit disponível no bit RX9D do registrador RCSTA;
8. Fazer a leitura do caractere recebido no registrador RCREG. Este procedimento automaticamente apaga o flag RCIF;
9. Se houver algum erro de recepção, apagar o bit CREN para limpar o erro.

#### **5.12.7.1 - Observações sobre a Recepção Assíncrona**

- ◆ Se a interrupção de recepção de caractere estiver habilitada (RCIE='1'), o flag RCIF irá permanecer em '1' até que o dado seja lido no registrador RCREG;
- ◆ A estrutura FIFO, na qual encontramos o registrador RCREG, possui na verdade duas posições de 10 bits (8 para o caractere recebido, uma para o nono bit e outra o armazenamento do flag FERR). Por isso, antes de realizar uma leitura do RCREG, devemos primeiramente ler os bits FERR (para verificar se o caractere a ser lido foi recebido corretamente) e em seguida ler o nono bit armazenado em RX9D (se o modo de 9 bits estiver ativado);
- ◆ Cada nova leitura do RCREG provoca a descida do nível superior da FIFO e consequentemente a atualização dos bits FERR, RX9D e do conteúdo do RCREG;

- ◆ Enquanto existirem caracteres armazenados na estrutura FIFO, o flag RCIF permanecerá setado;
- ◆ O flag RCIF é somente para leitura, somente sendo apagado por hardware, o que ocorre com o esvaziamento da FIFO (RCREG);
- ◆ A sinalização de erro de quadro (FERR) ocorre quando existirem problemas na detecção do bit de STOP. O caractere recebido é inválido, pois pode conter erros;
- ◆ O erro de overrun (OERR) ocorre quando a FIFO está cheia (2 caracteres) e ocorre a chegada de um terceiro caractere no RSR. A sinalização do erro (OERR='1') indicará que o terceiro caractere recebido foi descartado. Os dois caracteres armazenados na FIFO podem ser lidos pela leitura sucessiva do RCSTA e RCREG. Enquanto o bit OERR permanecer setado, não ocorrem transferências do RSR para a FIFO, e os novos caracteres serão descartados;
- ◆ Para efetuar a limpeza do bit OERR, devemos desabilitar a recepção (CREN='0'), o que provoca o reset do módulo, em seguida podemos habilitá-lo novamente (CREN='1') e prosseguir com a recepção de caracteres.

### 5.12.7.2 - Recepção no modo Assíncrono com Endereçamento

O modo de endereçamento faz uso da capacidade de transmissão de 9 bits para criar uma estrutura de endereçamento mestre-escravo.

Neste modo de operação, o nono bit é utilizado para sinalizar o tipo de dado sendo transmitido/recebido:

Se RX9D='1', o dado é um endereço;

Se RX9D='0', o dado é um caractere.

Desta forma, ao configurarmos o bit ADDEN='1' no registrador RCSTA, estamos habilitando a verificação de endereços pela USART. A partir de então a interrupção RCIF somente será ativada se o nono bit (armazenado em RX9D) for igual a '1'. Caso seja igual a zero, o dado recebido é descartado.

Assim, a USART permanecerá recebendo e descartando todos os dados cujo nono bit seja igual a '0', até que ocorra a recepção de um dado cujo nono bit seja igual a '1' (ou seja, o dado é um endereço). Neste caso, o endereço será descarregado do RSR para o RCREG e será ativado o flag RCIF.

A partir de então, o programa deve tratar a interrupção e efetuar a leitura dos registradores RCSTA (para verificar se não houve erros de recepção) e do RCREG para a leitura do endereço recebido.

Esse endereço recebido é então comparado com outro definido pelo programador/usuário no software em execução. Se os endereços forem diferentes, a USART deverá permanecer com ADDEN='1' esperando por um novo endereço (veja figura 5.57). Se forem iguais, significará que os dados a partir de agora deverão ser lidos e armazenados, pois são destinados ao dispositivo (veja figura 5.58).

Para tanto, o programador deve apagar o bit ADDEN, para permitir que seja feita a recepção normal de dados pela USART.

Não devemos nos esquecer de que para cada caractere recebido devemos primeiramente verificar a existência de erros e também se ele não é um novo endereço (bit RX9D = '1').

Caso RX9D seja igual a '1', o software deve providenciar o retorno da USART para o modo de verificação de endereços (ADDEN='1').

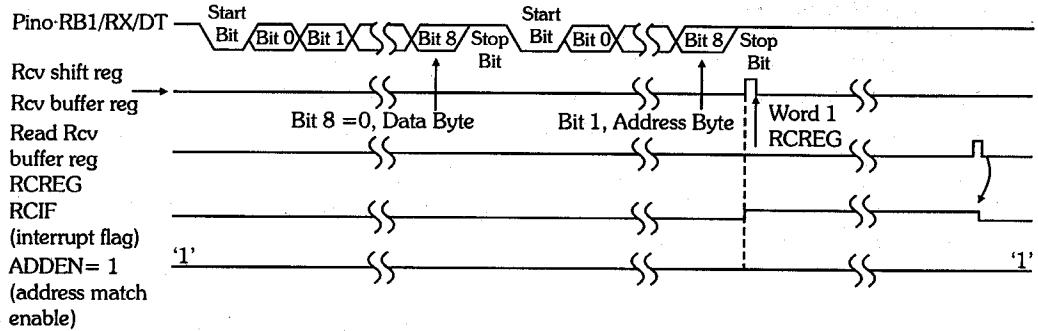


Figura 5.56 - Um dado seguido de endereço.

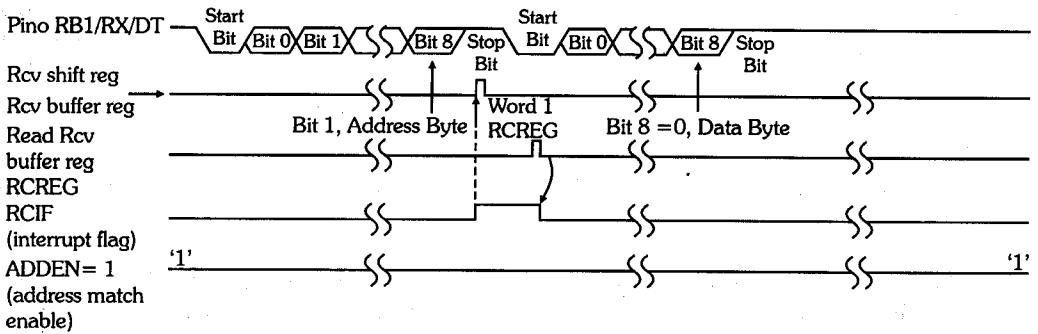


Figura 5.57 - Um endereço seguido por caractere, neste caso o endereço não era o mesmo da USART e a mesma permanece com ADDEN='1' esperando um novo endereço.

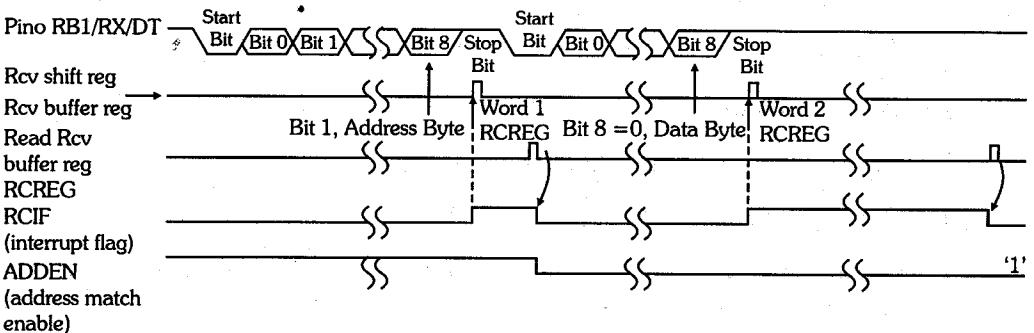


Figura 5.58 - Um endereço é recebido e após comparação interna é validado. A USART é comutada para recepção normal de caracteres.

## 5.12.8 - Transmissão Síncrona Modo Mestre

A comunicação síncrona, diferentemente da assíncrona, utiliza uma linha de clock para sincronizar o transmissor e o receptor.

Na transmissão síncrona, cada bit de informação é colocado na linha de dados ao mesmo tempo em que é fornecido um pulso na linha de clock.

Desta forma, o receptor irá amostrar a sua recepção a cada pulso de clock, em perfeita sincronia com o elemento transmissor.

A transmissão síncrona permite uma maior velocidade que a assíncrona, já que na síncrona a sincronização é feita por uma linha separada de clock, ao passo que na assíncrona, a sincronização é feita pelos caracteres de START e STOP.

Desta forma, para transmitir um caractere de forma assíncrona, serão necessários 10 bits (8 de dados mais 1 START e 1 STOP), ao passo que o mesmo caractere pode ser transmitido com apenas 8 bits no modo síncrono.

No entanto, devido à USART apenas utilizar dois pinos do chip, somente poderemos operar em modo half-duplex, já que uma linha é dedicada à sincronização (clock) e outra aos dados.

Quando operando no modo síncrono com clock interno, os pinos RB1 e RB2 assumem a função de recepção/transmissão de dados (RB1) e saída de clock (RB2).

Vejamos os procedimentos a serem efetuados para programar a USART para trabalhar neste modo:

1. Configurar o registrador SPBRG de acordo com a velocidade de comunicação desejada;
2. Programar a USART para o modo de comunicação serial síncrona com clock interno (SYNC='1', CSRC='1', ambos no registrador TXSTA);
3. Habilitar a USART (SPEN='1', registrador RCSTA);
4. Apagar os bits CREN e SREN no RCSTA;
5. Caso seja utilizada a interrupção, habilitar bits TXIE, PEIE e GIE;
6. Caso seja utilizada transmissão de 9 bits, habilitar bit TX9 no TXSTA;
7. Habilitar a transmissão setando o bit TXEN;
8. Caso seja utilizada a transmissão de 9 bits, carregar o nono bit em TX9D;
9. Carregar o dado a ser transmitido no registrador TXREG (caso o TSR esteja vazio a transmissão se inicie imediatamente);
10. Quando o dado presente no registrador TXREG for copiado para o TSR, o flag TXIF será automaticamente setado. Este flag somente será apagado quando um novo dado for armazenado no registrador TXREG.

### 5.12.8.1 - Observações sobre o Modo Mestre Síncrono

Vejamos algumas observações sobre este modo de operação:

- ◆ Caso o bit TXEN seja apagado ( $\text{TXEN}='0'$ ) durante uma transmissão, ela será abortada imediatamente e o transmissor será ressetado;
- ◆ O flag TXIF será setado pela USART cada vez que o conteúdo do registrador TXREG for copiado para o registrador de transmissão TSR. O flag TXIF somente poderá ser apagado escrevendo-se um novo dado no registrador TXREG;
- ◆ Se durante uma transmissão ativar-se o bit SREN, a transmissão será abortada, o pino de dados (RB1) passará a ser uma entrada, a USART permanecerá no modo de recepção síncrona até que receba um caractere completo. Após isso, o pino de dados reverterá para o modo de transmissão (saída) e a USART retornará normalmente ao modo de transmissão síncrona;
- ◆ Se durante uma transmissão ativar-se o bit CREN, a transmissão será abortada, o pino de dados (RB1) passará a ser uma entrada, a USART permanecerá no modo de recepção síncrona indefinidamente;
- ◆ Lembre-se que o bit BRGH não é utilizado no modo síncrono, devendo ser configurado como '0';
- ◆ Lembre-se também que a comunicação síncrona deve funcionar sempre com um mestre e um escravo. A implementação da USART não permite a comunicação entre dois mestres, dois escravos ou um mestre e dois ou mais escravos.

### 5.12.9 - Recepção Síncrona em Modo Mestre

A recepção no modo síncrono com clock interno segue basicamente os mesmos princípios da recepção assíncrona. Vejamos os passos para configuração da recepção síncrona da USART:

1. Configurar o registrador SPBRG de acordo com a velocidade de comunicação desejada;
2. Selecionar o modo síncrono ( $\text{SYNC}='1'$ ) e clock interno ( $\text{CSRC}='1'$ ) no registrador TXSTA;
3. Apagar os bits SREN e CREN no registrador RCSTA (desabilitando a recepção);
4. Habilitar (caso desejada) a interrupção de recepção ( $\text{RCIE}='1'$ );
5. Caso se deseje utilizar o modo de 9 bits, setar o bit RX9 no RCSTA;
6. Caso se deseje receber apenas um caractere, setar SREN; caso se deseje recepção contínua (vários caracteres), setar CREN.

### Observação

Lembre-se que o bit CREN tem prioridade sobre o SREN, o que significa dizer que no caso dos dois estarem setados, teremos o modo de recepção contínua;

7. Monitorar o flag RCIF (ou esperar pela interrupção), o qual irá a nível '1' quando um caractere completo for recebido;
8. Verificar erros de recepção e nono bit (caso habilitado) no registrador RCSTA;
9. Ler o caractere recebido no registrador RCREG. Este procedimento automaticamente apaga o flag RCIF.

#### **5.12.9.1 - Observações sobre Recepção Síncrona em Modo Mestre**

- ◆ No modo de recepção de um único caractere ( $SREN='1'$  e  $CREN='0'$ ), após a recepção de um caractere, a USART automaticamente apaga o bit SREN ( $SREN='0'$ ), cessando a recepção de dados;
- ◆ Para terminarmos a recepção de dados, basta limpar o bit CREN ( $CREN='0'$ );
- ◆ O flag OERR possui o mesmo significado que no modo assíncrono: caso esteja setado, sinaliza que os dois estágios do RCREG estão ocupados e que o terceiro caractere recebido (e que ainda está no RSR) será descartado. Lembre-se que sempre que  $OERR='1'$ , a USART descarta todos os novos caracteres recebidos. Neste caso, devemos providenciar a leitura dos dois caracteres armazenados no RCREG e em seguida o reset do módulo (que é feito desabilitando a recepção ( $CREN='0'$ ));
- ◆ O flag FERR não possui significado no modo síncrono, porquanto não há bits de STOP na comunicação síncrona;
- ◆ A interrupção RCIF permanecerá setada enquanto existirem dados a serem lidos no registrador RCREG.

#### **5.12.10 - Transmissão Síncrona Modo Escravo**

A operação da USART no modo síncrono com clock externo é praticamente idêntica a com clock interno, com as seguintes diferenças:

1. O pino de clock de comunicação (RB2) passa a ser uma entrada;
2. Pelo fato de operar com um clock de comunicação externo, a USART pode operar mesmo com o dispositivo em modo SLEEP.

A operação com clock externo é conseguida colocando o bit CSRC='0'.

Uma característica interessante do modo escravo é que é possível manter o PIC praticamente 90% do tempo em modo SLEEP e mesmo assim termos a comunicação dele com outro dispositivo mestre. Vejamos como isso pode ser feito:

1. Ativar o modo de transmissão da USART (SYNC='1', CSRC='0', TXEN='1', SREN='0' e CREN='0');
2. Habilitar a interrupção de transmissão (TXIE='1');
3. Carregar o primeiro caractere no registrador TXREG. A transmissão tem início imediatamente;
4. Carregar o segundo caractere no TXREG;
5. Entrar em modo SLEEP;
6. Tão logo acabe a transmissão do primeiro caractere, o segundo será carregado no TSR e o flag TXIF será setado;
7. O dispositivo irá "acordar" e caso o GIE='1', irá para o endereço 0x0004, em que uma rotina de tratamento de interrupção deverá providenciar a carga de um novo dado no TXREG;
8. Feito isso, o dispositivo pode retornar ao modo SLEEP e o processo irá se repetir até que não haja mais caracteres a serem transmitidos.

### 5.12.11 - Recepção Síncrona em Modo Escravo

O modo síncrono escravo também funciona de forma praticamente idêntica ao modo mestre, mas com as seguintes diferenças:

1. O pino de clock de comunicação (RB2) passa a ser uma entrada;
2. Pelo fato de operar com um clock de comunicação externo, a USART pode operar mesmo com o dispositivo em modo SLEEP;
3. Somente podemos operar em modo de recepção contínua (CREN='1'). O bit SREN não é utilizado no modo escravo.

Da mesma forma que na transmissão síncrona em modo escravo, a recepção também pode ocorrer em modo SLEEP. Vejamos a forma de fazê-lo:

1. Configurar o registrador SPBRG de acordo com a velocidade de comunicação desejada;
2. Selecionar o modo síncrono (SYNC='1') e clock externo (CSRC='0') no registrador TXSTA;
3. Apagar os bits SREN e CREN no registrador RCSTA (desabilitando a recepção);
4. Habilitar a interrupção de recepção (RCIE='1');
5. Caso se deseje utilizar o modo de 9 bits, setar o bit RX9 no RCSTA;
6. Habilitar a recepção contínua de caracteres (CREN='1');

7. Entrar em modo SLEEP;
8. O flag RCIF irá a nível '1' quando um caractere completo for recebido, fazendo com que o dispositivo "acorde";
9. A rotina de interrupção (caso GIE='1') efetua a leitura do RCSTA e RCREG, verifica erros e armazena o caractere recebido;
10. O dispositivo retorna ao modo SLEEP.

## 5.13 - EEPROM Interna

Alguns membros da série PIC 16 implementam uma pequena porção de memória EEPROM internamente. Essa memória pode ser utilizada para armazenamento de informações ou parâmetros que devem permanecer memorizados mesmo quando o dispositivo for desligado.

Na série 16 a capacidade dessa memória está limitada a um máximo de 256 bytes. Já na série 18 encontramos dispositivos com até 1024 bytes de memória EEPROM.

Como já foi dito anteriormente, o PIC 16F62x possui uma memória EEPROM interna com capacidade de 128 bytes.

Essa memória pode ser acessada com a utilização de quatro registradores SFR:

- ◆ EEADR
- ◆ EEDATA
- ◆ EECON1
- ◆ EECON2

### 5.13.1 - Registrador EEDATA (0x9A)

O registrador EEDATA, localizado no endereço 0x9B do banco de memória 1, é utilizado para a escrita/leitura de dados na memória EEPROM interna.

Nas operações de escrita na memória EEPROM, o conteúdo armazenado nesse registrador é escrito no endereço especificado da memória.

Nas operações de leitura da memória EEPROM, o conteúdo do registrador será atualizado com o dado lido do endereço especificado da memória.

### 5.13.2 - Registrador EEADR (0x9B)

O registrador EEADR, localizado no endereço 0x9B do banco de memória 1, é responsável pelo controle de endereços da memória EEPROM interna.

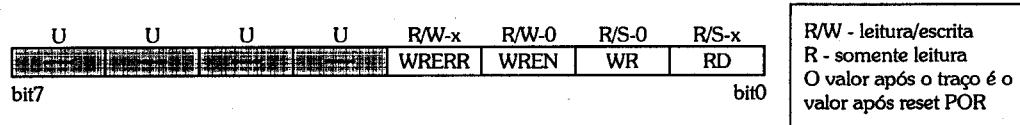
O valor armazenado nesse registrador será utilizado para especificar o endereço da EEPROM nas operações de leitura e escrita dela.

Observe que nos PICs 16F62x, a memória possui uma capacidade de 128 bytes, o que significa que apenas os 7 bits LSB do registrador serão utilizados.

O seu bit 7 não possui função nesses dispositivos e deve ser mantido em nível lógico '0'.

### 5.13.3 - Registrador EECON1 (0x9C)

O registrador EECON1, localizado no endereço 0x9C do banco de memória 1, é utilizado para controlar o acesso à memória EEPROM. Nesse registrador, cada bit possui uma função específica, conforme em seguida:



*Figura 5.59*

**WRERR** - Bit indicador de erro na operação da EEPROM

- 0** - A operação de escrita foi concluída;  
**1** - Indica que uma operação de escrita foi encerrada prematuramente (possivelmente por um reset).

**WREN** - Bit de permissão de escrita

- 0 - Não permite escrita na memória EEPROM;
  - 1 - Permite que se escreva na memória EEPROM;

#### **WR - Bit de acionamento de escrita**

- 0** - Indica que o ciclo de escrita terminou;

**1** - Inicia a operação de escrita na EEPROM (Uma vez iniciada a operação de escrita, este bit permanecerá em '1' até o término dela, quando então a CPU automaticamente o colocará em '0'). Este bit não pode ser colocado em '0' por software. O usuário pode apenas colocá-lo em '1'.

**RD** - Bit de açãoamento de leitura

- 0** - Indica que o ciclo de leitura terminou;

**1** - Inicia uma operação de leitura da EEPROM (Uma vez iniciada a operação, este bit permanecerá em '1' até o término dela, quando então a CPU automaticamente o colocará em '0'). Este bit não pode ser

colocado em '0' por software. O usuário pode apenas colocá-lo em '0'. A operação de escrita demora um ciclo de instrução para ser executada.

Em operações de leitura da EEPROM, o endereço deve ser armazenado previamente no registrador EEADR e em seguida o bit RD deve ser setado. Instantes após, o dado lida da memória EEPROM é armazenado no registrador EEDATA e o bit RD é apagado. Observe que o bit RD somente pode ser apagado pelo hardware do PIC.

Nas operações de escrita, o endereço deve ser armazenado previamente no registrador EEADR e o dado a ser escrito deve ser armazenado no registrador EEDATA. Feito isso, seta-se o bit WREN, escreve-se a seqüência 0x55 e 0xAA no registrador EECON2 e seta-se o bit WR, dando início a operação de gravação da EEPROM.

Ao término da operação de escrita da EEPROM, o hardware irá automaticamente apagar o bit WR e setar o flag EEIF, indicando o término de uma operação de escrita na EEPROM.

#### ***Observação***

Para garantir a integridade dos dados gravados na memória EEPROM interna, devemos sempre após o término bem sucedido de uma operação de escrita, apagar o bit WREN de forma a evitar possíveis escritas indesejadas na memória.

### **5.13.4 - Registrador EECON2 (0x9D)**

O registrador EECON2, localizado no endereço 0x9D do banco de memória 1, é utilizado para controlar o acesso de escrita na memória EEPROM.

Este foi um artifício encontrado pelo fabricante para evitar escritas indesejadas na memória EEPROM.

Assim, sempre que desejarmos escrever algum valor na EEPROM, devemos antes armazenar no EECON2 uma seqüência de valores predefinidos (como se fosse uma senha). Se essa seqüência não for seguida, a operação de escrita não pode ser completada.

Essa seqüência é implementada da seguinte forma: primeiramente armazena-se o valor 0x55 no registrador EECON2 e logo após o valor 0xAA.

Feito isso, a memória EEPROM estará pronta para a execução da operação de escrita.

### **5.13.5 - Exemplos de Acesso à Memória EEPROM Interna**

Em seguida temos dois exemplos de acesso à memória EEPROM, sendo uma operação de leitura do endereço 0x05 e outra de escrita do valor 0xBA no endereço 0x10:

**Leitura:**

BCF	STATUS, RP0	; seleciona banco 0
MOVLW	0x05	; coloca o endereço 05h
MOVWF	EEADR	; no registrador EEADR
BSF	STATUS, RP0	; seleciona banco 1
BSF	EECON1, RD	; inicia leitura da EEPROM
BCF	STATUS, RP0	; seleciona banco 0
MOVF	EEDATA, W	; copia o dado da EEPROM ; para o registrador W

**Escrita:**

MOVLW	0x10	; copia o endereço 10h
MOVWF	EEADR	; para o EEADR
MOVLW	0xBA	; copia o dado BAh a ser escrito
MOVWF	EEDATA	; para o EEDATA
BSF	STATUS, RP0	; seleciona banco 1
BCF	INTCON, GIE	; desliga interrupções
BSF	EECON1, WREN	; habilita escrita na EEPROM
MOVLW	0x55	; escreve 0x55 ...
MOVWF	EECON2	; no EECON2
MOVLW	0xAA	; escreve 0xAA ...
MOVWF	EECON2	; no EECON2
BSF	EECON1, WR	; inicia escrita
BTFSC	EECON1, WR	; testa se a escrita terminou, se terminou ; pula a próxima instrução
GOTO	\$-1	; volta para instrução anterior
BCF	EECON1, WREN	; desabilita escrita na EEPROM
BCF	STATUS, RP0	; seleciona banco 0
BSF	INTCON, GIE	; liga interrupções

## 5.14 - Palavra de Configuração

Os PICs possuem uma posição de memória, situada após o final da memória de programa, utilizada para armazenar configurações do hardware interno do PIC.

Nos PICs da série 16, essa palavra de configuração está armazenada no endereço 2007h que somente é acessível durante a programação do dispositivo:

CP1	CP0	CP1	CP0	—	CPD	LVP	BODEN	MCLRE	FOSC2	PWRTE	WDTE	FOSC1	FOSCO
bit7													bit0

Figura 5.60

**CP1,**

**CP0** - Proteção de código. Utilizado para proteger o chip contra leitura:

16F627 (1k de memória)		
CP1	CP0	Área de memória protegida
1	1	Proteção desligada
1	0	Proteção desligada
0	1	0x0200 até 0x03FF
0	0	0x0000 até 0x03FF

16F628 (2k de memória)		
CP1	CP0	Área de memória protegida
1	1	Proteção desligada
1	0	0x0400 até 0x07FF
0	1	0x0200 até 0x07FF
0	0	0x0000 até 0x07FF

### ***Observação***

Existem dois bits CP1 e dois CP0, que devem ser programados de forma idêntica.

**CPD** - Proteção de dados da EEPROM. Impede a leitura externa da EEPROM:

- 0** - Memória EEPROM interna desprotegida;
- 1** - Memória EEPROM interna protegida contra leituras externas;

### ***Observação***

Caso o CPD esteja igual a '1' e numa nova programação seja colocado em '0', o conteúdo da EEPROM será inteiramente apagado.

**LVP** - Habilitação de programação com baixa tensão

- 0** - Programação por baixa tensão desabilitada. Pino RB4 atua como I/O e a programação é feita pela aplicação de alta tensão ( $\geq 13V$ ) no terminal MCLR;
- 1** - Programação por baixa tensão habilitada. O pino RB4 passa a ser a entrada de programação.

### ***Observação***

Ao ativar o modo de programação por baixa tensão, o pino RA5 passará automaticamente a função de MCLR, independentemente do estado do bit MCLRE.

**BODEN** - Habilitação do reset por Brown-out:

- 0** - Desligado
- 1** - Ligado

**Observação**

Ao ativar o BODEN, o PWRT é automaticamente ligado, independente de estar ou não ativado na palavra de configuração.

**MCLRE - Habilitação do MCLR :**

- 0** - O pino RA5/MCLR funciona como entrada de dados. O MCLR fica ligado internamente ao +V<sub>DD</sub>.
- 1** - O pino RA5/MCLR funciona como entrada de reset.

**Fosc2** - Veja tabela adiante**PWRTE - Habilita temporizador de power-up (PWRT):**

- 0** - Temporizador desligado.
- 1** - Temporizador ligado.

**WDTE - Habilita temporizador Watchdog (WDT):**

- 0** - Watchdog desligado.
- 1** - Watchdog ligado.

**Fosc1**,**Fosc0 - Seleção de configuração do oscilador interno:**

<b>Fosc2</b>	<b>Fosc1</b>	<b>Fosc0</b>	<b>Modo</b>	<b>Descrição</b>
1	1	1	ER	Oscilador com resistor em RA7, RA6 saída de clock
1	1	0	ER	Oscilador com resistor em RA7, RA6 disponível como I/O
1	0	1	INTRC	Oscilador 4Mhz interno, RA6 saída de clock, RA7 disponível como I/O
1	0	0	INTRC	Oscilador 4Mhz interno, RA6 e 7 disponíveis como I/O
0	1	1	EC	Clock externo em RA7, RA6 disponível como I/O
0	1	0	HS	Cristal/resonador de alta freqüência ligado a RA6 e 7
0	0	1	XT	Cristal/resonador conectado a RA6 e 7
0	0	0	LP	Cristal de baixa freqüência conectado a RA6 e 7

## 5.15 - Circuito de Clock

Como já foi dito, os PICs 16F62x possuem um oscilador interno capaz de operar em oito diferentes modos:

- ◆ Cristal de baixa potência (até 200Khz);
- ◆ Cristal/Ressonador cerâmico(até 4Mhz);
- ◆ Cristal/Ressonador cerâmico de alta freqüência (até 20Mhz);
- ◆ Resistor externo sem saída de clock;
- ◆ Resistor externo com saída de clock;
- ◆ Oscilador RC interno sem saída de clock;
- ◆ Oscilador RC interno com saída de clock;
- ◆ Clock externo.

### 5.15.1 - Cristal/Ressonador

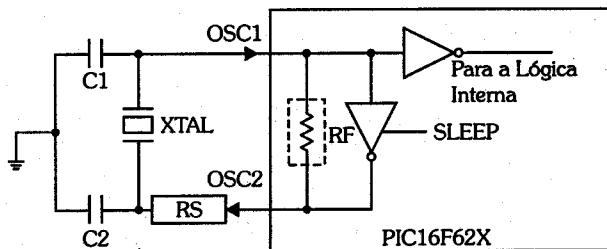
Utilizando um cristal de quartzo ou ressonador e dois capacitores, podemos implementar um oscilador de clock para o PIC. O modo do oscilador deve ser configurado em função da freqüência do cristal/rezonador utilizado:

<200 khz - modo LP

>200khz e <4Mhz - modo XT

>4Mhz - modo HS

Os capacitores C1 e C2 variam de acordo com a freqüência utilizada, estando normalmente entre 10 e 33pF.



**Figura 5.61**

#### Observação

O resistor RS pode ser necessário se forem utilizados cristais de quartzo com corte AT (neste caso esse resistor deve ser de baixo valor  $100\Omega$ ).

## 5.15.2 - Oscilador com Resistor Externo

Neste modo, utiliza-se um simples resistor externo conectando o pino RA7 ao terra, para gerar o clock para a CPU. O valor para esse resistor deve estar entre 38k e 1MΩ e o oscilador é capaz de trabalhar desde 10Khz até 8Mhz.

O pino RA6 pode ser configurado de duas formas, dependendo do valor escrito na palavra de configuração:

- ◆ Utilizado como I/O (RA6);
- ◆ Utilizado como saída de clock ( $F_{osc}/4$ ).

Observe ainda que, dependendo do estado do bit OSCF no registrador PCON, podemos selecionar entre duas freqüências de operação:

- 1** - Freqüência definida pelo resistor externo;
- 0** - Clock de 37Khz.

### ***Observação:***

A freqüência de 37khz é fixa e não depende do resistor externo.

## 5.15.3 - Oscilador Interno de 4Mhz

Este é o modo que permite a maior disponibilidade de pinos no PIC, já que pode liberar os pinos RA6 e RA7.

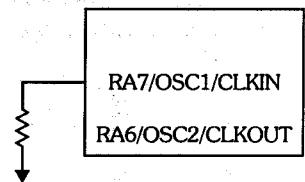
Quando operando neste modo, o clock da CPU será derivado de um clock de 4Mhz obtido a partir de um oscilador RC interno.

Esse oscilador possui razoável precisão com um valor típico de 4Mhz, com mínimo de 3,65 e máximo de 4,28 Mhz.

Dependendo da programação selecionada na palavra de configuração, podemos configurar o pino RA6 como uma saída de clock (de aproximadamente 1Mhz).

Neste modo também podemos selecionar, por meio do bit OSCF no registrador PCON, entre duas freqüências de operação:

- 0** - Clock de 4Mhz interno;
- 1** - Clock de 37Khz.



**Figura 5.62**

## 5.15.4 - Clock Externo

Podemos também utilizar uma fonte externa de clock autônoma. Neste caso, devemos configurar o modo do oscilador para EC.

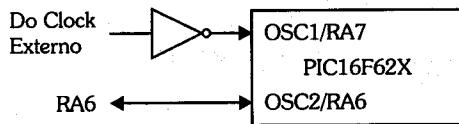


Figura 5.63

## 5.16 - Temporizador de Power-Up

O temporizador de power-up (PWRT) pode ser utilizado para fazer com que o chip permaneça em reset por aproximadamente 72ms após o chip ter sido ligado. Ele deve ser utilizado nos casos em que o chip não inicializa corretamente devido a um tempo muito elevado para estabilização da fonte de alimentação ou do circuito de reset externo.

Esse temporizador pode ser habilitado ou desabilitado somente durante a programação do chip.

## 5.17 - Detector de Brown-Out

O detector de Brown-out é utilizado para provocar o reset da CPU no caso de a tensão de alimentação cair abaixo de um determinado limite (aproximadamente 4 V).

Uma vez que a tensão retorne ao seu valor nominal, o circuito BOD irá retirar o sinal de RESET, permitindo que a CPU reinicie o programa.

Observe que nos PICs da série 16 (e também 12, 14 e 17) a tensão de Brown-out é prefixada e não pode ser alterada. Por isso, se desejarmos utilizar tensões de alimentação inferiores ou muito próximas aos 4V, teremos de utilizar uma técnica de detecção de Brown-out externa, como demonstrado na figura ao lado:

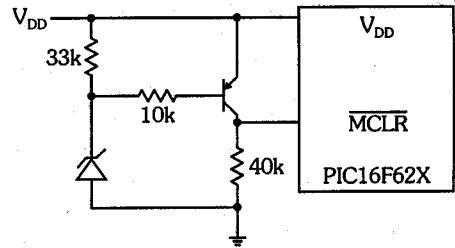


Figura 5.64

O circuito apresentado irá provocar o reset do MCU toda vez que a tensão de alimentação cair abaixo de  $V_z$  (tensão zener) + 0,7.

Observe também que se for utilizada uma baixa tensão de alimentação, devemos sempre desligar o circuito detector de Brown-out interno (colocando o bit BODEN = '0' na palavra de configuração).

### Observação

Quando habilitado, o BOD também ativa automaticamente o PWRT (Power Up Timer), independentemente da configuração dele na palavra de configuração.

## 5.18 - Watchdog

O watchdog (cão de guarda) consiste em um temporizador interno RC que pode ser utilizado para os seguintes fins:

- ◆ Temporizador de período fixo;
- ◆ Medição de temperatura;
- ◆ Proteção contra travamentos de programa;
- ◆ "Acordar" o dispositivo de um estado SLEEP.

A principal aplicação (e motivo de criação) do watchdog é sem dúvida a proteção de processos em casos críticos em que se deve evitar ao máximo a parada ou travamento do programa em execução.

A proteção oferecida pelo watchdog baseia-se na filosofia de que se o programa estiver funcionando corretamente, ele deve periodicamente ressetar o watchdog (ou seja, o programador deve fazer com que o programa periodicamente execute uma instrução CLRWDT de forma a ressetar o watchdog **antes** do seu estouro de contagem).

Se o programa, por algum motivo, deixar de ressetar o watchdog (por uma falha interna, erro de programa, etc.), ele irá atingir o final de sua contagem e provocará um reset da CPU, reiniciando o programa.

Normalmente, o tempo para contagem do watchdog é de 18ms (com prescaler 1:1). Isto é, se o programa permanecer 18ms ou mais sem ressetar o watchdog, este provocará um reset da CPU. Esse período de contagem pode ser alterado com o auxílio do prescaler, conforme a tabela apresentada mais à frente.

Observe que o reset provocado pelo watchdog pode ser constatado pelo software pela simples análise do bit TO no registrador STATUS.

Caso o bit esteja em nível '0', estará indicando que ocorreu um reset provocado pelo watchdog. Este bit somente pode ser modificado por uma instrução CLRWDT, que irá colocá-lo em nível '1'.

Mas a proteção do programa, como já dissemos, não é a única aplicação do watchdog. Muitos projetistas utilizam-no para efetuar medições de temperatura!

A teoria para tais projetos origina-se do fato de que o oscilador RC interno do watchdog possui uma determinada sensibilidade à temperatura. Isto significa que o watchdog varia o período do seu time-out (estouro de contagem) de acordo com a temperatura ambiente em que o chip se encontra.

Outra aplicação para o temporizador watchdog é "acordar" o dispositivo de um estado SLEEP.

Neste caso, o estouro de contagem do watchdog irá fazer com que o dispositivo saia do modo SLEEP. Repare que quando em modo SLEEP, o estouro de contagem do watchdog não provoca um reset da CPU!

Finalmente, uma outra aplicação para o watchdog é a de temporizador com período fixo. Na verdade, o período pode ser alterado (dentro das possibilidades do prescaler), podendo ir desde 18ms até 2,304 segundos! Veja a tabela em seguida:

<b>PS2</b>	<b>PS1</b>	<b>PS0</b>	<b>Período aproximado do watchdog</b>
0	0	0	18 ms
0	0	1	36 ms
0	1	0	72 ms
0	1	1	144 ms
1	0	0	288 ms
1	0	1	576 ms
1	1	0	1152 ms
1	1	1	2304 ms

Observe que os valores desta tabela são válidos para o caso de o bit PSA estar em nível lógico '1', ou seja, o prescaler estando designado para o watchdog.

É claro que o uso do watchdog como temporizador de uso geral apresenta diversas limitações e dificuldades de software, mas pode representar uma alternativa razoável em situações em que a precisão não seja um fator determinante.

Não se esqueça também de que o prescaler utilizado com o watchdog é aquele mesmo do timer 0, e que ele somente pode ser utilizado por um deles, cabendo ao projetista decidir a qual dispositivo conectar o prescaler.

Sendo assim, o uso do watchdog deve ser muito bem analisado, pois se mal utilizado, pode gerar problemas em vez de solucioná-los!

E lembre-se de que a opção pelo uso do Watchdog deve ser feita antes da programação do chip, já que o bit de controle dele está localizado na palavra de configuração do PIC.

## 5.19 - Modo SLEEP

O PIC possui também um modo de baixo consumo de energia, em que a maioria de seus periféricos internos é desligada e a execução do programa é paralisada. Neste modo, o consumo do PIC cai abaixo de  $2\mu A$  (dependendo da configuração de alguns periféricos internos).

Para colocar o chip no modo de baixo consumo, é necessária apenas a execução da instrução SLEEP.

Uma vez em modo SLEEP, as únicas formas de retornar o chip ao modo normal são:

- ◆ Reset externo por meio do pino MCLR (o que provoca obviamente o reset do programa também);
- ◆ Término da contagem do Watchdog (caso ele esteja ligado);
- ◆ A ocorrência de uma interrupção (desde que o respectivo flag individual de habilitação esteja em nível '1').

A forma mais utilizada para retornar à execução é por meio de interrupções.

Neste caso, conforme já estudado, após o evento de interrupção, caso o respectivo controle individual de interrupção esteja habilitado, teremos então o processo de *wake-up*, ou seja, o PIC irá acordar.

As duas últimas formas apresentadas comportam-se em relação ao modo SLEEP essencialmente da mesma forma, fazendo com que o programa reinicie a execução do ponto seguinte à instrução SLEEP.

O fluxo do programa após o "acordar" com uso de interrupções vai depender do estado do bit GIE. Maiores detalhes podem ser vistos no estudo do mecanismo de interrupções no capítulo três deste livro.

Observe que após o retorno de um modo SLEEP, teremos o flag PD em nível lógico '0' indicando que o PIC entrou em modo SLEEP durante a execução do programa.

A única forma de fazer com que PD volte ao nível '1' é pela execução de uma instrução CLRWDT.

## 5.20 - Reset

Um outro aspecto importante a ser observado no estudo de um microprocessador ou microcontrolador é o funcionamento e estados dele após um reset.

Os PICs possuem diversos tipos de reset:

- ◆ POR - Power On Reset - é o reset que ocorre quando o chip é ligado;
- ◆ Reset do MCLR durante a operação normal;
- ◆ Reset do MCLR durante o modo SLEEP;
- ◆ Reset do watchdog durante operação normal;
- ◆ Reset do detector de Brown-out (BOR);
- ◆ Em alguns dispositivos encontramos ainda um reset por erro de paridade da memória de programa (PER).

Observe que a maioria dos registradores dos PICs não é afetada por qualquer condição de reset, porém vários dos registradores SFR são forçados a um estado predefinido após um reset.

A tabela seguinte demonstra o estado de alguns flags dos registradores STATUS e PCON relacionados à sinalização de reset.

<u>POR</u>	<u>BOR</u>	<u>TO</u>	<u>PD</u>	<b>Significado</b>
0	x	1	1	Reset Power-on (POR)
1	x	0	x	Estado ilegal, o flag <u>TO</u> é setado no POR
0	x	x	0	Estado ilegal, o flag <u>PD</u> é setado no POR
1	0	x	x	Reset por detecção de brown-out
1	1	0	n	Reset pelo watchdog
1	1	0	0	Sinal de acordar gerado pelo watchdog
1	1	n	n	Reset <u>MCLR</u> em operação normal
1	1	1	0	Reset <u>MCLR</u> durante modo SLEEP

Legenda: x = estado desconhecido, n = estado não alterado

A tabela seguinte demonstra o estado do contador de programa (PC), do registrador STATUS e do registrador PCON para os diversos tipos de reset:

<b>Tipo de Reset</b>	<b>PC</b>	<b>STATUS</b>	<b>PCON</b>
Power-on (POR)	0x0000	00011xxxx	----1-0x
MCLR durante operação normal	0x0000	000nnnnnn	----1-nn
MCLR durante modo SLEEP	0x0000	00010nnnn	----1-nn
Watchdog	0x0000	00001nnnn	----1-nn
Watchdog durante modo SLEEP	PC+1	nnn00nnnn	-----nn
Detecção de brown-out	0x0000	00011nnnn	----1-n0
Interrupção durante modo SLEEP	PC+1 *	nnn10nnnn	-----nn

Legenda: x = estado desconhecido

- = bit não implementado (lido como '0')

n = estado não alterado

\*Note que, conforme já visto, a instrução localizada após a instrução SLEEP (no endereço PC+1) será sempre executada, mas o fluxo do programa após esse ponto vai depender do estado do bit GIE.

Na tabela seguinte temos o estado dos registradores após diversos tipos de reset:

<b>Registrador</b>	<b>Endereço</b>	<b>Reset Power-on (POR)</b>	<b>Reset de MCLR durante operação normal</b>	<b>Wake-up pelo estouro de contagem do watchdog</b>
			<b>Reset de MCLR durante modo SLEEP</b>	
			<b>Reset Watchdog</b>	<b>Reset por detecção de Brown-out</b>
				<b>Wake-up por interrupção</b>
W	-	xxxxxxxx	nnnnnnnn	nnnnnnnn
INDF	0x00	xxxxxxxx	nnnnnnnn	nnnnnnnn
TMR0	0x01	xxxxxxxx	nnnnnnnn	nnnnnnnn
PCL	0x02	00000000	00000000	PC+1
STATUS	0x03	00011xxx	000ccnnn	nnnccnnn
FSR	0x04	xxxxxxxx	nnnnnnnn	nnnnnnnn
PORTA	0x05	xxxxxxxx	xxnnnnnn	xxnnnnnn
PORTB	0x06	xxxxxxxx	xxxxxxxx	xxxxxxxx
T1CON	0x10	--000000	--nnnnnn	--nnnnnn
T2CON	0x12	-0000000	-0000000	-nnnnnnn
CCP1CON	0x17	--000000	--000000	--nnnnnn
RCSTA	0x18	0000-00x	0000-00x	nnnn-nnn
CMCON	0x1F	00000000	00000000	nn--nnnn
PCLATH	0x0A	---00000	---00000	---nnnnn
INTCON	0x0B	0000000x	0000000n	nnnnnccc
PIR1	0x0C	00000000	00000000	nnnnnnnn
OPTION_REG	0x81	11111111	11111111	nnnnnnnn
TRISA	0x85	11111111	11111111	nnnnnnnn
TRISB	0x86	11111111	11111111	nnnnnnnn
PIE1	0x8C	00000000	00000000	nnnnnnnn
PCON	0x8E	----1-0x	----1-nc	----n-nn
TXSTA	0x98	0000-010	0000-010	nnnn-nnn
EECON1	0x9C	----x000	----c000	----nnnn
VRCON	0x9F	000-0000	000-0000	nnn-nnnn

Legenda: x = estado desconhecido

n = não altera o estado

c = depende de condição

- = bit não implementado (é lido como '0')

Finalmente, devemos lembrar que durante e após um reset, os pinos de I/O passam a estar configurados como entradas (já que os registradores TRIS são carregados

com o valor 0xFF (B'11111111') e assim permanecerão até serem configurados de outra forma pelo programa do usuário.

### ***Observação***

O conteúdo dos registradores GPR não é alterado por nenhum tipo de reset de sistema e o estado dos mesmos após o dispositivo ser ligado é desconhecido.

## **5.21 - ICSP**

A técnica de programação da memória de programa dos PICs é chamada de ICSP, do inglês **In Circuit Serial Programming** (Programação Serial no Circuito), e consiste num protocolo síncrono de duas linhas, permitindo inclusive a programação do chip dentro do próprio circuito de aplicação.

Como vantagens do ICSP podemos destacar:

- ◆ Capacidade de programar o chip dentro do próprio circuito final;
- ◆ Economia de custos;
- ◆ Maior flexibilidade de atualização do software.

O ICSP é baseado em cinco linhas:

- ◆ Clock Serial (RB6);
- ◆ Dados Seriais (RB7);
- ◆ Tensão de programação/Reset (MCLR);
- ◆ Tensão de alimentação ( $V_{DD}$ );
- ◆ Terra ( $V_{SS}$ ).

O modo de programação serial pode ser ativado de duas formas:

1. Pela elevação da tensão no pino MCLR até o valor nominal ( $V_{IHH}$ ), enquanto mantêm-se os pinos RB6 e RB7 em nível lógico '0'. O valor de  $V_{IHH}$  deve estar situado entre  $V_{DD}+3,5$  até 13,5 Volts;
2. Pela ativação (nível lógico '1') do pino PGM (quando o modo de programação por baixa tensão (LVP) estiver habilitado na palavra de configuração. Neste caso, o pino MCLR deve ser mantido em nível lógico '1').

Uma vez que o chip esteja no modo de programação/verificação, os pinos RB6 e RB7 assumem a função de entrada de clock (RB6) e entrada/saída de dados (RB7) e o chip permanecerá aguardando um comando do usuário.

Na tabela seguinte podemos observar os comandos e seus efeitos no chip.

Comando	BITS (MSB... LSB)	Dados
Carregar palavra de configuração	X X 0 0 0 0	0, palavra (14 bits), 0
Escrever dado na memória de programa	X X 0 0 1 0	0, dado (14 bits), 0
Ler dado da memória de programa	X X 0 1 0 0	0, dado (14 bits), 0
Incremento de endereço	X X 0 1 1 0	Não tem
Iniciar ciclo de apagamento e programação	0 0 1 0 0 0	Não tem
Iniciar ciclo de programação	0 1 1 0 0 0	Não tem
Escrever dado na memória EEPROM	X X 0 0 1 1	0, dado (14 bits), 0
Ler dado da memória EEPROM	X X 0 1 0 1	0, dado (14 bits), 0
Apagamento completo da memória de programa	X X 1 0 0 1	Não tem
Apagamento completo da memória EEPROM	X X 1 0 1 1	Não tem
Apagamento Geral 1	0 0 0 0 0 1	Não tem
Apagamento Geral 2	0 0 0 1 1 1	Não tem

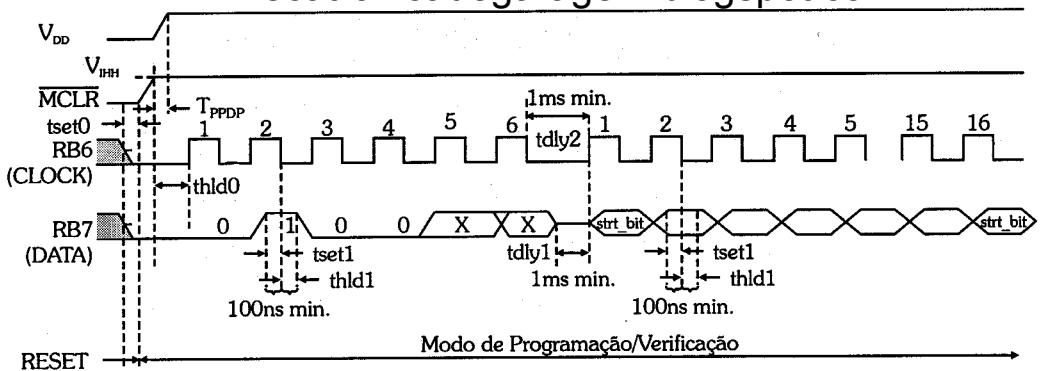
Os comandos devem ser transmitidos iniciando-se pelo bit LSB e são amostrados a cada borda de descida do clock aplicado no pino RB6. Observe ainda que deve haver um intervalo de no mínimo 1μs.

A seguir, veremos uma breve descrição de cada comando:

- ◆ Carga de configuração - após receber o comando de carga de palavra de configuração, o contador de programa (PC) será carregado com o endereço 0x2000, e os próximos 16 bits recebidos serão utilizados da seguinte forma: primeiro bit de início de transmissão (START), os 14 bits seguintes da palavra de configuração e ao final um bit de parada (STOP). Os bits da palavra de configuração devem ser transmitidos iniciando pelo LSB. Após a carga da palavra de configuração, a única forma de retornar ao modo de comando/verificação é aplicando um reset ao pino MCLR (ou seja, colocando o pino em nível '0');
- ◆ Escrita na memória de programa - após receber este comando, a CPU irá carregar os próximos 16 bits recebidos da seguinte forma: primeiro bit de início da palavra, os 14 bits seguintes de dados a serem escritos na memória de programa (iniciando pelo LSB) e finalmente um bit de parada. Observe que o contador de programa não é incrementado automaticamente após este comando; isto deve ser providenciado pelo usuário. Na figura 5.65 temos um diagrama de tempo para este comando:

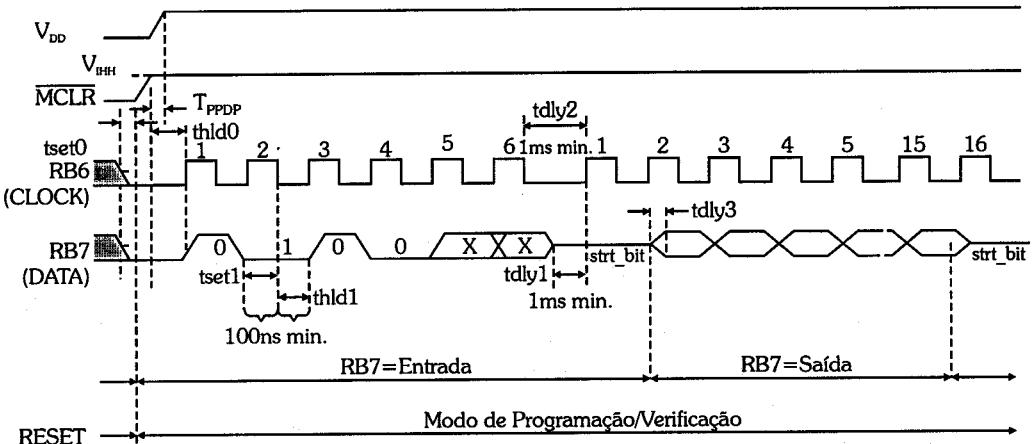
### Observação

Este comando deve ser seguido por um comando de início de ciclo de programação.



**Figura 5.65**

- Ler dado da memória de programa - este comando faz com que a CPU, nos próximos 16 pulsos de clock, descarregue o conteúdo da posição de memória atual indicada pelo PC. A informação deve ser lida da CPU da seguinte forma: primeiro bit de início da palavra, a seguir 14 bits de dados referentes à memória de programa (iniciando pelo LSB) e finalmente um bit de parada. Os bits devem ser lidos pelo programador a cada borda de subida do clock no pino RB6. Na figura 5.66, temos um diagrama de tempo para este comando:



**Figura 5.66**

- Incremento do PC - este comando é utilizado para fazer com que a CPU incremente o valor do contador de programa (PC). Na figura 5.67 temos um diagrama de tempo para este comando:

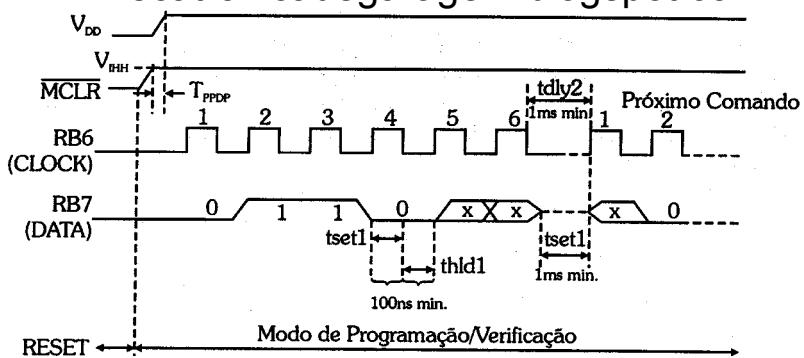


Figura 5.67

- ◆ Iniciar ciclo de apagamento e programação - este comando é utilizado para iniciar um ciclo de apagamento e programação de um endereço da memória do PIC. Esse ciclo inicia-se com o apagamento do conteúdo indicado pelo registrador PC e em seguida a informação que havia sido carregada previamente (por um comando de escrita na memória de programa ou escrita na memória EEPROM) é programada fisicamente no dispositivo. O tempo típico para a duração de um ciclo de apagamento e programação é da ordem de 6 ms e o máximo é da ordem de 13 ms;
- ◆ Iniciar ciclo de programação - este comando, muito parecido com o anterior, também fará com que ocorra a programação de uma posição de memória, mas ao contrário do anterior, não ocorre um ciclo prévio de apagamento. Por isso, para utilizar este comando, é interessante primeiramente que seja comandado um apagamento geral da memória;
- ◆ Escrever dado na memória EEPROM - este comando, de funcionamento idêntico ao de escrita na memória de programa, é utilizado para escrever um dado na memória EEPROM interna. O contador de programa (PC) é utilizado para especificar o endereço da memória EEPROM a ser escrito, mas observe que a capacidade máxima de memória EEPROM da série 16 é de 256 bytes, o que se traduz em 8 bits de endereçamento. Desta forma, apenas os 8 bits menos significativos do PC possuem utilidade. Nos PICs 16F62x temos apenas 128 bytes de memória e assim, somente os 7 bits menos significativos do PC são utilizados. O dado a ser escrito na EEPROM deve ser transmitido após o comando, seguindo o mesmo protocolo: primeiro um bit de início da palavra, a seguir os 8 bits do dado a ser escrito, seguido por mais 6 bits sem função e finalmente finalizado com um bit de parada. O diagrama da figura 5.65 ilustra este comando;
- ◆ Leitura da memória EEPROM - este comando é utilizado para a leitura de uma posição (indicada pelo PC) da memória EEPROM. Após o comando, os dados da memória EEPROM são colocados no pino RB7 a cada borda de subida do clock no pino RB6 e obedecem a seguinte forma: primeiro bit de início de palavra, seguido dos 8 bits do dado da memória EEPROM (iniciando pelo LSB) e mais 6 bits de valor indefinido (devem ser descartados) e finalmente um bit de parada. Veja o diagrama de tempo da figura 5.66;

- ◆ Apagamento completo da memória de programa - este comando é utilizado para promover o apagamento de toda a memória de programa do PIC. Observe que o apagamento somente irá ocorrer após um comando de início de programação. Para utilizar este comando, devemos inicialmente utilizar um comando para escrita na memória de programa com todos os bits de dados em '1', após isso, utiliza-se o comando de apagamento da memória de programa e então o comando de início de ciclo de programação. O apagamento pode demorar até 5 ms para ser concluído. Observe que se a proteção de código estiver ativada, o apagamento não será bem sucedido;
- ◆ Apagamento completo da memória EEPROM - este comando é utilizado para promover o apagamento completo do conteúdo da memória EEPROM do PIC. O apagamento somente tem início após um comando de início de programação. A seqüência deve ser esta:

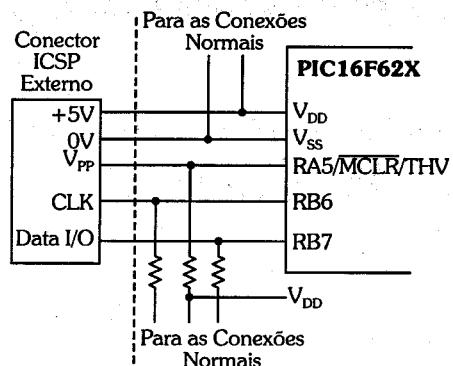
1. Primeiro deve ser escrito um dado (tudo '1') com o comando de escrita na memória;
2. Feito isso, deve ser executado o comando de apagamento da memória EEPROM;
3. Finalmente deve ser executado o comando de início de programação.

O tempo de apagamento pode ser de até 5 ms.

- ◆ Os comandos de apagamento geral 1 e apagamento geral 2 são utilizados para apagar completamente o chip, mesmo estado o mesmo protegido pelo bit de proteção de código. Para fazer isso devemos:
  1. Executar um comando de carga de configuração;
  2. Incrementar o PC até o endereço 0x2007, onde está localizada a palavra de configuração;
  3. Executar o comando de apagamento geral 1;
  4. Executar o comando de apagamento geral 2;
  5. Executar o comando de início de ciclo de apagamento e programação;
  6. Aguardar 10ms;
  7. Executar o comando de apagamento geral 1;
  8. Executar o comando de apagamento geral 2.

O chip estará totalmente apagado ao término deste ciclo de operação, estando apto a ser programado novamente.

Na figura 5.68, temos um diagrama típico da ligação entre um PIC, o circuito de aplicação e o conector de programação ICSP.



**Figura 5.68**

## Linguagens e Ambientes de Programação

Existem diversos softwares que podem ser utilizados para o desenvolvimento de programas para microcontroladores PIC.

Em seguida faremos realizar um breve estudo de um desses softwares: o MPLAB, disponível gratuitamente no site da Microchip: [www.microchip.com](http://www.microchip.com).

Além do MPLAB, veremos também alguns outros softwares que podem ser utilizados no desenvolvimento de programas, testes, simulação e programação de dispositivos.

### 6.1 - MPLAB

O MPLAB é um ambiente integrado de programação (IDE) contendo módulos de edição, montagem e simulação de programas para MCUs PIC.

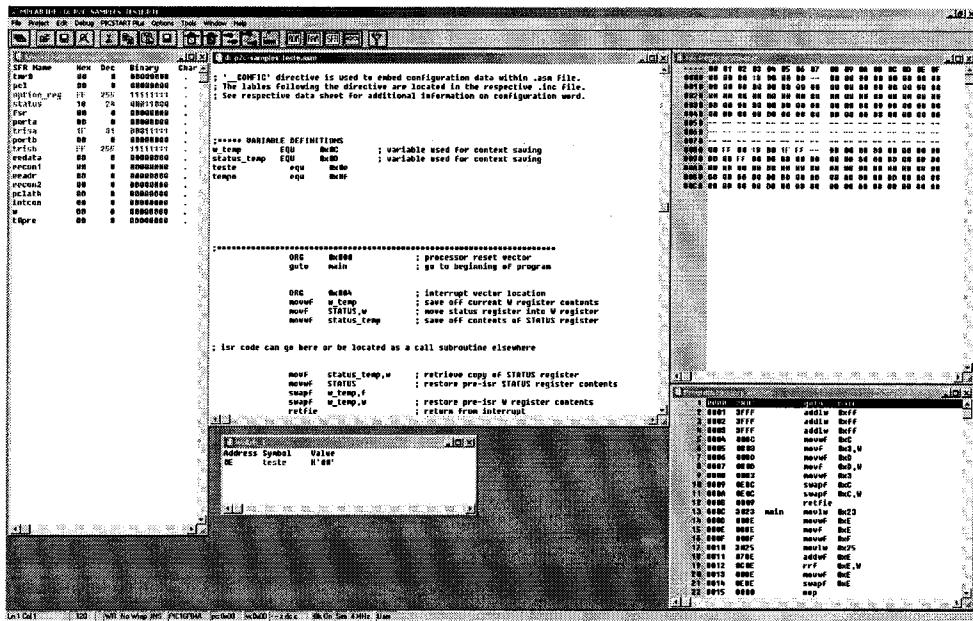


Figura 6.1

A linguagem de programação padrão do MPLAB é o assembly, e para o uso de tal linguagem o programa dispõe de um assembler (montador) chamado MPASM.

Além da assembly, podemos utilizar outras linguagens de programação dentro do MPLAB: Pascal, C, etc. Claro que estas linguagens não fazem parte do pacote do MPLAB e são comercializadas por terceiros.

Na figura 6.1 apresentada anteriormente, podemos verificar a tela do MPLAB no ambiente windows.

Como podemos observar na figura anterior, o MPLAB é composto de diversas janelas destinadas à visualização do programa, registradores SFR, registradores GPR, memória de programa, além de barras de ferramentas com as seguintes funções:

**1. Barra definida pelo usuário:**



**Figura 6.2**

- [Icone de barra de ferramentas] Utilizado para comutar entre os quatro modos da barra de ferramentas disponíveis. Neste modo, se pressionado, muda para a barra de edição;
- [Icone de documento] Abrir arquivo de projeto;
- [Icone de disco] Salvar o projeto atual;
- [Icone de lupa] Procura uma string de texto no arquivo atual;
- [Icone de recorte] Recorta a seleção de texto atual para o clipboard;
- [Icone de cópia] Copia a seleção de texto atual para o clipboard;
- [Icone de colagem] Cola o conteúdo do clipboard no local do cursor;
- [Icone de disco] Salva o arquivo em disco;
- [Icone de janela] Cria nova janela de visualização;
- [Icone de terminal] Make - utilizado para compilar o projeto atual;
- [Icone de terminal] RUN - Tecla de atalho F9 - Inicia a simulação do programa atual;
- [Icone de parada] HALT - Tecla de atalho F5 - Para a execução/simulação do programa;
- [Icone de escada] STEP BY STEP - Tecla de atalho F7 - Execução passo a passo do programa;
- [Icone de escada] STEP OVER - Tecla de atalho F8 - Pula a execução da próxima instrução. Observe que na realidade a próxima instrução é executada, mas o usuário não acompanha a sua execução. Isto é útil em

instruções CALL, quando não se deseja acompanhar a execução da sub-rotina. Neste caso, pular a instrução CALL faz com que a sub-rotina seja executada completamente, sem acompanhamento do usuário;



RESET - Tecla de atalho F6 - Reset da CPU (equivale a um reset MCLR);



Abre/fecha a visualização da memória de programa do PIC;



Abre/fecha a visualização dos registradores GPR;



Abre/fecha a visualização dos registradores SFR.

## 2. Barra de edição:



*Figura 6.3*



Utilizado para comutar entre os quatro modos da barra de ferramentas disponíveis. Neste modo, se pressionado, muda para a barra de depuração;



Novo arquivo;



Abrir arquivo (normalmente .ASM);



Salva o arquivo em disco;



Recorta a seleção de texto atual para o clipboard;



Copia a seleção de texto atual para o clipboard;



Cola o conteúdo do clipboard no local do cursor;



Imprime o arquivo atual;



Procura uma string de texto no arquivo atual;



Repete a última procura exatamente;



Substitui o texto;



Repete a última substituição de texto exatamente;



Desfaz a última edição de texto;



Move o texto uma tabulação à direita;



Move o texto uma tabulação à esquerda;

- [Icone de cursor] Move o cursor para uma linha específica;
- [Icone com ícones] Mostra/esconde os números das linhas;
- [Icone com ponto de interrogação] Aciona o help sensível ao contexto.

### 3. Barra de depuração:



**Figura 6.4**

- [Icone com ícone de carro] Utilizado para comutar entre os quatro modos da barra de ferramentas disponíveis. Neste modo, se pressionado, muda para a barra de projeto;
- [Icone com ícone de computador] RUN - Tecla de atalho F9 - Inicia a simulação do programa atual;
- [Icone com ícone de parada] HALT - Tecla de atalho F5 - Para a execução/simulação do programa;
- [Icone com ícone de escada] STEP BY STEP - Tecla de atalho F7 - Execução passo a passo do programa;
- [Icone com ícone de escada com seta] STEP OVER - Tecla de atalho F8 - Pula a execução da próxima instrução. Observe que na realidade a próxima instrução é executada, mas o usuário não acompanha a sua execução. Isto é útil em instruções CALL, quando não se deseja acompanhar a execução da sub-rotina. Neste caso, pular a instrução CALL faz com que a sub-rotina seja executada completamente, sem acompanhamento do usuário;
- [Icone com ícone de circuito] RESET - Tecla de atalho F6 - Reset da CPU (equivale a um reset MCLR);
- [Icone com ícone de contador] PC= Altera o valor do contador de programa;
- [Icone com ícone de seta] Executa uma instrução fora do programa;
- [Icone com ícone de janela] Abre/fecha a janela de modificação de valores;
- [Icone com ícone de parada] Configura condições de parada do programa;
- [Icone com ícone de rastreamento] Configura as opções de rastreamento de programa;
- [Icone com ícone de eventos] Configura os eventos externos de programa;
- [Icone com ícone de limpeza] Limpa qualificadores;
- [Icone com ícone de parada condicional] Parada condicional;

-  Parada do rastreamento do programa;
-  RESET - Tecla de atalho F6 - Reset da CPU (equivale a um reset MCLR);
-  Notas de lançamento do MPLAB.

#### 4. Barra de projeto:



**Figura 6.5**

-  Utilizado para comutar entre os quatro modos da barra de ferramentas disponíveis. Neste modo, se pressionado, muda para a barra do usuário;
-  Novo projeto;
-  Abrir arquivo de projeto;
-  Fechar projeto atual;
-  Salvar o projeto atual;
-  Editar definições de projeto;
-  Monta o projeto atual;
-  Constrói todo o projeto;
-  Constrói somente o nó atual;
-  Instala pacote de linguagem;
-  Ajuda do MPASM.

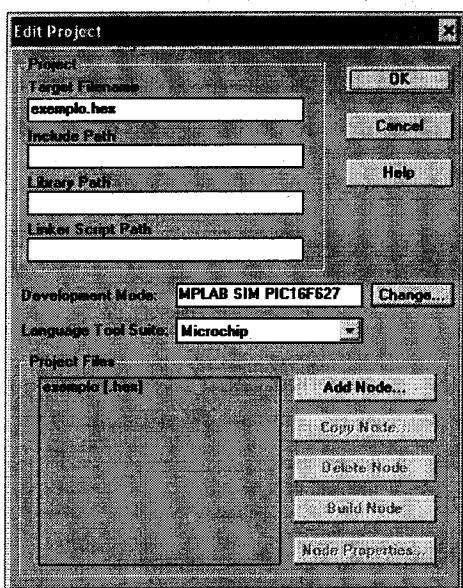
## 6.2 - Criação de um Projeto

Para utilizar o MPLAB, devemos primeiramente criar um arquivo de projeto que conterá os dados relativos ao programa em desenvolvimento, como o(os) nome(s) do(s) arquivo(s) com o código-fonte (em assembly ou outra linguagem), arquivo de destino da montagem (normalmente terminado com a extensão .HEX), etc.

Para criar um novo projeto a partir dos menus, clique em Project > New Project.

Uma janela se abre pedindo para o usuário informar o nome do projeto a ser criado: o usuário deve então fornecer o nome do arquivo de projeto e teclar <ENTER>.

Surgirá então uma janela de opções do projeto, como a da figura seguinte:



**Figura 6.6**

O Target Filename é o nome do arquivo que será gerado após a montagem/compilação do programa e possui a extensão .HEX. Esse arquivo binário é utilizado para a programação de dispositivos e na simulação de programas.

Normalmente o nome desse arquivo é o mesmo do projeto, mas isto pode ser alterado pelo usuário.

Nessa janela podemos ainda selecionar o tipo de projeto que está sendo desenvolvido. Essa informação está disponível em Development Mode. O modo pode ser alterado clicando no botão <CHANGE>.

Podemos desenvolver um projeto em um dos seguintes modos:

- ◆ None - neste modo somente podemos fazer a edição / compilação de programas;
- ◆ MPLAB SIM - neste podemos editar, compilar e simular o funcionamento de um programa dentro do MPLAB;
- ◆ MPLAB ICE - utiliza-se quando em conjunto com o MPLAB ICE 2000- In Circuit Emulator (emulador no circuito), um equipamento que permite a simulação do funcionamento do PIC no próprio circuito de aplicação. Neste modo podemos fazer uma simulação em tempo real do funcionamento do programa, bem como para a sua execução, verificar variáveis, registradores, etc;
- ◆ PICMASTER Emulator - é utilizado em conjunto com o PICMASTER, um emulador no circuito (como o ICE 2000);

- ◆ ICEPIC - outro modo utilizado para emuladores de hardware. O ICEPIC, no caso, é um emulador de hardware de menor custo que o ICE 2000;
- ◆ MPLAB ICD Debugger - esse modo é utilizado em conjunto com o ICD - *In Circuit Debugger*, uma funcionalidade adicionada à família 16F87x e que permite o debug serial do chip no próprio circuito de aplicação (com algumas limitações);

Caso seja pressionado o botão <CHANGE>, teremos a janela apresentada em seguida para alteração do modo de desenvolvimento:

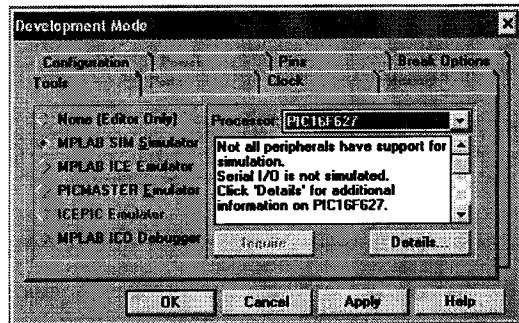


Figura 6.7

Para a simulação do programa no micro, certifique-se de que esteja selecionado "**MPLAB SIM Simulator**" o "**Processor**" seja o MCU utilizado no projeto, no nosso caso, por exemplo, o PIC 16F627.

Se já existir um arquivo com o programa, o usuário pode adicioná-lo ao projeto. Para isso, basta clicar em <ADD NODE> para adicionar o arquivo do programa ao projeto atual.

Clique em <OK> e aparecem duas janelas de advertência. Apenas clique <OK>.

Caso ainda não exista um arquivo com o programa, após clicar em <OK>, na janela da figura 6.8 o usuário deve então criar um novo arquivo para digitar o programa.

Para criar um novo programa, clique em File > New no menu principal.

Após a digitação do programa selecione a opção File > Save no menu principal. Na janela que aparece pedindo o nome do arquivo a ser salvo digite o nome dele e em seguida clique em <OK>.

Vá até as opções do projeto em Project > Edit Project no menu principal. Será apresentada a mesma janela da figura 6.9. Clique no botão (Add Node) e aparece uma janela para selecionar o nó a ser adicionado, ou seja, o programa assembly que acabamos de salvar. Selecione o arquivo e clique <OK>.

Agora o projeto está pronto e já é possível compilar o programa e também simular sua execução.

## 6.3 - Compilação e Simulação de um Programa

Uma vez que o projeto esteja criado e perfeitamente definido, é necessário proceder à montagem do programa.

A montagem é o processo pelo qual o montador traduz as instruções assembly em códigos binários que podem ser interpretados pelo microcontrolador.

Para proceder à montagem, basta pressionar a tecla <F10> ou o item de menu PROJECT > MAKE PROJECT.

O montador procederá então à montagem do programa e ao final gera uma janela com os resultados da compilação (chamada BUILD RESULTS).

Nessa janela, podemos verificar o processo de montagem passo a passo, por meio de mensagens escritas pelo montador, inclusive com a indicação dos eventuais erros encontrados.

Podemos classificar as mensagens do montador em três categorias básicas:

- ◆ MESSAGE - as mensagens são informações sobre algumas condições encontradas pelo MPASM durante a montagem do programa;
- ◆ WARNING - são mensagens de aviso emitidas pelo montador. Elas não impedem a montagem do programa, mas podem indicar erros ou descuidos do programador, que podem provocar um funcionamento errático do programa. As mensagens de aviso obedecem ao seguinte formato:

WARNING [<número>] <arquivo> <linha> (<texto>) onde:

- ◆ <número> é o número interno da mensagem;
- ◆ <nome do arquivo> é o nome completo do arquivo que estava sendo montado quando o evento foi detectado;
- ◆ <linha> é o número da linha do arquivo a qual gerou o aviso;
- ◆ <texto> é o conteúdo de texto da linha especificada.
- ◆ ERROR - as mensagens de erro são emitidas pelo montador quando um erro de montagem é detectado. Os erros de montagem impedem a finalização da montagem e por isso devem sempre ser solucionados antes de uma montagem bem sucedida. As mensagens de erro obedecem ao seguinte formato:

ERROR [<número>] <arquivo> <linha> (<texto>) onde:

- ◆ <número> é o número interno da mensagem de erro;
- ◆ <nome do arquivo> é o nome completo do arquivo que estava sendo montado quando o erro foi detectado;
- ◆ <linha> é o número da linha do arquivo na qual foi detectado o erro;
- ◆ <texto> é o conteúdo de texto gerador do erro.

Uma montagem bem sucedida apresenta a mensagem final: **Build completed successfully**. Caso a montagem não seja bem sucedida, teremos a mensagem final: **Build failed**.

Com o programa montado, já é possível simular a sua execução. Para isto devemos escolher entre os dois modos básicos de execução:

- ◆ Para a execução contínua do programa, basta pressionar a tecla <F9>, ou clicar no ícone RUN. Para parar a execução, basta pressionar a tecla <F5>, ou clicar no ícone HALT;
- ◆ Para a execução passo a passo do programa, basta pressionar a tecla <F7> ou clicar no ícone STEP BY STEP.

O programa pode ser reiniciado a qualquer momento, pressionando a tecla <F6> ou clicando no botão RESET PROCESSOR.

### 6.3.1 - Adição de Janelas de Observação

Durante a simulação do programa, podemos verificar o conteúdo dos registradores por meio das janelas SFR (para os registradores SFR) e RAM (para todo o conteúdo da memória RAM do chip (registradores SFR e GPR)).

No entanto, com o aumento do número de variáveis, torna-se difícil rastrear o conteúdo de cada variável, devido à dispersão delas dentro da memória RAM.

Para solucionar este inconveniente, podemos utilizar um artifício chamado de janela de observação (Watch Window), na qual o usuário pode definir quais símbolos serão apresentados.

Cada símbolo é apresentado com o seu nome, endereço do registrador na memória RAM e o seu conteúdo.

### 6.3.2 - Cálculo do Tempo de Execução

Muitas vezes necessitamos calcular o tempo necessário para executar um determinado programa ou parte dele.

Apesar de esta tarefa ser relativamente simples devido ao tempo de execução relativamente constante para as instruções PIC (um ou dois ciclos de clock, dependendo da instrução), em programas muito extensos ou complexos ela pode ser mais complicada.

Para auxiliar nesta tarefa, podemos utilizar o cronômetro embutido no MPLAB.

Para acessá-lo, basta selecionar WINDOW > STOPWATCH no menu principal. Em seguida, surgirá as janelas apresentadas na figura 6.8.

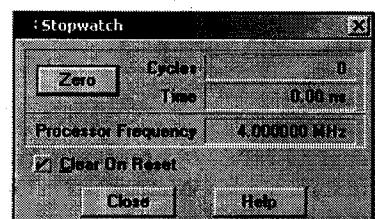


Figura 6.8

Observe que são apresentados o tempo e o número total de ciclos executados.

Para zerar o cronômetro, basta clicar no botão ZERO. Ele também é zerado ao resetar a CPU se a caixa Clear On Reset estiver selecionada.

### 6.3.3 - Linguagem Assembly MPASM

O MPASM é o programa encarregado de traduzir as instruções em linguagem assembly, fornecidas pelo usuário, em códigos binários conhecidos como códigos de máquina. Esses códigos serão então utilizados para programar a memória do PIC desejado.

O MPASM é capaz de traduzir todo o conjunto de instruções dos PICs (tanto da série 16, como também da série 12, 14, 17 e 18).

Observe que além das instruções PIC, o montador também possui alguns comandos internos, destinados a facilitar ou modificar a montagem (tradução) dos programas. Tais comandos são chamados de diretivas e são vistos no apêndice D.

Além das diretivas, encontramos também operadores e comandos especiais que podem ser utilizados pelo programador.

Operadores são elementos que estabelecem uma relação entre dois operandos. Os operadores do MPASM somente são interpretados pelo MPASM. Eles não geram diretamente código de máquina.

O MPASM reconhece os operadores matemáticos (adição (+), subtração (-), divisão (/), multiplicação (\*) e resto da divisão (%)), lógicos (~ complemento (~), rotação à esquerda (<<), rotação à direita (>>), AND bit a bit (&&), OR bit a bit ( | ), XOR bit a bit ( ^ ), AND lógico ( & ) e OU lógico ( || )), relacionais ( igual a (==), diferente de (!=), maior que (>), menor que (<), maior ou igual a (>=), menor ou igual a (<=) e finalmente o operador de atribuição (=):

Em seguida encontramos uma tabela com outros operadores reconhecidos pelo MPASM:

Operador	Descrição
HIGH	Retorna o byte MSB de um valor de 16 bits
LOW	Retorna o byte LSB de um valor de 16 bits
UPPER	Retorna o byte MSB de um valor de 24 bits
\$	Retorna o endereço atual (valor do PC)

O MPASM apresenta ainda um conjunto estendido de instruções que não estão implementadas internamente na CPU dos PICs, mas são traduzidas pelo montador em instruções PIC durante a montagem do programa (tais quais as macro instruções que veremos adiante).

O propósito desse conjunto estendido de instruções é proporcionar uma maior semelhança com os mnemônicos de outros microcontroladores disponíveis comercialmente, além de economizar a escrita de certas seqüências repetitivas de instruções.

Instrução	Descrição	Instruções Equivalentes	Flags e registradores alterados
ADDCF f,d	Soma 1 ao conteúdo do registrador <b>f</b> , caso o flag C esteja em '1', armazenando o resultado no destino <b>d</b> .	BTFSC STATUS,C INCF f,d	Z
ADDDCF f,d	Soma 1 ao conteúdo do registrador <b>f</b> , caso o flag DC esteja em '1', armazenando o resultado no destino <b>d</b> .	BTFSC STATUS,DC INCF f	Z
SUBCF f,d	Decrementa o registrador especificado por <b>f</b> caso o flag C esteja em nível '1', o resultado é armazenado em <b>d</b> .	BTFSC STATUS,C DECF f,d	Z
SUBDCF f,d	Decrementa o registrador especificado por <b>f</b> caso o flag DC esteja em nível '1', o resultado é armazenado em <b>d</b> .	BTFSC STATUS,DC DECF f,d	Z
NEGF f,d	Modifica o resultado negativo no registrador <b>f</b> , o resultado é armazenado no destino <b>d</b> .	COMF f,F INCF f,d	Z registrador f
LCALL k	Chamada de sub-rotina no endereço <b>k</b> , com acerto automático da página da memória de programa.	BSF/BCF PCLATH,3 BSF/BCF PCLATH,4 CALL k	-
LGOTO k	Desvia o programa para o endereço <b>k</b> , com acerto automático da página da memória de programa.	BSF/BCF PCLATH,3 BSF/BCF PCLATH,4 GOTO k	-
B k	Desvia o programa para o endereço definido por <b>k</b>	GOTO k	-
BC k	Desvia para o endereço especificado por <b>k</b> caso o flag C esteja em '1'.	BTFSC STATUS,C GOTO k	-
BNC k	Desvia para o endereço especificado por <b>k</b> caso o flag C esteja em '0'.	BTFSS STATUS,C GOTO k	-
BDC k	Desvia para o endereço especificado por <b>k</b> caso o flag DC esteja em '1'.	BTFSC STATUS,DC GOTO k	-

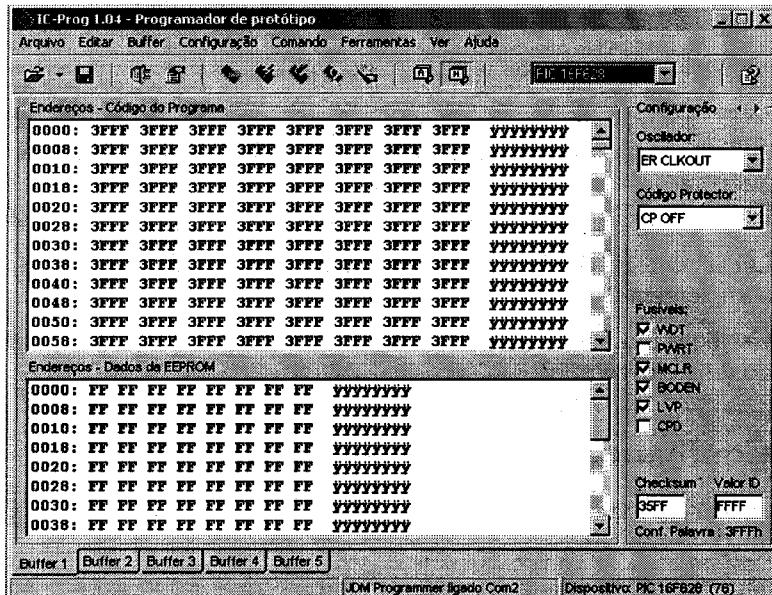
Instrução	Descrição	Instruções Equivalentes	Flags e registradores alterados
BNDC k	Desvia para o endereço especificado por k caso o flag DC esteja em '0'.	BTFS S STATUS,DC GOTO k	-
BZ k	Desvia para o endereço especificado por k caso o flag Z esteja em '1'.	BTFS C STATUS,Z GOTO k	-
BNZ k	Desvia para o endereço especificado por k caso o flag Z esteja em '0'.	BTFS C STATUS,Z GOTO k	-
SKPC	Pula a próxima instrução caso o flag C='1'.	BTFS S STATUS,C	-
SKPNC	Pula a próxima instrução caso o flag C='0'.	BTFS C STATUS,C	-
SKPDC	Pula a próxima instrução caso o flag DC='1'.	BTFS S STATUS,DC	-
SKPNDC	Pula a próxima instrução caso o flag DC='0'.	BTFS C STATUS,DC	-
SKPZ	Pula a próxima instrução caso o flag Z='1'.	BTFS S STATUS,Z	-
SKPNZ	Pula a próxima instrução caso o flag Z='0'.	BTFS C STATUS,Z	-
CLRC	Apaga o flag de Carry (C).	BCF S STATUS,C	C='0'
SETC	Seta o flag de Carry (C).	BSF S STATUS,C	C='1'
CLRDC	Apaga o flag de Digit Carry (DC).	BCF S STATUS,DC	DC='0'
SETDC	Seta o flag de Digit Carry (DC).	BSF S STATUS,DC	DC='1'
CLRZ	Apaga o flag de Zero (Z).	BCF S STATUS,Z	Z='0'
SETZ	Seta o flag de Zero (Z).	BSF S STATUS,Z	Z='1'
MOVFW f	Copia o conteúdo do registrador especificado por f para o registrador W.	MOVF f,W	-
TSTF f	Testa o registrador f para determinar se o seu conteúdo é igual a 0.	MOVF f,F	-

Observe ainda que o caractere ponto-e-vírgula (;) é utilizado para comentários no código. Isto significa que qualquer texto após o caractere (;) é desprezado pelo montador.

### 6.3.4 - IC-Prog

Outro software gratuito que pode auxiliar na tarefa de programação física dos PICs é o IC-Prog. É um software de programação que pode ser utilizado com diversos programadores (entre eles o JDM apresentado no apêndice I deste livro), permitindo também a programação de diversos dispositivos, como: PICs séries 12,16 e 18, memórias EEPROM seriais I<sub>2</sub>C, além de outros microcontroladores.

O software é de utilização e configuração muito simples. Na figura seguinte temos a tela principal do programa.



**Figura 6.9**

Como se pode constatar, o programa é composto de três janelas principais:

- ◆ Endereços - Código do Programa: esta janela apresenta o conteúdo da memória de programa do PIC (no caso da figura 6.9, ela representa um PIC totalmente apagado). Clicando sobre qualquer posição, podemos alterar o seu conteúdo;
- ◆ Endereços - Dados da EEPROM: esta janela permite a visualização do conteúdo da memória EEPROM interna. Esse conteúdo pode ser alterado clicando em qualquer endereço da memória e digitando um novo valor;
- ◆ Configuração : esta janela representa o estado da palavra de configuração interna do PIC. Por meio dela podemos modificar qualquer dos parâmetros da palavra de configuração do dispositivo.

Observe que existe um total de cinco conjuntos de janelas, como as descritas anteriormente. Cada conjunto pode ser acessado clicando na aba correspondente na parte inferior da janela do programa (Buffer 1, Buffer 2, etc.).

Além das janelas, encontramos também ícones como os seguintes:

- Ler tudo - lê o dispositivo que está conectado ao programador;
- Programar tudo - inicia a gravação do dispositivo;
- Limpar tudo - apaga totalmente o dispositivo no programador;
- Verificar - compara o programa no dispositivo com o do buffer atual;
- Hardware - abre a janela de configuração do programador;
- Opções - abre a janela de opções do software.

Uma vez configurado o programador corretamente (por meio do botão de configuração de hardware descrito anteriormente) e selecionado o chip desejado (o que pode ser feito acessando o comando CONFIGURAÇÃO > DISPOSITIVOS > MICROCHIP PIC > MAIS > 16F628 no menu principal), podemos:

- ◆ Ler o conteúdo da memória de programa (e da memória EEPROM, caso ele a implemente) do chip conectado ao programador para dentro do software;
- ◆ Carregar um arquivo (normalmente em formato .HEX) para a programação do dispositivo conectado ao programador.

### 6.3.5 - Configuração do IC-PROG

Vejamos agora como configurar o IC-PROG e o programador JDM para o funcionamento correto:

A princípio, para facilitar o entendimento, vamos configurar o software para operar na língua portuguesa. Para isso, acesse o menu SETTINGS > OPTIONS no menu principal.

Surgirá na tela uma janela como a seguinte:

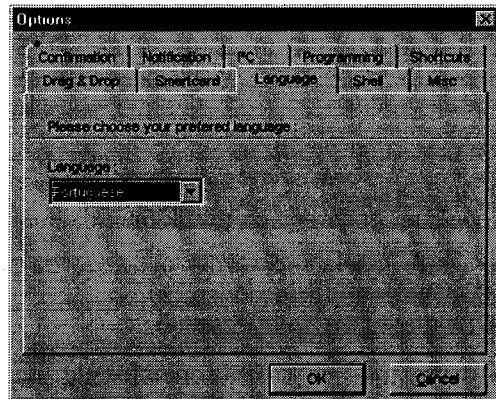
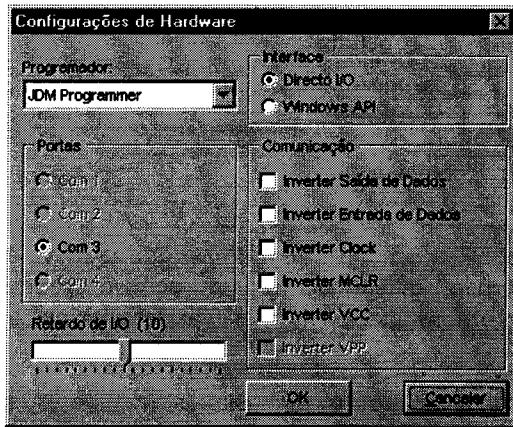


Figura 6.10

Selecione a aba LANGUAGE e em seguida o idioma desejado: PORTUGUESE. Após clicar no botão OK, a janela fecha e o programa será configurado para a língua portuguesa.

Com o programa operando na língua portuguesa, podemos iniciar então a sua configuração para trabalhar com o programador. No presente caso, o programador em questão é o JDM, apresentado no apêndice F.

A configuração do programa em si é bastante simples: pressionando a tecla de função F3, abre-se a janela de configuração de hardware do programa.



**Figura 6.11**

Nesta janela, o usuário deve:

1. Selecionar o programador desejado (no caso: JDM Programmer);
2. Selecionar a porta serial à qual foi ligado o programador (COM1, COM2, COM3 ou COM4. Observe que as portas seriais não disponíveis encontram-se desabilitadas nesta janela);
3. Configurar o tipo de interface utilizado: DIRECT I/O ou WINDOWS API. Normalmente o usuário de sistemas operacionais como o Windows 95, Windows 98 e Windows ME, podem utilizar a opção DIRECT I/O sem maiores preocupações. Para os usuários de sistemas operacionais, como o Windows NT, Windows 2000 ou Windows XP, os quais não permitem, ou restringem o acesso direto ao hardware do computador, deve ser utilizada a opção WINDOWS API. Além disso, no caso do NT, 2000 ou XP, o usuário deve ainda instalar um driver especial (ICPROG.SYS), bem como habilitar a opção correspondente (janela OPÇÕES aba DIVERSOS, opção ativar driver NT/2000);
4. Configurar o fator de retardo de I/O. Este valor varia de acordo com a velocidade do computador. Valores menores propiciam uma maior velocidade de leitura/gravação, no entanto se a velocidade for muito alta, pode ocorrer perda de dados ou erro de leitura ou gravação.

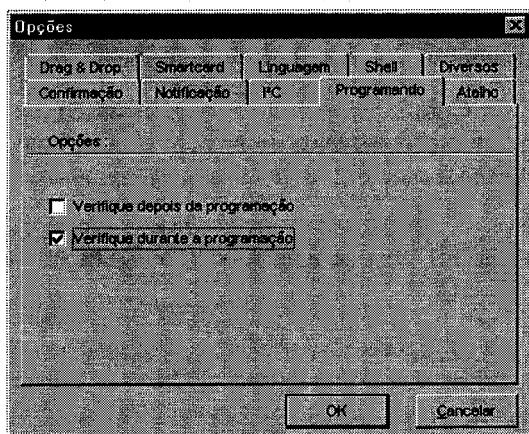
Finalmente, uma outra interessante opção a ser utilizada é a de verificação da programação. O IC-PROG pode verificar automaticamente se a programação do dispositivo foi feita com sucesso ou não, apresentando tal fato ao usuário.

Existem dois tipos de verificação disponíveis:

No primeiro a verificação é feita após a programação, isto é, após o chip ter sido inteiramente programado, o software realiza uma leitura de verificação para testar se os dados foram gravados corretamente.

No segundo, cada endereço após ser programado é em seguida lido para verificação do seu conteúdo. Este modo de verificação é especialmente útil durante a fase de configuração do programador, pois permite detectar erros de programação logo no início do processo.

Feito isso, o programa está pronto para ser utilizado, bastando selecionar o dispositivo adequado e iniciar os trabalhos.



**Figura 6.12**

## 6.4 - Outros Pacotes de Programação

Além dos softwares descritos até aqui, existem outros, entre os quais podemos destacar:

- ◆ Compiladores C: são programas que traduzem uma linguagem de alto nível (C) em código de máquina. Eles são indicados para a construção de programas cujo nível de complexidade inviabilize o uso da linguagem Assembly. Podemos destacar os seguintes compiladores e pacotes de programação C:
  - ◆ CCS C - O pacote de programação C da CCS, conhecido como PCW, constitui-se num ambiente integrado de desenvolvimento (IDE) que permite a edição e compilação de programas em linguagem C. Os compiladores CCS são conhecidos pela sua eficiência de compilação, além da diversidade de tipos de variáveis (incluindo variáveis do tipo ponto flutuante, inteiros, etc., até 32 bits). Os compiladores CCS apresentam também uma grande variedade de bibliotecas de funções e procedimentos prontos para o uso. Maiores detalhes podem ser vistos em <http://www.ccsinfo.com/picc.shtml>;
  - ◆ PIC-C - O PIC-C da Hitech é um compilador C de grande eficiência e que implementa também diversas funções e procedimentos prontos para

o uso. Existe uma versão de download grátis do compilador (PICC lite), no entanto ela é limitada ao PIC 16F84. Os compiladores C da Hitech são bastante utilizados, mas o seu custo muito elevado, impede a utilização pelos pequenos usuários. Maiores detalhes podem ser vistos em <http://www.htsoft.com/>:

- ◆ CC5X - Desenvolvido pela BKD, o CC5X foi um dos primeiros compiladores disponíveis para microcontroladores PIC. O CC5X apresenta um bom nível de otimização, matemática, ponto flutuante até 32 bits, etc. No entanto, assim como os compiladores da Hitech, o CC5X também apresenta um custo elevado, dificultando a sua utilização pelos pequenos usuários e estudantes. Há uma versão gratuita do CC5X, mas está limitada a um máximo de 1024 instruções e variáveis inteiras de até 16 bits. Maiores detalhes podem ser obtidos em <http://www.bknd.com/cc5x/index.shtml>;
- ◆ C2C - Desenvolvido pelo russo Pavel Baranov, é um ótimo compilador, apresentando baixo custo e boa eficiência. Possui versões para Windows e Linux. Existe uma versão gratuita de demonstração, além da versão paga que implementa um nível maior de otimização. Maiores detalhes sobre o compilador podem ser obtidos em <http://www.geocities.com/SiliconValley/Network/3656/c2c/c.html>;
- ◆ C2C++ - O primeiro compilador orientado a objetos para PICs que apresenta um número maior de funções que o seu antecessor C2C. Isto permite a sua utilização tanto como um compilador C melhorado, como um compilador C orientado a objetos.
- ◆ BASIC - A linguagem BASIC é, desde os primórdios da programação de microcomputadores, bastante utilizada pela sua facilidade de aprendizado. Como não poderia deixar de ser, encontramos diversas implementações da linguagem BASIC voltada para PICs, algumas sob a forma de compiladores, outras sob a forma de interpretadores residentes. Os interpretadores são programas armazenados na memória de programa dos PICs, que realizam a tradução em tempo real de uma linguagem de alto nível (no caso o BASIC) para linguagem de máquina. Observe que no caso dos PICs, o programa em BASIC fica normalmente armazenado em uma memória EEPROM externa. Vejamos alguns exemplos de softwares desta linguagem:
  - ◆ BASIC STAMP - Desenvolvidos pela empresa PARALLAX, os BASIC STAMP são circuitos integrados híbridos, construídos com a utilização de MCUs PIC ou Scenix e que contam com memória EEPROM integrada e um interpretador BASIC extremamente poderoso, com inúmeras funções embutidas. Os circuitos integrados híbridos estão disponíveis sob a forma de encapsulamento DIP, podendo substituir os PICs em algumas aplicações. Maiores detalhes podem ser obtidos em: <http://www.parallaxinc.com>;
  - ◆ PIC BASIC - Os compiladores BASIC desenvolvidos pela Crownhill Associates são bastante poderosos e compatíveis com a linguagem BASIC utilizada pela Parallax nos seus BASIC STAMP. No site do

fabricante existem alguns exemplos do que é possível fazer com o compilador: a implementação do jogo Tetris e Pong feita com um 16F84 e com saída de áudio e vídeo para TV. Maiores detalhes podem ser obtidos em: <http://www.picbasic.co.uk/index.htm>.

- ◆ Outros programas: além do C, encontramos também outras linguagens de programação para o PIC:
  - ◆ JAL - O compilador JAL (Just Another Language - Somente uma outra linguagem) mistura Pascal com outras para formar uma linguagem de programação híbrida. O compilador apresenta suporte para diversos PICs da série 16 e 12 e está disponível gratuitamente, representando uma boa opção para o desenvolvimento de aplicações de pequena ou média complexidade em que a velocidade de execução e o tamanho do código não são fatores determinantes. Maiores detalhes podem ser obtidos em <http://www.voti.nl/jal/>;
  - ◆ P2C - Desenvolvido pelo mesmo autor do C2C é um compilador Pascal simples que possibilita a utilização de variáveis de até 16 bits. Apesar de limitado, é também uma boa opção para o desenvolvimento de pequenos projetos.

## 6.5 - Projeto Passo a Passo com PIC

Para demonstrar os procedimentos de programação desde a escrita do programa até programação final do dispositivo, vamos construir um projeto simples utilizando apenas um PIC 16F627 ou 16F628, um resistor e um LED.

O circuito apresentado em seguida deve fazer com que o LED pisque em uma freqüência de aproximadamente 1Hz.

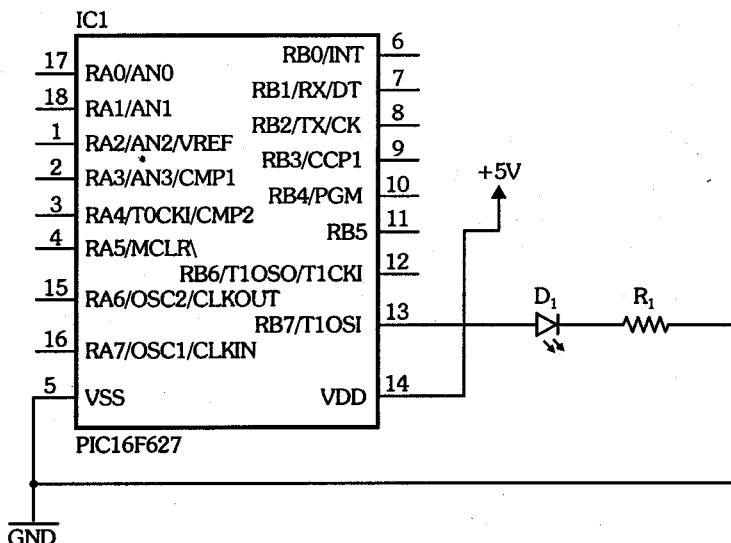


Figura 6.13

```

; Pisca LED
LIST      P=16F627
INCLUDE   <P16F627.INC>
_CONFIG  _CP_OFF  & _WDT_OFF  & _PWRTE_OFF  & _BODEN_OFF  & _MCLRE_OFF  &
_INTRC_OSC_NOCLKOUT & _LVP_OFF

TEMPO    EQU      0x20      ; define a variável TEMPO no endereço 0x20
                                ; da RAM

ORG      0x0000
GOTO    INICIO      ; vetor de RESET
                  ; desvia para o início da programa

ORG      0x0004
BCF      INTCON,T0IF    ; apaga flag de interrupção do timer 0
DECFSZ  TEMPO,F      ; decrementa a variável TEMPO e
RETFIE
MOVLW   D'31'        ; retorna se TEMPO maior que zero
                  ; caso TEMPO igual a zero, copia o valor 31
                  ; decimal
MOVWF   TEMPO        ; para a variável TEMPO
MOVLW   B'10000000'    ; copia o valor 128 decimal para W
XORWF   PORTB,F      ; inverte o BIT 7 do PORTB
RETFIE
INICIO: BANKSEL TRISB    ; retorna da interrupção e liga o GIE
                          ; seleciona o banco de memória correto para
                          ; o TRISB
CLRF    TRISB        ; apaga todos os bits do TRISB (configura o
                      ; PORTB
                  ; como saída)
MOVLW   B'10000111'; configura o OPTION_REG: desliga resistores
                  ; Pull-up
MOVWF   OPTION_REG    ; clock do timer 0 interno, prescaler divide
                  ; por 256
BANKSEL PORTB        ; seleciona o banco de memória correto para
                      ; o PORTB
MOVLW   B'10100000'; configura o INTCON, habilitando o GIE
MOVWF   INTCON        ; e a interrupção do timer 0
MOVLW   D'31'        ; inicializa a variável TEMPO com o
MOVWF   TEMPO        ; valor 31 decimal
FIM:    GOTO     FIM       ; aguarda uma interrupção
END

```

Primeiramente devemos digitar este programa (ou baixá-lo do site da Editora).

Para digitá-lo, devemos inicialmente criar um novo arquivo. Para isso, selecione FILE > NEW no menu principal.

Deve surgir uma janela intitulada CREATE PROJECT, informando que não há projeto aberto e perguntando se deseja abrir um. Responda não.

Em seguida, teremos uma janela em branco intitulada UNTITLED1, na qual deve ser digitado o programa.

```
1; Fisca LED
2; LIST P=16F627
3; XREFS=NO, LIST=NO, ZIF=NO
4; CONFIG _IP_0FF & _INT1_NFY & _PORTA_BFF & _dden_BFF & _MCLE_BFF & _INTB_ESC_INTERRUPT & _LVP_BFF
5;
6; tempo equ $d20 ; Define o variável TEMPO no endereço $d20 da RAM
7; bds db 0000 ; vetor de RESET
8; gdt equ $0000 ; declara para a $0000 a inicializaçao do programa
9; brc equ $0000 ; declara para a $0000 a inicializacão do timer 0
10; bcf _TCON,T0TF ; apaga flag de interrupção do timer 0
11; decf r0, #1 ; decrementa a variável TEMPO
12; movf r0, z0 ; retorna se o tempo é menor que zero
13; movl D:$97, MCLE ; caso TEMPO igual a zero, copia o valor $97 decimal
14; movf MCLE, PORTB ; para o PORTB
15; movl D:$10000000, MCLE ; caso TEMPO maior que zero, copia o valor $10000000 decimal para MCLE
16; zora; PORTB,F ; inserir o bit 7 do PORTB
17; retf ; retorna ao sistema operacional
18;
19; INCICL: BCFSEL TRISB ; seletoriza a função de saída para o TRISB
20; CLRF TRISH ; apaga todos os bits de TRISH (configura o PORTB
21; BCF _OPTION_REG, OPTION_REG ; configura o OPTION_REG: desliga resistores Pull-up
22; BCFSEL _TCON, T0EN ; clock do timer 0 é timer, prescaler dividido por 256
23; BCF _OPTION_REG, OPTION_REG ; configura o OPTION_REG: habilitando o PORTB
24; BCF _OPTION_REG, OPTION_REG ; configura o OPTION_REG: habilitando o SIE
25; BCF _OPTION_REG, OPTION_REG ; e desabilitando a interrupção do timer 0
26; BCF _OPTION_REG, OPTION_REG ; habilitando a interrupção do timer com o
27; BCF _OPTION_REG, OPTION_REG ; valor $97 decimal
28; GOTO INCICL ; aguarda uma interrupção
29;
```

*Figura 6.14*

Após a digitação do programa, selecione FILE > SAVE AS no menu principal, surgirá uma janela para que sejam fornecidos o nome e o local para salvamento do arquivo. Utilizaremos como nome PISCA.ASM, desta forma, na caixa de texto File Name, devemos escrever PISCA.ASM, em seguida clicamos no botão OK. Observe que neste caso não se definiu a pasta para o salvamento, sendo assim o arquivo foi salvo na pasta do MPLAB.

Uma vez salvo o programa, devemos proceder à criação do projeto. Para isso, selecione PROJECT > NEW PROJECT no menu principal. Surge uma janela (NEW PROJECT) solicitando um nome para o projeto. No nosso projeto, utilizamos o nome PISCA.PJT. Após digitar o nome do projeto, clicamos no botão OK.

Feito isso, vamos nos deparar com uma janela de edição de projeto, tal como a janela da figura 6.6. Selecione o processador adequado (PIC 16F627) e também o modo desejado (simulação).

Além disso, nessa janela, devemos ainda adicionar outro arquivo ao projeto (a Microchip chama cada arquivo de um projeto de nó).

Para adicionar o programa (PISCA.ASM) ao projeto, devemos clicar no botão ADD FILES e então selecionar o arquivo PISCA.ASM.

Depois de configurado corretamente o projeto podemos clicar no botão OK. Agora, temos um arquivo de projeto pronto e podemos proceder à compilação do programa.

Pressionando a tecla F10, daremos início ao processo de montagem do programa e se tudo correr bem, a janela BUILD STATUS trará a mensagem: BUILD COMPLETED SUCCESSFULLY.

Provavelmente a janela BUILD STATUS é semelhante à apresentada em seguida.

```
Building PISCA.HEX...
Compiling PISCA.ASM
Command line: "D:\MPLAB\MPASMWIN.EXE /p14F627 /q D:\MPLAB\PISCA.ASM"
Warning[205] D:\MPLAB\PISCA.ASM 2 : Found directive in column 1. (.LIST)
Warning[205] D:\MPLAB\PISCA.ASM 3 : Found directive in column 1. (.INCLUDE)
Warning[205] D:\MPLAB\PISCA.ASM 4 : Found directive in column 1. (.CONFIG)
Warning[205] D:\MPLAB\PISCA.ASM 7 : Found directive in column 1. (.ORG)
Error[113] D:\MPLAB\PISCA.ASM 8 : Symbol not previously defined (INICIO)
Warning[205] D:\MPLAB\PISCA.ASM 9 : Found directive in column 1. (.ORG)
Error[113] D:\MPLAB\PISCA.ASM 10 : Symbol not previously defined (B'10000000)
Warning[209] D:\MPLAB\PISCA.ASM 15 : Missing quote
Error[124] D:\MPLAB\PISCA.ASM 15 : Illegal argument (expected single character)
Error[112] D:\MPLAB\PISCA.ASM 15 : Missing operator
Error[121] D:\MPLAB\PISCA.ASM 15 : Illegal label (INICIO)
Message[302] D:\MPLAB\PISCA.ASM 19 : Register in operand not in bank B. Ensure that bank bits are correct.
Message[302] D:\MPLAB\PISCA.ASM 22 : Register in operand not in bank B. Ensure that bank bits are correct.
Warning[205] D:\MPLAB\PISCA.ASM 29 : Found directive in column 1. (.END)

MPLAB is unable to find output file "PISCA.HEX". This may be due to a compile, assemble, or link process failure.
Build failed.
```

Figura 6.15

A mensagem final BUILD FAILED significa que existem erros de sintaxe no programa e não foi gerado o arquivo resultante (.HEX). Observe que existem, como já dito, três tipos de mensagem: Warnings, Messages e Error.

Os Warnings e Messages não são tão importantes; apenas se referem, na maioria dos casos, a problemas de ordem sintática que não prejudicam o funcionamento do programa.

Vamos observar então a primeira mensagem de erro:

```
Error[113] D:\MPLAB\PISCA.ASM 8 : Symbol not previously defined (INICIO)
```

Esta mensagem indica que há um problema relacionado ao símbolo INICIO e para o montador ainda não foi definido. A localização do erro está indicada pelo número 8 na mensagem, ou seja, o erro foi encontrado na linha número 8.

No entanto, ao analisarmos a oitava linha do programa (GOTO INICIO), não encontramos qualquer erro, pois a etiqueta INICIO está grafada de forma idêntica à etiqueta da linha 18.

Vale então uma dica: devemos analisar sempre o conjunto de erros e não cada erro isoladamente.

Passemos então para a próxima mensagem de erro:

```
Error[113] D:\MPLAB\PISCA.ASM 15 : Symbol not previously defined (B'10000000)
```

Neste caso, a simples análise da linha 15 evidencia o erro: a grafia do operando da instrução MOVLW 'B'10000000' está errada, pois o caractere após o B é um acento agudo, ao contrário do que deveria ser, um apóstrofo.

As mensagens seguintes ela evidenciam este fato:

```
Warning[209] D:\MPLAB\PISCA.ASM 15 : Missing quote
Error[124]  D:\MPLAB\PISCA.ASM 15 : Illegal argument (expected single character)
Error[112]  D:\MPLAB\PISCA.ASM 15 : Missing operator
```

A linha 15 passa então a ser escrita: MOVLW 'B'10000000'.

Finalmente, a última mensagem de erro:

```
Error[121]  D:\MPLAB\PISCA.ASM 18 : Illegal label (INICIO)
```

Faz novamente referência à etiqueta INICIO, indicando que ela é ilegal. Isto vem ao encontro do dito anteriormente: a primeira mensagem de erro, indicando que o símbolo INICIO não foi previamente definido.

Desta forma, para regularizar a situação, passamos a escrever a linha 18 da seguinte forma:

```
INICIO:
BANKSEL TRISB
```

Feitas estas alterações, o programa deve ser montado corretamente e a janela BUILD STATUS deve conter a mensagem: BUILD COMPLETED SUCCESFULLY.

Uma vez o programa montado, será gerado um arquivo binário (com extensão .HEX), que pode ser utilizado para fins de simulação e também para gravação de dispositivos.

Sendo assim, de posse do arquivo binário, podemos passar então à programação do dispositivo com o uso do software IC-PROG.

Após a execução do programa e devidamente configurado, o usuário deve primeiramente carregar o arquivo binário no buffer ativo do software de programação.

Para fazer isso, selecione a opção ARQUIVO (ou FILE) > ABRIR (ou OPEN).

Na janela de abertura de arquivo, selecione a pasta em que foi gravado o projeto do MPLAB e então o arquivo, no caso, PISCA.HEX.

O arquivo binário será então carregado no buffer ativo.

Carregado o arquivo no programador, o software utiliza a configuração definida pela diretiva CONFIG no programa assembly, mas o usuário pode ainda proceder à configuração da palavra de configuração do dispositivo.

## Técnicas de Programação

Estando familiarizados com todo o hardware e software do PIC em questão, podemos agora nos dedicar a um estudo mais avançado sobre diversas técnicas de software e hardware que podem ser utilizadas para melhorar e/ou facilitar os projetos com esses microcontroladores.

De agora em diante, estudaremos as mais diversas técnicas, envolvendo desde os princípios básicos de programação, configuração de pinos de I/O, operações com displays, teclados, rotinas matemáticas, etc.

### 7.1 - Princípios Básicos

Inicialmente veremos alguns princípios e procedimentos básicos na programação de microcontroladores PIC, como: atribuição de valores a registradores, utilização de registradores GPR e terminando com uma visão sobre as formas de comparar valores de registradores.

#### 7.1.1 - Atribuição de Valores

Uma das operações básicas de qualquer dispositivo programável é a operação de atribuição de valores. Com essa operação o programador pode armazenar um dado qualquer em um registrador específico.

Neste tipo de operação são utilizadas duas instruções básicas: MOVLW e MOVWF.

Suponha que se deseje armazenar o valor 100 decimal no registrador PORTA. Isto poderia ser conseguido com a seqüência de instruções apresentada em seguida:

```
MOVLW    D'100'  
MOVWF    PORTA
```

Observe que a operação de atribuição necessita do registrador W como intermediário da operação, no entanto é possível, apesar de nada prático, realizar a mesma operação sem a utilização do registrador W. Observe o programa seguinte que realiza a mesma operação anterior:

CLRF	POR TA
BSF	POR TA, 6
BSF	POR TA, 5
BSF	POR TA, 3

Repare que o efeito final foi o mesmo, ou seja, o registrador PORTA terminou com o valor 0x64 (100 decimal), mas o registrador W não foi alterado.

### 7.1.2 - Utilização dos Registradores GPR

Como já foi visto, o PIC implementa um conjunto de 128 registradores GPR, que constituem na realidade toda a memória RAM disponível para o usuário.

A utilização dos registradores GPR ocorre da mesma forma que os registradores SFR, ou seja, podemos acessar um registrador tanto pelo seu nome definido no MPASM, como, por exemplo, STATUS, como pelo seu endereço na memória RAM, 0x03 no caso do registrador STATUS.

No caso dos registradores GPR, é possível então atribuir nomes a cada um deles, de forma a facilitar a escrita e entendimento do programa.

A operação de atribuição de nomes pode ser feita por várias diretivas do MPASM, como EQU, CBLOCK, etc.

Vejamos uma típica operação de atribuição de nome de registrador:

```
TESTE    EQU 0x20
```

Observe que a diretiva EQU determina ao montador MPASM que doravante, o símbolo TESTE corresponde ao endereço 0x20, que é do primeiro registrador GPR.

Isto significa que a sequência de instruções:

```
MOVLW 0x64  
MOVWF TESTE
```

Irá armazenar o valor 0x64 (100 decimal) no registrador GPR TESTE.

### 7.1.3 - Comparação de Valores

Muitas vezes necessitamos comparar o valor de um registrador com um valor fixo ou de outro registrador e saber se os valores são iguais, ou se não o forem, se o primeiro registrador é maior ou menor que o segundo.

Para isso são utilizadas as operações de comparação. No entanto, o PIC não implementa, ao contrário de outros dispositivos, instruções para comparação de registradores.

Desta forma, devemos utilizar outros artifícios para efetuar a comparação de registradores no PIC.

Neste tópico vamos abordar dois métodos de comparação: o primeiro utilizando a instrução XORWF e o segundo utilizando a instrução SUBWF.

### 7.1.3.1 - XORWF

A comparação utilizando a instrução XORWF baseia-se na propriedade da função lógica XOR de discriminar valores iguais de valores diferentes.

Vejamos uma análise da tabela-verdade da função XOR:

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Pela tabela-verdade, verificamos claramente que quando as entradas são iguais, a saída é '0'; quando as entradas são diferentes, a saída é '1'.

Isto significa que se após uma operação XOR entre dois registradores tivermos o flag Z='1', implicará que o conteúdo dos registradores envolvidos é igual.

Veja um exemplo prático envolvendo um registrador GPR, chamado de TEMP, e o registrador PORTB:

```
TEMP    EQU 0x2A      ; GPR TEMP está no endereço 0x2A  
...  
MOVF    TEMP,W       ; copia o TEMP para W  
XORWF  PORTB,W     ; compara W(TEMP) com o PORTB  
BTFSZC STATUS,Z     ; testa se Z='1' (são iguais)  
...  
;
```

Se após a instrução XORWF tivermos o flag Z em '1', significa que o registrador TEMP e o registrador PORTB possuem o mesmo valor.

### 7.1.3.2 - SUBWF

A instrução SUBWF permite realizar uma comparação muito mais detalhada do que com a XORWF.

A comparação utilizando a instrução de subtração SUBWF pode ser entendida da seguinte forma:

Se após uma subtração o flag Z estiver em nível '1', significa que os valores subtraídos eram iguais.

Se o flag C estiver em '1', significa que o resultado foi maior que zero.

E se o flag C estiver em '0', significa que o resultado foi menor que zero.

Veja a tabela seguinte:

Operação	C	Z
W=F	1	1
W<F	1	0
W>F	0	0

Observe que esta tabela se refere aos registradores W (operando implícito da instrução SUBWF) e F (operando imediato da instrução SUBWF).

Veja o exemplo em seguida:

```
VAR1      EQU 0x20
VAR2      EQU 0x21
...
MOVF      VAR1,W      ;
SUBWF    VAR2,W      ;
...
```

No exemplo efetuou-se a comparação entre a variável VAR2 e a variável VAR1 (armazenada em W). Observe que VAR2 é o F da tabela anterior, enquanto VAR1 equivale ao W da tabela anterior.

Outro detalhe importante sobre a comparação é que normalmente direcionamos o resultado da operação para o registrador W, de forma a não alterar o conteúdo dos registradores comparados.

## 7.2 - Configurando as Portas de I/O

Vejamos então como configurar as portas e ler/escrever dados nelas.

Digamos que vamos configurar inicialmente os pinos da porta B como saída e os da porta A como entrada:

```
BSF      STATUS, RP0      ; muda para o banco de memória 1
MOVLW   0xFF
MOVWF   TRISA            ; configura porta A como entrada
CLRF    TRISB            ; configura a porta B como saída
BCF     STATUS, RP0      ; volta para o banco de memória 0
MOVF    PORTA,W          ; lê o PORTA e escreve em W
MOVWF   PORTB            ; escreve o W no PORTB
```

Não fosse a existência de outros periféricos multiplexados aos pinos das portas A e B, a seqüência de programa descrita anteriormente seria o suficiente para realizar a tarefa de configuração, mas, como já foi visto, existem muitos periféricos internos que compartilham diversos pinos das portas A e B.

Desta forma, uma sequência típica de inicialização das portas deve configurar, além dos registradores TRIS, também os periféricos internos referentes à porta (ou às portas) em questão.

Na porta A, caso não seja desejado utilizar o módulo comparador analógico, devemos configurá-lo para o modo 7, de forma que as quatro entradas (RA0 a RA3) estejam configuradas para I/O digital.

Observe ainda que o modo padrão do módulo de referência interna de tensão é desligado, sendo que o pino RA2 pode ser utilizado para I/O digital.

A utilização dos pinos RA5, RA6 e RA7 vai depender da configuração programada na palavra de configuração do chip.

Já em relação à porta B, devemos ter em mente que caso desejemos utilizar todos os pinos da porta para I/O digital, deveremos:

Desabilitar a transmissão e a recepção da USART, que é feito apagando o bit SPEN no registrador RCSTA.

Caso não seja desejado utilizar o módulo de captura/comparação/PWM, devemos programar o módulo CCP para o modo 0 (por meio dos bits CCP1M0 a CCP1M3, no registrador CCP1CON).

Além disso, caso o oscilador interno do timer 1 não seja utilizado, ele deve ser desabilitado, apagando o bit T1OSCEN do registrador T1CON.

Finalmente, devemos observar que para utilizar o pino RB4/PGM para I/O digital, devemos desabilitar a programação em baixa tensão, o que pode ser feito apagando o bit LVP da palavra de configuração.

Em seguida, temos então um fragmento de programa para a inicialização da porta A como entrada e da porta B como saída.

MOVlw	B'00000111'	; Configura o comparador analógico
MOVwf	CMCON	; para o modo 7 (desligado)
		; o módulo de referência por padrão
		; inicia desligado
BCF	RCSTA, SPEN	; desliga a USART
		; o módulo CCP por padrão inicia no modo
		; desligado
BSF	STATUS, RP0	; muda para o banco de memória 1
MOVlw	0xFF	
MOVwf	TRISA	; configura o port A como entrada
CLRF	TRISB	; configura o port B como saída
BCF	STATUS, RP0	; volta para o banco de memória 0

**Não esqueça que os registradores de configuração das portas (TRISA e TRISB) estão localizados no banco 1 (e 3 no caso do TRISB)!**

Deveremos também fazer uma pequena consideração a respeito do pino RA4: observe que ele possui uma saída em dreno aberto, o que significa que se configurarmos

esse pino como saída, não poderemos manter nível lógico '1' nele sem auxílio externo (o que pode ser feito por um resistor conectado entre o pino RA4 e o V<sub>DD</sub>).

### 7.2.1. Considerações sobre Operações de I/O

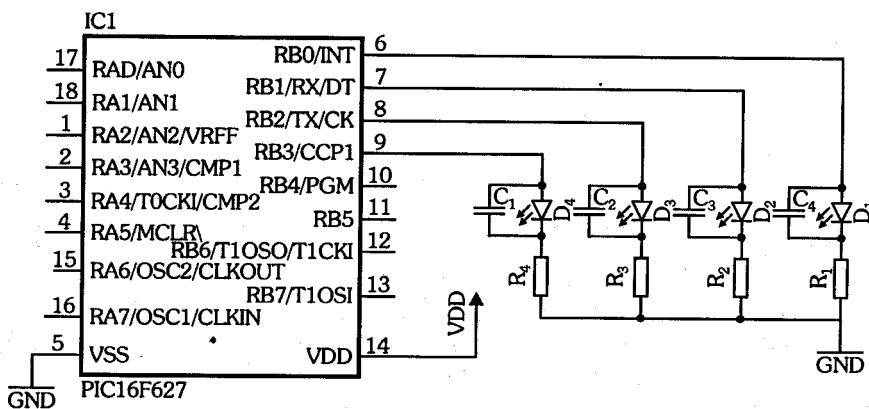
Todas as instruções de modificação de conteúdo de um registrador f (ADDWF, SUBWF, ANDWF, IORWF, XORWF, RLF, RRF, BCF, BSF e COMF) operam em um ciclo chamado de **Read-Modify-Write** (leitura-modificação-escrita) ou simplesmente R-M-W.

Isto significa que uma instrução do tipo ADDWF PORTA,F primeiramente vai fazer a leitura do registrador PORTA, em seguida fará a soma do seu conteúdo com o conteúdo do registrador W e finalmente escreve o resultado de volta no registrador PORTA.

Da mesma forma, uma instrução BCF PORTA,0 primeiramente vai fazer a leitura do registrador PORTA, em seguida zera o bit 0 do conteúdo dele e finalmente escreve o resultado de volta no registrador PORTA.

Tal fato não deveria causar maiores preocupações, mas como o conteúdo dos registradores PORTx é sempre um resultado direto do estado lógico externo do pino, devemos observar melhor o funcionamento de tais operações.

Um dos exemplos clássicos do fabricante aparece na figura seguinte:



**Figura 7.1**

Como podemos observar, conectados aos pinos RB0 até RB3 existem quatro leds e em paralelo com eles temos quatro capacitores.

O problema depende de dois fatores: o valor dos capacitores e a velocidade do processador.

Observe o seguinte programa:

```
CLRF    PORTB      ; apaga o PORTB  
BSF     PORTB,0    ; seta o bit 0 do PORTB  
BSF     PORTB,1    ; seta o bit 1 do PORTB  
BSF     PORTB,2    ; seta o bit 2 do PORTB  
BSF     PORTB,3    ; seta o bit 3 do PORTB  
...
```

Seria de se esperar que após a quarta instrução BSF terímos os quatro leds acesos, mas isto não é verdade. Na prática teremos apenas o LED D4 aceso.

Para entender melhor o porquê deste resultado, devemos executar uma análise detalhada do funcionamento do programa:

1. A primeira instrução (BSF PORTB,0) fará inicialmente uma leitura do conteúdo do registrador PORTB (o que equivale a dizer que se fará a leitura do estado dos pinos da porta B). Em seguida, o bit 0 do valor lido é setado e o resultado escrito novamente no registrador PORTB;
2. Da mesma forma, a segunda instrução (BSF PORTB,1) também fará a leitura do estado da porta B antes de alterar o bit 1 dele. Neste momento é que ocorre o problema: devido ao tempo de carga dos capacitores e à alta velocidade de execução das instruções, pode ocorrer de a tensão de saída do pino RB0 ainda não ter atingido um valor suficiente para ser interpretada como nível lógico '1', fazendo com que o nível lógico lido para este pino seja '0'. Em seguida o bit 1 será setado pela instrução e o resultado B'00000010' será escrito no registrador PORTB. Resultado: o led D1 irá apagar e o led D2 irá acender;
3. O mesmo ocorre para a instrução BSF PORTB,2: o led D2 irá apagar e o led D3 irá acender;
4. Finalmente, após a última instrução BSF, teremos apenas o led D4 aceso.

Observe ainda que este comportamento peculiar não se aplica somente a circuitos como o anterior.

Em velocidades de clock elevadas, seqüências de instruções BCF, BSF, INCF, DECF, etc. sucessivas, tendo como destino uma das portas, podem ter efeitos indesejáveis, como o descrito anteriormente. Isto porque após a escrita numa porta (que ocorre ao final do ciclo de instrução), os drivers de saída demoram um certo tempo para modificar o nível lógico de saída (tempo de condução dos MOSFETs de saída, atrasos, capacitâncias internas e externas, etc.). Desta forma, se após uma escrita numa porta, executarmos uma instrução R-M-W, os dados escritos pela instrução anterior podem não ter tido tempo de se estabilizar na saída da porta, fazendo com que a nova instrução leia valores diferentes do escrito pela instrução anterior.

Para resolver problemas como os descritos, podemos utilizar uma das seguintes soluções:

- ◆ Inserir uma ou mais instruções NOP entre cada instrução R-M-W. Desta forma, a saída da porta terá tempo para se estabilizar. O número de

instruções NOP a ser utilizado será uma função da velocidade de clock utilizada, além de outros fatores internos e externos já comentados. Veja em seguida a utilização de NOPs para correção do programa anterior:

CLRF	PORTE	; apaga o PORTE
NOP		; a inserção da instrução NOP gera um atraso de 4 ciclos de clock, de forma a permitir que o nível externo do port E se estabilize
BSF	PORTE, 0	; seta o bit 0 do PORTE
NOP		; gera atraso de forma que a tensão de saída do pino RB0 se estabilize
BSF	PORTE, 1	; seta o bit 1 do PORTE
NOP		; gera atraso de forma que a tensão de saída do pino RB0 se estabilize
BSF	PORTE, 2	; seta o bit 2 do PORTE
NOP		; gera atraso de forma que a tensão de saída do pino RB0 se estabilize
BSF	PORTE, 3	; seta o bit 3 do PORTE
NOP		; gera atraso de forma que a tensão de saída do pino RB0 se estabilize
...		

- ◆ Utilizar registradores GPR para o armazenamento do estado desejado para a saída da porta. Desta forma, todas as modificações são feitas no registrador GPR e quando necessário, o registrador é copiado para a porta desejada. Em seguida temos uma nova versão do programa inicial utilizando esta filosofia:

; a variável temporária PORTB\_TEMP deve ser criada previamente e  
; associada a um registrador GPR qualquer

CLRF	PORTB_TEMP	; apaga o PORTB_TEMP
BSF	PORTB_TEMP, 0	; seta o bit 0 do PORTB_TEMP
BSF	PORTB_TEMP, 1	; seta o bit 1 do PORTB_TEMP
BSF	PORTB_TEMP, 2	; seta o bit 2 do PORTB_TEMP
BSF	PORTB_TEMP, 3	; seta o bit 3 do PORTB_TEMP
MOVF	PORTB_TEMP,W	; copia o conteúdo do PORTB_TEMP para
MOVWF	PORTB	; o registrador PORTB
...		

### 7.3 - Sub-rotinas

Muitas vezes em um programa, necessitamos repetir diversas vezes uma seqüência de instruções, seja para a geração de um atraso, seja para leitura de um teclado, verificação de uma condição, escrita em display, etc.

Nestes casos, em vez de repetir em cada ocasião todo o código necessário, é mais prático utilizar uma sub-rotina que é uma seqüência de instruções, que pode ser executada a partir de qualquer ponto do programa.

Para execução de uma sub-rotina é utilizada a instrução CALL que, conforme já visto, armazena o valor atual do PC na pilha e em seguida desvia o fluxo do programa para o endereço da sub-rotina.

Após a execução da sub-rotina, utiliza-se uma instrução de retorno (RETURN ou RETLW) para fazer com que o fluxo do programa retorne ao ponto seguinte ao da chamada. Isto é feito desempilhando o endereço armazenado na pilha.

Nos PICs, devemos ter em mente que estamos limitados a um máximo de oito chamadas de sub-rotina consecutivas, conforme já vimos no tópico 3.2.3 no estudo da pilha.

As sub-rotinas são normalmente utilizadas para compartilhar uma porção de código entre diversas partes de um programa, por exemplo: uma sub-rotina de atraso, escrita em display, cálculo, etc. Desta forma, sempre que surge a necessidade de utilizar o código da sub-rotina, basta uma chamada para ela utilizando uma instrução CALL.

Vejamos um exemplo: em seguida, temos dois programas que realizam a mesma função: somar os dois nibbles de um registrador e armazenar o resultado de volta no registrador, mas o primeiro não utiliza sub-rotinas e o segundo utiliza:

```
;programa 1 sem sub-rotina
LIST P=16F627
INCLUDE <P16F627.INC>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
V_A EQU 0X20
V_B EQU 0X21
V_C EQU 0X22
TEMP EQU 0X24
ORG 0X0000
    MOVLW 0X15          ; armazena o valor hexadecimal 0x15
    MOVWF V_A            ; na variável V_A
    MOVLW 0X20          ; armazena o valor hexadecimal 0x20
    MOVWF V_B            ; na variável V_B
    MOVLW D'100'         ; armazena o valor decimal 100 (0x64)
    MOVWF V_C            ; na variável V_C
; agora soma a variável V_C
    ANDLW 0X0F           ; W=0x04
    MOVWF TEMP           ; TEMP=0X04
    SWAPF V_C,W          ; W=0X46
    ANDLW 0X0F           ; W=0X06
    ADDWF TEMP,W          ; W=TEMP+W ou seja W=0x04+0x06=0x0A
    MOVWF V_C             ; V_C=0x0A
; agora soma a variável V_B
    MOVF V_B,W           ; W=V_B ou seja, W=0x20
    ANDLW 0X0F           ; W=0x00
    MOVWF TEMP           ; TEMP=0X00
    SWAPF V_B,W          ; W=0x02
    ANDLW 0X0F           ; W=0x02
    ADDWF TEMP,W          ; W=TEMP+W -> W=0x00+0x02=0x02
    MOVWF V_B             ; V_B=0x02
; agora soma a variável V_A
    MOVF V_A,W           ; W=0x15
    ANDLW 0X0F           ; W=0x05
    MOVWF TEMP           ; TEMP=0x05
    SWAPF V_A,W          ; W=0x51
    ANDLW 0X0F           ; W=0x01
    ADDWF TEMP,W          ; W=TEMP+W -> W=0x05+0x01=0x06
    MOVWF V_A             ; V_A=0x06
END
```

```

;programa 2 utilizando sub-rotina
LIST P=16F627
INCLUDE <P16F627.INC>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
V_A EQU 0X20
V_B EQU 0X21
V_C EQU 0X22
TEMP EQU 0X24
ORG 0X0000
    MOVLW 0X15      ; armazena o valor hexadecimal 0x15
    MOVWF V_A        ; na variável V_A
    MOVLW 0X20      ; armazena o valor hexadecimal 0x20
    MOVWF V_B        ; na variável V_B
    MOVLW D'100'    ; armazena o valor decimal 100 (0x64)
    MOVWF V_C        ; na variável V_C
; calcula a soma de V_C
    CALL SOMA
    MOVWF V_C
    MOVF V_B,W
    CALL SOMA
    MOVWF V_B
    CALL SOMA
    MOVWF V_A
; abaixo está a sub-rotina chamada de SOMA. O valor da variável deverá ser
; passado pelo registrador W. Ao final da sub-rotina, W irá conter a soma.
SOMA:
    MOVWF TEMP
    SWAPF TEMP,W
    ADDWF TEMP,W
    ANDLW 0X0F
    BTFSC STATUS, DC
    IORLW B'00010000'
    RETURN
END

```

A seguir, apresentamos uma técnica que permite realizar chamadas seletivas a sub-rotinas em função do valor de uma variável.

Suponha um teclado com teclas de 0 a 9. A cada tecla apertada deve corresponder uma sub-rotina diferente. Poderíamos utilizar a sub-rotina seguinte para realizar tal tarefa:

```

;Sub-rotina de tratamento de teclas 0 a 9
; o valor da tecla pressionada deve estar inicialmente no registrador W
; a sub-rotina utiliza uma variável chamada TEMP que deve ser declarada
; inicialmente.
TECLAS:
    MOVWF TEMP
    SUBLW 0x09          ; testa se W é maior que 9
    BTFS S, STATUS,C    ; se W é maior que 9, o resultado foi
                        ; negativo e o flag C estará em '0'
                        ; neste caso ocorrerá o retorno da
                        ; sub-rotina
    RETURN
    MOVLW LOW TECLAS_TABELA ; copia o endereço LSB da tabela em W
    ADDWF TEMP,F          ; adiciona ao valor da tecla em TEMP
    MOVLW HIGH TECLAS_TABELA ; copia o endereço MSB da tabela em W
    BTFSC STATUS,C        ; testa se houve transbordo na adição
    ADDLW 1                ; se houve, adiciona 1 ao W
    MOVWF PCLATH           ; copia o endereço MSB da tabela no

```

```

    MOVF    TEMP,W          : PCLATH
    ; copia o índice da tabela em W
TECLAS_TABELA:
    MOVWF   PCL             ; copia o índice em W para o PCL
    GOTO    TECLA_0
    GOTO    TECLA_1
    GOTO    TECLA_2
    GOTO    TECLA_3
    GOTO    TECLA_4
    GOTO    TECLA_5
    GOTO    TECLA_6
    GOTO    TECLA_7
    GOTO    TECLA_8

TECLA_9:
    ...
    RETURN           ; rotina do usuário para a tecla 9

TECLA_0:
    ...
    RETURN           ; rotina do usuário para a tecla 0

TECLA_1:
    ...
    RETURN           ; rotina do usuário para a tecla 1

TECLA_2:
    ...
    RETURN           ; rotina do usuário para a tecla 2

TECLA_3:
    ...
    RETURN           ; rotina do usuário para a tecla 3

TECLA_4:
    ...
    RETURN           ; rotina do usuário para a tecla 4

TECLA_5:
    ...
    RETURN           ; rotina do usuário para a tecla 5

TECLA_6:
    ...
    RETURN           ; rotina do usuário para a tecla 6

TECLA_7:
    ...
    RETURN           ; rotina do usuário para a tecla 7

TECLA_8:
    ...
    RETURN           ; rotina do usuário para a tecla 8

```

## 7.4 - Interrupções

As interrupções são sem dúvida uma poderosa ferramenta dos PICs. Antes de utilizá-las, devemos observar alguns detalhes importantes sobre elas.

### 7.4.1 - Desativação do GIE

Em algumas ocasiões, pode ser necessário desabilitar as interrupções (como, por exemplo, em rotinas sensíveis a atrasos de tempo). Conforme já estudado, uma instrução do tipo BCF INTCON,GIE, a princípio, é suficiente para desabilitar o bit GIE e consequentemente as interrupções globais.

No entanto, existe uma condição na qual mesmo após uma instrução BCF INTCON,GIE, ainda assim o bit GIE estará habilitado. Vejamos o seguinte fragmento de programa:

```
BCF      INTCON, GIE      ; DESLIGA O GIE
MOVlw    0x1A              ; COPIA O VALOR 0x1A PARA W
BSF      STATUS, C        ; APAGA O FLAG C
```

Considere que ocorra um evento de interrupção durante a execução da primeira instrução. Devido à latência das interrupções nos PICs, a instrução BCF será executada corretamente e desabilita as interrupções no PIC.

O programa prosseguirá executando a instrução seguinte (MOVLW) e então ocorrerá o desvio para endereço 0x0004 para o tratamento da interrupção que havia acontecido durante a execução do BCF INTCON,GIE.

É aí que ocorre o problema: ao término do tratamento de interrupção, utiliza-se normalmente uma instrução RETFIE para o retorno ao programa, no entanto essa instrução também faz com que o bit GIE seja setado, habilitando novamente as interrupções.

Resultado final: as interrupções continuam habilitadas, mesmo após um comando explícito para desabilitação do GIE.

Para evitarmos este tipo de inconveniente, podemos utilizar um dos seguintes métodos (baseados na nota de aplicação AN576 da Microchip):

## Método 1

O primeiro método para garantir a desabilitação das interrupções é mostrado em seguida. Ele utiliza um LOOP que somente permite que o programa continue se o bit GIE estiver desabilitado:

```
...
LOOP:
BCF      INTCON, GIE      ; desliga o GIE
NOP
BTFS C  INTCON, GIE      ; aguarda 1 ciclo
GOTO    LOOP              ; testa se o GIE está desligado e
                          ; volta para LOOP caso esteja ligado
...
...
BSF      INTCON, GIE      ; caso esteja desligado, o programa
                          ; continua, executando a rotina desejada
...
                          ;
...
BSF      INTCON, GIE      ; e em seguida habilita novamente o GIE
...
```

## Método 2

O segundo método, demonstrado em seguida, utiliza um registrador GPR para funcionar como um "espelho" do registrador INTCON. Desta forma, sempre que desejarmos habilitar/desabilitar as interrupções no PIC, devemos setar/apagar, além do bit GIE, também o "bit espelho", o qual chamaremos de GIE2:

```
... FLAG EQU 0x20 ; o endereço do registrador deverá ser alterado
DEFINE GIE2 FLAG,7 ; de acordo com o programa do usuário
; define que o bit 7 da variável FLAG será
; o GIE2

ORG 0x0004
; rotina de tratamento de interrupções
...
...
BTFSC GIE2 ; testa se o GIE2 está habilitado
RETFIE ; se estiver, então retorna ativando o GIE
RETURN ; se não, utiliza retorno simples (GIE=0)

; programa principal
INICIO:
...
...
BCF GIE2 ; desliga o GIE2 e
BCF INTCON,GIE ; também o GIE
...
...
BSF GIE2 ; liga o GIE2 e
BSF INTCON,GIE ; também o GIE
...
...
END
```

## 7.4.2 - Tratamento de Interrupções

Uma vez que ocorra um evento de interrupção e caso o bit GIE e a habilitação individual da interrupção estejam ativados, teremos o desvio do programa para o vetor de interrupção 0x0004.

Vejamos agora alguns detalhes importantes sobre os procedimentos de tratamento de interrupções:

### 1. Salvamento do contexto atual

O salvamento do contexto é um procedimento que objetiva preservar o conteúdo de registradores que estejam sendo utilizados no programa principal e que possam ser alterados pela sub-rotina de tratamento de interrupção. Normalmente, os registradores mais importantes a serem salvos durante uma RTI são o W e o STATUS.

### 2. Verificação do tipo de interrupção ocorrido

O segundo passo a ser observado em uma RTI é a verificação da interrupção ocorrida.

Normalmente, utilizamos uma seqüência de testes de BIT (instruções BTFSCs) para verificar cada flag de interrupção individualmente, desviando para o tratamento individual caso ele esteja ativado (nível lógico '1').

Uma observação muito interessante sobre este procedimento é que a ordem que utilizamos para verificar os flags de interrupção, irá da mesma definir a prioridade do tratamento da interrupção.

### 3. Tratamento da interrupção

É no tratamento da interrupção que o software realmente toma conhecimento da interrupção e realiza tarefas específicas associadas com o evento de interrupção em particular.

Lembre-se de que é no tratamento de interrupção que o usuário deve providenciar o apagamento do flag de interrupção relativo ao processo em andamento.

### 4. Recuperação do contexto anterior

A recuperação do contexto anterior é uma operação inversa à de salvamento do contexto.

Na recuperação do contexto, o conteúdo dos registradores previamente salvos pelo salvamento de contexto é devolvido aos seus registradores de origem, fazendo com que a CPU retorne ao estado em que se encontrava no momento em que ocorreu a interrupção.

Em seguida temos um exemplo típico de uma rotina de tratamento de interrupção (RTI) que implementa os passos descritos anteriormente.

```
; Exemplo do tratamento de interrupções
; No exemplo seguinte, o programa irá verificar entre três interrupções
; individuais: T0IF, T1IF e INTF
W_TEMP EQU 0x20
STATUS_TEMP EQU 0x21
...
ORG 0x0004
MOVWF W_TEMP ; salva o conteúdo do W em W_TEMP
SWAPF STATUS,W ; salva o conteúdo do STATUS ...
MOVWF STATUS_TEMP ; ... em STATUS_TEMP
BTFS C INTCON,T0IF ; testa se ocorreu uma interrupção do timer 0
GOTO TRATA_TMR0 ; se ocorreu, desvia para o tratamento da mesma
BTFS C PIR1,TMR1IF ; testa se ocorreu uma interrupção do timer 1
GOTO TRATA_TMR1 ; se ocorreu, desvia para o tratamento da mesma
BTFS C INTCON,INTF ; testa se ocorreu uma interrupção externa INT
GOTO TRATA_INT ; se ocorreu, desvia para o tratamento da mesma
FIM_INT:
    SWAPF STATUS_TEMP,W ; restaura o conteúdo do ...
    MOVWF STATUS ; ... registrador STATUS
    SWAPF W_TEMP,F ; restaura o conteúdo ...
    SWAPF W_TEMP,W ; ... do registrador W
    RETFIE ; retorna da interrupção
TRATA_TMR0:
    BCF INTCON,T0IF ;
    ...
    GOTO FIM_INT
TRATA_TMR1:
    BCF PIR1,T1IF ;
    ...
    GOTO FIM_INT
```

```
TRATA_INT:  
    BCF      INTCON, INTF      ;  
    ...  
    ...  
    GOTO    FIM_INT          ;  
    ...  
    ...
```

Para facilitar a vida dos leitores, incluímos no arquivo MACROS.ASM duas macros para salvamento e recuperação do contexto (W e STATUS).

As macros são chamadas de: SALVA\_CONTEXTO e RESTAURA\_CONTEXTO.

```
SALVA_CONTEXTO MACRO  
    MOVWF  W_TEMP           ; salva o conteúdo do W em W_TEMP  
    SWAPF   STATUS,W         ; salva o conteúdo do STATUS ...  
    MOVWF  STATUS_TEMP       ; ... em STATUS_TEMP  
ENDM  
  
RESTAURA_CONTEXTO MACRO  
    SWAPP  STATUS_TEMP,W    ; restaura o conteúdo do ...  
    MOVWF  STATUS             ; ... registrador STATUS  
    SWAPP  W_TEMP,F          ; restaura o conteúdo ...  
    SWAPP  W_TEMP,W          ; ... do registrador W  
ENDM
```

### 7.4.3 - Sub-rotinas em Interrupções

Em rotinas de tratamento de interrupções muito complexas, pode acontecer de o programador necessitar realizar chamada(s) a sub-rotina(s), mas esta prática não é recomendada. Para verificar os problemas envolvidos com a utilização de sub-rotinas dentro de interrupções, vejamos o exemplo hipotético em seguida:

```
ORG 0x0000  
GOTO INICIO  
ORG 0x0004          ; rotina de tratamento de interrupção  
...  
CALL SUB-ROTINA1    ; chamada para sub-rotina 1 de dentro da RTI  
...  
RETFFIE  
SUB-ROTINA1:        ; sub-rotina 1  
...  
...  
...  
RETURN  
INICIO:  
...  
CALL SUB-ROTINA1    ; chamada para sub-rotina 1 de dentro do programa  
...  
; principal  
...  
END
```

Observe que o programa anterior possui uma rotina de tratamento de interrupção (RTI) a qual realiza uma chamada para uma sub-rotina SUB-ROTINA1, que também é chamada pelo programa principal.

Os problemas surgem quando:

1. Uma interrupção acontece no instante em que o CPU está executando a sub-rotina 1, chamada pelo programa principal. Neste caso, a execução da sub-rotina será suspensa e o programa será desviado para o endereço 0x0004 para o tratamento da interrupção. Dentro da RTI, ocorre uma chamada para a sub-rotina SUB-ROTINA1. Resultado: os dados envolvidos na execução anterior da SUB-ROTINA1 serão perdidos;
2. Durante o tratamento de interrupção, que inclui a chamada a SUB-ROTINA 1, acontece um novo evento de interrupção. Neste caso, como o bit GIE ainda está em nível lógico '0' (pois a CPU encontra-se tratando a primeira interrupção), haverá um atraso até que RTI relativa ao primeiro evento de interrupção seja concluída, após o que, terá início o tratamento da segunda interrupção. Observe que em aplicações críticas, pode ocorrer a perda de dados, ou mesmo a completa inviabilização do programa.

Lembre-se ainda de que a capacidade da pilha interna do PIC é muito limitada e o evento de interrupção irá utilizar uma posição da pilha, assim como a chamada à sub-rotina. Desta forma, a prática de chamada de sub-rotina a partir de um RTI deve ser desencorajada, pois pode conduzir o sistema a um estouro de pilha.

## 7.5 - Matemática no PIC

### 7.5.1 - Adição

Vejamos alguns exemplos de adições simples utilizando a instrução ADDWF e ADDLW:

			C	DC	Z
MOVlw	D'10'	; W=D'10'	X	X	X
ADDlw	D'5'	; W=D'10'+D'5'=D'15'	0	0	0

Observe que o programa anterior faz com que a CPU execute a soma de 10 decimal com 5 decimal, o que resulta em 15 decimal.

Antes da execução da instrução ADDLW, o estado dos flags Z, DC e C era desconhecido, no entanto, após a execução da instrução, os flags foram modificados da seguinte forma:

O flag C foi apagado, pois o resultado não causou o transbordo do registrador de destino (o resultado foi menor que 256).

O flag DC também foi apagado, pois não houve transbordo do bit 3 para o bit 4 do registrador de destino.

Finalmente, o flag Z foi também apagado, pois o resultado foi diferente de zero.

Vejamos agora uma adição que cause o transbordo do registrador de destino:

		$C \quad DC \quad Z$ X    X    X 1    1    0
MOVlw	D'250' ; W=D'250'	
ADDlw	D'7' ; W=D'250'+D'7'=D'1'	

Observe que na soma anterior (250 decimal + 7 decimal) o resultado (257 decimal) é um número de 9 bits.

Neste caso, como o registrador de destino possui apenas 8 bits, somente os 8 bits LSB serão armazenados no destino.

Para determinar o valor decimal armazenado no destino, devemos então subtrair 256 do resultado da soma. Assim,  $257 - 256 = 1$ .

Observe na tabela seguinte o comportamento da CPU durante a soma anterior:

D'250'	Resultado Parcial			C
	Decimal	Hexa	Binário	
início	250	FA	11111010	0
+1	251	FB	011111011	0
+1	252	FC	011111100	0
+1	253	FD	0111111101	0
+1	254	FE	0111111110	0
+1	255	FF	0111111111	0
+1	256	100	100000000	1
+1	257	101	100000001	1

Veja que ao final da operação de adição os oito bits inferiores do registrador de destino foram iguais a 00000001 binário, ou 1 decimal. Observe também que o nono bit terminou setado, o que indica um transbordo do registrador.

Note que o nono bit não está implementado realmente, sendo aquele bit registrado no flag C do registrador STATUS.

Vejamos agora uma outra operação de soma:

VA EQU 0x20.

		$C \quad DC \quad Z$ X    X    X X    X    X X    X    X 1    1    0
MOVlw	D'250' ; W=D'250'	
MOVwf	VA ; VA=D'250'	
MOVlw	D'10' ; W=D'10'	
ADDwf	VA, F ; VA=VA+W = D'250'+D'10'=D'4'	

D'250'	Resultado Parcial			C
	Decimal	Hexa	Binário	
início	250	FA	11111010	0
+1	251	FB	11111011	0
+1	252	FC	11111100	0

D'250'	Resultado Parcial			C
	Decimal	Hexa	Binário	
+1	253	FD	11111101	0
+1	254	FE	11111110	0
+1	255	FF	11111111	0
+1	256	100	100000000	1
+1	257	101	100000001	1
+1	258	102	100000010	1
+1	259	103	100000011	1
+1	260	104	100000100	1

Neste exemplo, a soma 250+10 resulta em um valor de 9 bits (260 decimal). Os oito bits menos significativos são armazenados no registrador de destino. O nono bit (=1) será armazenado no carry (C), indicando o transbordo do registrador.

Para sabermos o resultado decimal armazenado no registrador, simplesmente subtraímos 256 e então:  $260-256=4$  ! Veja que os oito bits inferiores (00000100 binário) eqüivalem a 4 decimal.

Lembre-se que uma adição **sempre** resulta em um número de 9 bits, sendo que os 8 bits menos significativos são escritos no destino e o nono bit é escrito no flag C.

### 7.5.1.1 - Adição de 16 bits

Em seguida temos um exemplo de uma sub-rotina para a adição de dois números hexadecimais de 16 bits. Lembre-se que para efetuar corretamente uma adição de dezesseis bits, devemos, inicialmente, adicionar os LSB, verificar se houve um transbordo na soma e caso tenha ocorrido, adicionar 1 ao MSB, em seguida somamos os MSB.

```

; ADD16 - Soma dois valores de 16 bits (A=A+B)
; Entrada
;     A0,A1,B0,B1 - (0-LSB, 1-MSB)
; Saída
;     A0=A0+B0 e A1=A1+B1
; As variáveis A0,A1,B0 e B1 precisam ser definidas previamente
ADD16:
    MOVF    B0,W      ; adiciona os 8 bits
    ADDWF   A0,F      ; menos significativos
    BTFSC   STATUS,C  ; testa se houve transbordo
    GOTO    ADD16_2    ; se houve pula para ADD16_2
    MOVF    B1,W      ; se não, adiciona os
    ADDWF   A1,F      ; 8 bits mais significativos e
    RETURN
ADD16_2:
    MOVF    B1,W      ; adiciona os 8 bits
    ADDWF   A1,F      ; mais significativos e
    MOVLW   0x01      ; soma o transbordo dos 8 bits LSB
    ADDWF   A1,F      ; ao dígito MSB
    RETURN

```

**Exemplo**

```

LIST P=16F627
INCLUDE <P16F627.INC>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
A0 EQU 0x20
A1 EQU 0x21
B0 EQU 0x22
B1 EQU 0x23
ORG 0x0000
GOTO INICIO

ADD16:
    MOVE B0,W ; adiciona os 8 bits
    ADDWF A0,F ; menos significativos
    BTFS C STATUS ; testa se houve transbordo
    GOTO ADD16_2 ; se houve pula para ADD16_2
    MOVE B1,W ; se não, adiciona os
    ADDWF A1,F ; 8 bits mais significativos e
    RETURN ; retorna

ADD16_2:
    MOVE B1,W ; adiciona os 8 bits
    ADDWF A1,F ; mais significativos e
    MOVLW 0x01 ; soma o transbordo dos 8 bits LSB
    ADDWF A1,F ; ao dígito MSB
    RETURN ; retorna

INICIO:
    MOVLW 0xFF ;
    MOVWF A0 ; variável A0=0xFF
    MOVLW 0x01 ;
    MOVWF A1 ; variável A1=0x01 (A=0x01FF)
    MOVLW 0x02 ;
    MOVWF B0 ; variável B0=0x02
    MOVLW 0x00 ;
    MOVWF B1 ; variável B1=0x00 (B=0x0002)
    CALL ADD16 ; soma A+B (resultados: A1=0x02 e A0=0x01 => A=0x0201)
    SLEEP ; o programa termina aqui
END

```

**7.5.2 - Subtração**

Vejamos alguns exemplos de subtrações simples utilizando a instrução SUBWF e SUBLW:

		C	DC	Z
MOVLW D'2'	; W=D'2'	X	X	X
SUBLW D'1'	; W=D'1'-D'2'=D'255'	0	0	0

Veja que nesta subtração (1 - 2), o resultado (-1) é representado no registrador de destino como 255 decimal.

D'1'	Resultado Parcial			C
	Decimal	Hexa	Binário	
início	1	101	100000001	1
-1	0	100	100000000	1
-1	-1	OFF	011111111	0

Como podemos observar, em operações de subtração, a primeira providência tomada pela CPU é inicialmente setar o flag de Carry. Isto é feito de forma a garantir que a CPU terá uma "reserva" a ser utilizada no caso de empréstimo.

Desta forma, a operação  $1 - 2 = -1$  resulta efetivamente no valor 255 decimal, no PIC.

Veja que devido à necessidade do empréstimo, o flag C terminou a operação em nível lógico '0'.

Da mesma forma o flag DC, indicando que houve um empréstimo do bit 5 para o bit 4 do registrador de destino.

Finalmente, o flag Z em '0' indica que o resultado foi diferente de zero.

Vejamos um outro exemplo:

			C	DC	Z
MOVLW	D'10'	; W=D'10'	X	X	X
MOVWF	VA	; VA=D'10'	X	X	X
SUBWF	VA	; VA=D'10'-D'10'=0	1	1	1

Neste exemplo, podemos observar que foi realizada a operação 10-10 que resulta em 0. Observe o estado dos flags após a subtração:

C='1', indicando que não foi necessário realizar um empréstimo do carry;

DC='1', indicando que não houve empréstimo do bit 4 para o bit 3;

Z='1', indicando que o resultado (armazenado em VA) é igual a zero.

			C	DC	Z
MOVLW	D'10'	; W=D'10'	X	X	X
MOVWF	VA	; VA=D'10'	X	X	X
MOVLW	D'15'	; W=D'15'	X	X	X
SUBWF	VA	; VA=D'10'-D'15'=D'251'	1	1	1

Observe agora o exemplo anterior: foi realizada a subtração de  $10 - 15 = -5$ . O valor efetivamente armazenado no registrador de destino (W) não será D'5', e sim D'251'. Isto porque o resultado -5 somado a 256 é igual a D'251'.

Para compreender melhor a operação, veja a demonstração em seguida:

D'1'	Resultado Parcial			C
	Decimal	Hexa	Binário	
início	10	10A	100001010	1
-1	9	109	100001001	1
-1	8	108	100001000	1
-1	7	107	100000111	1
-1	6	106	100000110	1
-1	5	105	100000101	1

D'1'	Resultado Parcial			C
	Decimal	Hexa	Binário	
-1	4	104	100000100	1
-1	3	103	100000011	1
-1	2	102	100000010	1
-1	1	101	100000001	1
-1	0	100	100000000	1
-1	-1	OFF	011111111	0
-1	-2	0FE	011111110	0
-1	-3	0FD	011111101	0
-1	-4	0FC	011111100	0
-1	-5	0FB	011111011	0

Lembre-se que uma subtração **sempre** resulta em um valor de 9 bits, sendo os 8 bits menos significativos escritos no destino e o nono bit escrito no flag C.

### 7.5.2.1 - Subtração de 16 bits

Muitas vezes pode ser necessária a realização de uma operação de subtração envolvendo valores de 16 bits.

Neste caso, podemos utilizar a sub-rotina apresentada em seguida:

```

; SUB16 - Subtrai dois valores de 16 bits (A=A-B)
; Entrada
;   A0,A1 - Minuendo, B0,B1 - Subtraendo (0-LSB e 1-MSB)
; Saída
;   A0,A1 - Resultado (A0-LSB, A1-MSB)
; As variáveis A0,A1,B0,B1 devem ser definidas previamente no programa

SUB16:
    MOVF    B0,W          ; subtrai os 8 bits
    SUBWF  A0,F          ; menos significativos
    BTFSS  STATUS,C      ; testa se houve empréstimo
    GOTO   SUB2           ; se houve pula para SUB2
    MOVF    B1,W          ; se não, subtrai os
    SUBWF  A1,F          ; 8 bits mais significativos e
    RETURN                         ; retorna

SUB2:
    MOVF    B1,W          ; subtrai os 8 bits
    SUBWF  A1,F          ; mais significativos e
    MOVLW  0X01           ; subtrai 1 de A1 por conta
    SUBWF  A1,F          ; do empréstimo na subtração do LSB
    RETURN                         ; retorna

```

### Exemplo

```

LIST P=16F627
INCLUDE <P16F627.INC>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
A0    EQU 0x20

```

```

A1 EQU 0x21
B0 EQU 0x22
B1 EQU 0x23
ORG 0x0000
GOTO INICIO

SUB16:
    MOVF B0,W ; subtrai os 8 bits
    SUBWF A0,F ; menos significativos
    BTFSS STATUS,C ; testa se houve empréstimo
    GOTO SUB2 ; se houve pula para SUB2
    MOVF B1,W ; se não, subtrai os
    SUBWF A1,F ; 8 bits mais significativos e
    RETURN ; retorna

SUB2:
    MOVF B1,W ; subtrai os 8 bits
    SUBWF A1,F ; mais significativos e
    MOVLW 0x01 ; subtrai 1 de A1 por conta
    SUBWF A1,F ; do empréstimo na subtração do LSB
    RETURN ; retorna

INICIO:
    MOVLW 0x00 ;
    MOVWF A0 ; variável A0=0x00
    MOVLW 0x10 ;
    MOVWF A1 ; variável A1=0x10 (A=0x1000)
    MOVLW 0xFF ;
    MOVWF B0 ; variável B0=0xFF
    MOVLW 0x01 ;
    MOVWF B1 ; variável B1=0x01 (B=0x01FF)
    CALL SUB16 ; subtrai A-B (resultado= A1=0x0E e A0=0x01 => A=0x0E01)
    SLEEP ; o programa termina aqui
END

```

Em seguida temos um outro exemplo de sub-rotina de subtração de 16 bits. A diferença desta sub-rotina para a anterior é que esta gera corretamente o estado dos flags Z e C de acordo com o resultado da operação:

Se a operação resultar em 0x0000, o flag Z será setado ('1');

Se a operação resultar em um número menor que zero (acontecer um empréstimo), o flag C estará apagado ('0');

E finalmente, se a operação resultar em um número maior que zero, ou seja, não foi necessário um empréstimo para completar a operação, o flag C estará setado ('1').

```

; SUB16B - Subtrai dois valores de 16 bits (A=A-B) flags OK
; Entrada
;     A0,A1 - Subtraendo, B0,B1 - Minuendo (0-LSB, 1-MSB)
; Saída
;     A0,A1 - resultado da subtração
;
; Os flags C e Z são alterados ao final da operação de acordo com o resultado:
;
; C='1' - não houve empréstimo (resultado positivo)
; C='0' - houve empréstimo (resultado negativo)
; Z='1' - resultado igual a zero
; Z='0' - resultado diferente de zero
;
SUB16B:
    MOVF B0,W

```

```

SUBWF    A0, F
MOVlw    0x01
BTFS S   STATUS, C
SUBWF    A1, F
BTFS S   STATUS, C
GOTO    EMPRESTA
MOVF    B1, W
SUBWF    A1, F
FIM_SUB16B:
MOVF    A0, F
BTFS C   STATUS, Z
MOVF    A1, F
RETURN
EMPRESTA:
MOVF    B1, W
SUBWF    A1, F
BCF     STATUS, C
GOTO    FIM_SUB16B

```

## 7.5.3 - Multiplicação

Além da adição e da subtração, podemos também executar outras operações matemáticas no PIC.

### 7.5.3.1 - Multiplicação Inteira com RLF

Vejamos uma forma simples de multiplicar um valor de um registrador por um valor inteiro múltiplo de uma potência de 2:

```

; MACRO para Multiplicação com RLF. Incluída no arquivo MACROS.ASM
; Utilização: MULT2 <registrador>,<valor múltiplo de potência de 2>
MULT2 MACRO    REG,VAL
    BCF     STATUS,C           ; limpa o flag de carry
    IF (VAL==2)
        RLF     REG,F          ; rotaciona o registrador à esquerda
                                ; o que equivale a multiplicar por 2
    ENDIF
    IF (VAL==4)
        RLF     REG,F          ; multiplica por 2
        BCF     STATUS,C          ; zera carry para não atrapalhar a pró
                                ; xima multiplicação
        RLF     REG,F          ; multiplica novamente por 2 (total 4)
    ENDIF
    IF (VAL==8)
        RLF     REG,F          ; multiplica por 2
        BCF     STATUS,C          ; zera carry
        RLF     REG,F          ; multiplica novamente por 2 (total 4)
        BCF     STATUS,C          ; zera carry
        RLF     REG,F          ; multiplica novamente por 2 (total 8)
    ENDIF
    IF (VAL==16)
        RLF     REG,F          ; multiplica por 2
        BCF     STATUS,C          ; zera carry
        RLF     REG,F          ; multiplica novamente por 2 (total 4)
        BCF     STATUS,C          ; zera carry
        RLF     REG,F          ; multiplica novamente por 2 (total 8)
        BCF     STATUS,C          ; zera carry
    ENDIF

```

```

        RLF      REG, F      ; multiplica novamente por 2 (total 16)
ENDIF
IF (VAL==32)
        RLF      REG, F      ; multiplica por 2
        BCF      STATUS, C   ; zera carry
        RLF      REG, F      ; multiplica novamente por 2 (total 4)
        BCF      STATUS, C   ; zera carry
        RLF      REG, F      ; multiplica novamente por 2 (total 8)
        BCF      STATUS, C   ; zera carry
        RLF      REG, F      ; multiplica novamente por 2 (total 16)
        BCF      STATUS, C   ; zera carry
        RLF      REG, F      ; multiplica novamente por 2 (total 32)
ENDIF
ENDM

```

### **Exemplo**

```

LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
A0 EQU 0x20
ORG 0x0000
MOVLW D'25'
MOVWF A0          ; A0=D'25'
MULT2 A0,4        ; multiplica A0 por 4
MOVF  A0,W        ; W=D'100'
SLEEP
END

```

### **7.5.3.2 - Multiplicação Didática 8x8 bits**

A seguir, temos uma sub-rotina de multiplicação que utiliza o método de somas consecutivas para calcular o resultado:

```

; Multiplica dois números (A0 e B0) e armazena o resultado em C0 E C1.
; é necessário definir as variáveis A0, B0 , C0 e C1
MULTB8X8:
    CLRF    C0          ; limpa variável C0
    CLRF    C1          ; limpa variável C1
    MOVF    A0,W        ; copia o valor de A0 ...
    MOVWF   C0          ; para a variável C0
LOOP_MULTB:
    DECF    B0,F        ; decrementa B0
    BTFSC  STATUS,Z    ; se B0=0 ...
    RETURN
    MOVF    A0,W        ; copia A0 para o W
    ADDWF  C0,F        ; soma o A0 (em W) com C0 e guarda o resultado em C0
    BTFSC  STATUS,C    ; se houver transbordo do C0 ...
    INCF    C1,F        ; incrementa o C1
    GOTO    LOOP_MULTB ; retorna ao LOOP_MULTB para mais um ciclo

```

**Exemplo**

```

LIST P=16F627
INCLUDE <P16F627.INC>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
A0 EQU 0x20
B0 EQU 0x21
C0 EQU 0x22
C1 EQU 0x23
ORG 0x0000
GOTO INICIO
MULTB8X8:
    CLRF C0 ; limpa variável C0
    CLRF C1 ; limpa variável C1
    MOVF A0,W ; copia o valor de A0 ...
    MOVWF C0 ; para a variável C0
LOOP_MULTB:
    DECF B0,F ; decrementa B0
    BTFSC STATUS,Z ; se B0=0 ...
    RETURN ; retorna da sub-rotina
    MOVF A0,W ; copia A0 para o W
    ADDWF C0,F ; soma o A0 (em W) com C0 e guarda o resultado em C0
    BTFSC STATUS,C ; se houver transbordo do C0 ...
    INCF C1,F ; incrementa o C1
    GOTO LOOP_MULTB ; retorna ao LOOP_MULTB para mais um ciclo
INICIO:
    MOVLW 0x0A ; inicializa a variável A0
    MOVWF A0 ; com o valor 0x0A (10 decimal)
    MOVLW 0x03 ; inicializa a variável B0
    MOVWF B0 ; com o valor 0x03
    CALL MULTB8X8 ; chama sub-rotina de multiplicação
    SLEEP ; o programa termina aqui
; resultado: C0=0x1E (30 decimal) e C1=0. Z='1', C='0' e DC='0'
END

```

### 7.5.3.3 - Multiplicação Inteira 8x8 bits

Outro método muito mais eficiente e prático para efetuar uma operação de multiplicação pode ser visto na listagem seguinte:

```

: MULTIPLICA8X8 - Multiplica dois números de 8 bits
; Entrada
;     A0,B0 - Fatores da multiplicação
; Saída
;     B0,B1 - Produto da multiplicação (B0 - LSB, B1 - MSB)
; É necessário definir previamente as variáveis A0, B0 e B1
MULT8 MACRO BIT
    BTFSC A0, BIT
    ADDWF B1,F
    RRF B1,F
    RRF B0,F
ENDM
MULTIPLICA8X8:
    MOVF B0, W
    CLRF B0
    CLRF B1
    BCF STATUS,C

```

```
MULT8    0
MULT8    1
MULT8    2
MULT8    3
MULT8    4
MULT8    5
MULT8    6
MULT8    7
RETURN
```

**Exemplo:**

```
LIST P=16F627
INCLUDE <P16F627.INC>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
A0    EQU 0x20
B0    EQU 0x21
B1    EQU 0x22
ORG 0x0000
GOTO INICIO
MULT8 MACRO BIT
    BTFSC A0, BIT
    ADDWF B1,F
    RRF   B1,F
    RRF   B0,F
ENDM
MULTIPLICA8X8:
    MOVF B0, W
    CLRF B0
    CLRF B1
    BCF  STATUS,C
    MULT8 0
    MULT8 1
    MULT8 2
    MULT8 3
    MULT8 4
    MULT8 5
    MULT8 6
    MULT8 7
    RETURN
INICIO:
    MOVLW D'9'
    MOVWF A0          ; A0=90 decimal
    MOVLW D'5'
    MOVWF B0          ; B0=5 decimal
    CALL  MULTIPLICA8X8 ; chamada da sub-rotina de multiplicação
    SLEEP            ; fim do programa
END
; o resultado final do programa será B0=0xC2 e B1=0x01 (B=0x01C2 ou
; 450 decimal) Z='0', C='0' e DC='0'
```

### 7.5.3.4 - Multiplicação Inteira 16x16 bits

A seguir, podemos verificar uma rotina para efetuar a multiplicação de dois valores de 16 bits (o que resulta em um valor de 32 bits):

; Multiplica dois números de 16 bits (A1:A0 e B1:B0) e armazena o resultado em C3:C2:C1:C0, sendo C0 o LSB. É necessário definir as variáveis A0,A1,B0,B1,C0,C1,C2 E C3.

MULTIPLICA16X16:

```
    CLRF    C3  
    CLRF    C2  
    CLRF    C1  
    CLRF    C0  
    BSF     C1, 7
```

MULTL1:

```
    RRF     A1, F  
    RRF     A0, F  
    BTFSS  STATUS, C  
    GOTO   MULTL2  
    MOVF   B0, W  
    ADDWF  C2, F  
    MOVF   B1, W  
    BTFSC  STATUS, C  
    INCFSZ B1, W  
    ADDWF  C3, F
```

MULTL2:

```
    RRF     C3, F  
    RRF     C2, F  
    RRF     C1, F  
    RRF     C0, F  
    BTFSS  STATUS, C  
    GOTO   MULTL1  
    RETURN
```

### ***Exemplo***

```
LIST P=16F627  
INCLUDE <P16F627.INC>  
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &  
_INTRC_OSC_NOCLKOUT & _LVP_OFF  
A0    EQU 0x20  
A1    EQU 0x21  
B0    EQU 0x22  
B1    EQU 0x23  
C0    EQU 0x24  
C1    EQU 0x25  
C2    EQU 0x26  
C3    EQU 0x27  
ORG   0x0000  
GOTO  INICIO  
MULTIPLICA16X16:  
    CLRF    C3  
    CLRF    C2  
    CLRF    C1  
    CLRF    C0  
    BSF     C1, 7  
MULTL1:  
    RRF     A1, F  
    RRF     A0, F  
    BTFSS  STATUS, C  
    GOTO   MULTL2  
    MOVF   B0, W  
    ADDWF  C2, F  
    MOVF   B1, W  
    BTFSC  STATUS, C
```

```
INCFSZ    B1,W
ADDWF     C3,F
MULTL2:
    RRF      C3,F
    RRF      C2,F
    RRF      C1,F
    RRF      C0,F
    BTFSS    STATUS,C
    GOTO    MULTL1
    RETURN

INICIO:
    MOVLW    0xF1
    MOVWF    A0          ; A0= 0xF1
    CLRF     A1          ; A1= 0x00
    MOVLW    0x05
    MOVWF    B0          ; B0= 0x05
    MOVLW    0x1B
    MOVWF    B1          ; B1=0x1B
    CALL     MULTIPLICA16X16 ; chamada da sub-rotina de multiplicação de
                           ; 16 bits
    SLEEP
                           ; fim do programa, CPU em modo SLEEP

END
; resultado: C3=0, C2=0x19, C1=0x6F e C0=0xB5. Z='0', DC='1' e C='1'
; Ax=B=C=> 0x00F1 x 0x1B05 = 0x00196FB5
```

## 7.5.4 - Divisão

Além da multiplicação, podemos também realizar divisões no PIC. Veremos agora algumas formas de realizar divisões inteiras.

### 7.5.4.1 - Divisão Inteira com RRF

Conforme já vimos, podemos utilizar a instrução RLF para efetuar operações de multiplicação, mas o que aconteceria se em vez de um RLF utilizássemos uma instrução RRF? Teríamos uma operação de divisão.

Vejamos uma forma simples de dividir um valor de um registrador por um valor inteiro múltiplo de uma potência de 2:

```
; MACRO para Divisão com RRF. Esta macro está incluída no arquivo MACROS.ASM
; Utilização: DIV2 <registrarador>, <valor múltiplo de potência de 2>
DIV2 MACRO    REG,VAL
    BCF      STATUS,C           ; limpa o flag de carry
    IF (VAL==2)
        RRF      REG,F          ; rotaciona o registrador à esquerda
    ENDIF
    IF (VAL==4)
        RRF      REG,F          ; divide por 2
        BCF      STATUS,C          ; zera carry para não atrapalhar a pró
        RRF      REG,F          ; xima divisão
    ENDIF
    IF (VAL==8)
        RRF      REG,F          ; divide por 2
        BCF      STATUS,C          ; zera carry
        RRF      REG,F          ; divide novamente por 2 (total 4)
        BCF      STATUS,C          ; zera carry
```

```

        RRF      REG,F      ; divide novamente por 2 (total 8)
ENDIF
IF (VAL==16)
    RRF      REG,F      ; divide por 2
    BCF      STATUS,C   ; zera carry
    RRF      REG,F      ; divide novamente por 2 (total 4)
    BCF      STATUS,C   ; zera carry
    RRF      REG,F      ; divide novamente por 2 (total 8)
    BCF      STATUS,C   ; zera carry
    RRF      REG,F      ; divide novamente por 2 (total 16)
ENDIF
IF (VAL==32)
    RRF      REG,F      ; divide por 2
    BCF      STATUS,C   ; zera carry
    RRF      REG,F      ; divide novamente por 2 (total 4)
    BCF      STATUS,C   ; zera carry
    RRF      REG,F      ; divide novamente por 2 (total 8)
    BCF      STATUS,C   ; zera carry
    RRF      REG,F      ; divide novamente por 2 (total 16)
    BCF      STATUS,C   ; zera carry
    RRF      REG,F      ; divide novamente por 2 (total 32)
ENDIF
ENDM

```

### ***Exemplo***

```

LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
A0 EQU 0x20
ORG 0x0000
MOVLW D'25'
MOVWF A0           ; A0=D'25'
DIV2  A0,4         ; divide A0 por 4
MOVF  A0,W         ; W=D'6'
SLEEP
END

```

### **7.5.4.2 - Divisão Didática 8/8 bits**

Em seguida temos um exemplo de uma sub-rotina didática para realizar a divisão de duas variáveis de 8 bits.

Esta sub-rotina utiliza o princípio de que uma divisão pode ser realizada por meio de múltiplas subtrações, até que o valor do dividendo seja zero ou menor que zero.

```

; DIV8_8D - Divide dois números (A0 e B0) e coloca o resultado em C0;
; Entrada
;     A0 - Dividendo, B0 - Divisor
; Saída
;     C0 - Quociente
; As variáveis A0,B0 e C0 precisam ser definidas previamente;
; Esta rotina não testa se o divisor é zero !
DIV8_8D:
    CLRF    C0          ; apaga a variável C0
DIV8_8D_LOOP:

```

```

MOVF    B0,W           ; copia a divisor para W
SUBWF   A0,F           ; subtrai o divisor (B0) do dividendo (A0)
BTFS S STATUS,C        ; testa para ver se houve empréstimo
GOTO    DIV8_8D_MENOR  ; dividendo menor que zero, pula para
                  ; DIV8_8_D_MENOR
BTFS C STATUS,Z         ; testa se o dividendo foi igual a zero
GOTO    DIV8_8D_MENOR  ; se o dividendo igual a zero, pula para
                  ; DIV8_8_D_MENOR
INCF    C0,F           ; se o dividendo maior que zero incrementa o
                  ; quociente
GOTO    DIV8_8D_LOOP   ; retorna para novo ciclo de subtração
DIV8_8D_MENOR:
INCF    C0,F           ; se o dividendo <= zero, incrementa quociente
RETURN

```

### **Exemplo**

```

LIST P=16F627
INCLUDE <P16F627.INC>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
A0 EQU 0x20
B0 EQU 0x21
C0 EQU 0x22
ORG 0x0000
GOTO INICIO
DIV8_8D:
CLR F C0             ; apaga a variável C0
DIV8_8D_LOOP:
MOV F B0,W            ; copia a divisor para W
SUBWF A0,F            ; subtrai o divisor (B0) do dividendo (A0)
BTFS S STATUS,C        ; testa para ver se houve empréstimo
GOTO DIV8_8D_MENOR  ; dividendo menor que zero, pula para
                  ; DIV8_8_D_MENOR
BTFS C STATUS,Z         ; testa se o dividendo foi igual a zero
GOTO DIV8_8D_MENOR  ; se o dividendo igual a zero, pula para
                  ; DIV8_8_D_MENOR
INCF C0,F             ; se o dividendo maior que zero incrementa o
                  ; quociente
GOTO DIV8_8D_LOOP   ; retorna para novo ciclo de subtração
DIV8_8D_MENOR:
INCF C0,F             ; se o dividendo <= zero, incrementa quociente e...
RETURN
INICIO:
MOVLW D'180'
MOVWF A0              ; A0=180 decimal
MOVLW D'5'
MOVWF B0              ; B0=5 decimal
CALL DIV8_8D           ; chamada de sub-rotina
SLEEP                 ; coloca o chip em modo SLEEP
END
; resultado: 180/5=36 ou 0x24 hexadecimal. C0=0x24 hexadecimal. Z='0', DC='1' e
; C='1'

```

### **7.5.4.3 - Divisão Inteira 8/8 bits**

A sub-rotina seguinte é uma versão muito mais eficiente que a anterior para realizar a divisão de dois números inteiros:

```
; DIV8_8 - Divisão inteira 8/8 bits
; Entrada
;     A0 - Dividendo, B0 - Divisor
; Saída
;     W - Quociente
; As variáveis A0, B0, TEMP1, TEMP2 e TEMP3 devem ser definidas previamente
DIV8_8:
    MOVLW   0x08
    MOVWF   TEMP1
    CLRF    TEMP2
    MOVF    A0,W
    MOVWF   TEMP3
DIV8_8_LOOP:
    BCF    STATUS,C
    BTFSC  TEMP3,7
    BSF    STATUS,C
    RLF    TEMP2,F
    MOVF   B0,W
    SUBWF  TEMP2,F
    BTFSC  STATUS,C
    GOTO   DIV8_8_PTA
    ADDWF  TEMP2,F
    BCF    STATUS,C
DIV8_8_PTA:
    RLF    TEMP3,F
    DECFSZ TEMP1,F
    GOTO   DIV8_8_LOOP
    MOVF   TEMP3,W
    RETURN
```

### **Exemplo**

```
LIST P=16F627
INCLUDE <P16F627.INC>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
A0    EQU 0x20
B0    EQU 0x21
TEMP1 EQU 0x22
TEMP2 EQU 0x23
TEMP3 EQU 0x24
ORG   0x0000
GOTO  INICIO
DIV8_8:
    MOVLW   0x08
    MOVWF   TEMP1
    CLRF    TEMP2
    MOVF    A0,W
    MOVWF   TEMP3
DIV8_8_LOOP:
    BCF    STATUS,C
    BTFSC  TEMP3,7
    BSF    STATUS,C
    RLF    TEMP2,F
    MOVF   B0,W
    SUBWF  TEMP2,F
    BTFSC  STATUS,C
    GOTO   DIV8_8_PTA
    ADDWF  TEMP2,F
    BCF    STATUS,C
```

```

DIV8_8_PTA:
    RLF      TEMP3,F
    DECFSZ  TEMP1,F
    GOTO    DIV8_8_LOOP
    MOVF    TEMP3,W
    RETURN

INICIO:
    MOVLW   D'199'          ;
    MOVWF   A0               ; Dividendo (A0) = 199 decimal
    MOVLW   D'23'           ;
    MOVWF   B0               ; Divisor (B0) = 23 decimal
    CALL    DIV8_8           ; chamada da sub-rotina
    SLEEP

END
; resultado: W=0x08, Z='0', DC='0' e C='1'. 199/23=8,65...

```

### 7.5.5 - Raiz Quadrada Inteira

Para a extração da raiz quadrada de um número de 16 bits, utilizaremos um método de subtração de números ímpares, até que o valor do radicando seja menor ou igual a zero. A raiz inteira será então igual ao número total de subtrações efetuadas:

```

; Raiz Quadrada Inteira de um número de 16 Bits
; Variáveis utilizadas : A0(LSB) e A1(MSB): radicando, W raiz quadrada
; Outras variáveis utilizadas: B0, B1, TEMP1, TEMP2 e TEMP3
; Esta sub-rotina utiliza a sub-rotina SUB16

RAIZ16:
    CLRF    TEMP1          ; apaga o registrador temporário de raiz
    CLRF    B1              ;
    MOVLW   0x01            ;
    MOVWF   B0              ; inicializa o subtrator inicialmente para 0x0001

RAIZ16_LOOP:
    CALL    SUB16           ; subtrai o subtrator do radicando atual
    BTFSS  STATUS,C         ; testa para ver se o radicando é menor que zero
    GOTO   RAIZ16_RET        ; se menor, vai para a saída da sub-rotina
    MOVF   A0,F             ; testa se A0 ...
    BTFSS  STATUS,Z         ; é igual a zero ...
    GOTO   RAIZ16_CONT       ; se não, vai para RAIZ16_CONT

    GOTO   RAIZ16_CONT       ; se não, vai para RAIZ16_CONT
    MOVF   A1,F             ; testa se A1 ...
    BTFSC  STATUS,Z         ; é igual a zero ...
    GOTO   RAIZ16_FIM        ; se é, vai para RAIZ16_FIM

RAIZ16_CONT:
    INCF    TEMP1,F          ; incrementa o resultado atual da raiz
    MOVLW   0x02              ; adiciona 2 ...
    ADDWF  B0,F              ; ao subtrator ...
    BTFSC  STATUS,C          ; verifica se houve transbordo do LSB ...
    INCF    B1,F              ; e se houve incrementa um no MSB do subtrator
    GOTO   RAIZ16_LOOP        ; reinicia o loop para encontrar a raiz

RAIZ16_RET:
    MOVF   TEMP1,W          ; copia o resultado da raiz atual para o W
    RETURN

RAIZ16_FIM:
    INCF    TEMP1,W          ; incrementa um no resultado da raiz atual e
    RETURN                         ; guarda no W
                                    ; retorna da sub-rotina

SUB16:

```

```

MOVF   B0,W      ; subtrai os 8 bits
SUBWF A0,F      ; menos significativos
BTFS S STATUS,C ; testa se houve empréstimo
GOTO  SUB2      ; se houve pula para SUB2
MOVF   B1,W      ; se não, subtrai os
SUBWF A1,F      ; 8 bits mais significativos e
RETURN

SUB2:
MOVF   B1,W      ; subtrai os 8 bits
SUBWF A1,F      ; mais significativos e
MOVLW 0x01      ; subtrai 1 de A1 por conta
SUBWF A1,F      ; do empréstimo na subtração do LSB
RETURN

```

***Exemplo***

```

LIST P=16F627
INCLUDE <P16F627.INC>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF

A0 EQU 0x20
B0 EQU 0x21
TEMP1 EQU 0x22
TEMP2 EQU 0x23
TEMP3 EQU 0x24
A1 EQU 0x25
B1 EQU 0x26
ORG 0x0000
GOTO INICIO

RAIZ16:
    CLRF TEMP1      ; apaga o registrador temporário de raiz
    CLRF B1         ;
    MOVLW 0x01      ;
    MOVWF B0         ; inicializa o subtrator inicialmente para 0x0001

RAIZ16_LOOP:
    CALL SUB16      ; subtrai o subtrator do radicando atual
    BTFS STATUS,C   ; testa para ver se o radicando é menor que zero
    GOTO RAIZ16_RET ; se menor, vai para a saída da sub-rotina
    MOVF A0,F       ; testa se A0 ...
    BTFS STATUS,Z   ; é igual a zero ...
    GOTO RAIZ16_CONT ; se não, vai para RAIZ16_CONT
    MOVF A1,F       ; testa se A1 ...
    BTFS STATUS,Z   ; é igual a zero ...
    GOTO RAIZ16_FIM ; se é, vai para RAIZ16_FIM

RAIZ16_CONT:
    INCF TEMP1,F   ; incrementa o resultado atual da raiz
    MOVLW 0x02      ; adiciona 2 ...
    ADDWF B0,F      ; ao subtrator ...
    BTFS STATUS,C   ; verifica se houve transbordo do LSB ...
    INCF B1,F      ; e se houve incrementa um no MSB do subtrator
    GOTO RAIZ16_LOOP ; reinicia o loop para encontrar a raiz

RAIZ16_RET:
    MOVF TEMP1,W   ; copia o resultado da raiz atual para o W
    RETURN          ; retorna da sub-rotina

RAIZ16_FIM:
    INCF TEMP1,W   ; incrementa um no resultado da raiz atual e
    RETURN          ; guarda no W
                    ; retorna da sub-rotina

SUB16:
    MOVF B0,W      ; subtrai os 8 bits

```

```

SUBWF A0,F ; menos significativos
BTFS S STATUS,C ; testa se houve empréstimo
GOTO SUB2 ; se houve pula para SUB2.
MOVF B1,W ; se não, subtrai os
SUBWF A1,F ; 8 bits mais significativos e
RETURN ; retorna

SUB2:
MOVF B1,W ; subtrai os 8 bits
SUBWF A1,F ; mais significativos e
MOVLW 0x01 ; subtrai 1 de A1 por conta
SUBWF A1,F ; do empréstimo na subtração do LSB
RETURN ; retorna

INICIO:
MOVLW 0X39 ;
MOVWF A0 ;
MOVLW 0x1C ;
MOVWF A1 ; radicando= 0x1C39 (7225 decimal)
CALL RAIZ16 ; chamada da sub-rotina
SLEEP ; fim do programa

END
; Resultado: W=0x55 (85 decimal) --> a raiz quadrada de 7225 é igual a 85 !

```

## 7.5.6 - Resolução de Equações Simples

Veremos agora, algumas técnicas de programação que podemos utilizar para resolver equações de pequena e até média complexidade:

Digamos que em determinada aplicação, é necessário calcular a média entre quatro variáveis, as quais chamaremos aqui de VA, VB, VC e VD. O resultado (média) deve ser armazenado na variável RESU.

Ora, sabemos que para calcular a média entre quatro números, devemos somar os quatro e então dividir o resultado por 4.

Uma implementação simples para a resolução de um problema como este seria realizar a soma sucessiva das quatro variáveis e em seguida duas rotações à direita. O programa seguinte calcula a média entre as quatro variáveis sendo: VA=D'100', VB=D'50', VC=D'100' e VD=D'50'. Sabemos com antecedência que a média aritmética deve ser igual a 75. Vejamos então o programa:

```

; Média de quatro valores
LIST P=16F627
INCLUDE <P16F627.INC>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
VA EQU 0x20 ; associa o registrador GPR do endereço 0x20 com
              ; a variável VA
VB EQU 0x21 ; associa o registrador GPR do endereço 0x21
              ; com a variável VB
VC EQU 0x22 ; associa o registrador GPR do endereço 0x22
              ; com a variável VC
VD EQU 0x23 ; associa o registrador GPR do endereço 0x23
              ; com a variável VD
RESUL EQU 0x24 ; variável RESU menos significativa no endereço
                  ; 0x24

```

```

RESUH EQU 0x25 ; variável RESU mais significativa no endereço
; 0x25
ORG 0x0000 ; início do programa no endereço 0000 Z C
MOVLW D'100' ; W=D'100' X X
MOVWF VA ; VA=D'100' X X
MOVLW D'50' ; W=D'50' X X
MOVWF VB ; VB=D'50' X X
MOVLW D'100' ; W=D'100' X X
MOVWF VC ; VC=D'100' X X
MOVLW D'50' ; W=D'50' X X
MOVWF VD ; VD=D'50' X X
CLRF RESUL ; RESUL=0 1 X
CLRF RESUH ; RESUH=0 1 X
MOVF VA,W ; W=D'100' 0 X
ADDWF VB,W ; W=W+VB=D'150' 0 0
BTFS C STATUS,C ; TESTA O CARRY, PULA SE 0 0 0
INCF RESUH,F ; NÃO FOI EXECUTADA
ADDWF VC,W ; W=W+VC=D'150'+D'100'=D'250' 0 0
BTFS C STATUS,C ; TESTA O CARRY, PULA SE 0 0 0
INCF RESUH,F ; NÃO FOI EXECUTADA
ADDWF VD,W ; W=W+VD=250+50=300(-256)=D'44' 0 1
BTFS C STATUS,C ; TESTA O CARRY, PULA SE 0 0 1
INCF RESUH,F ; incrementa RESUH (=D'1') 0 1
MOVWF RESUL ; RESU=W=D'44' 0 0
BCF C STATUS,C ; C=0 0 0
RRF RESUH,F ; RESUH B'00000001' -> B'00000000' 0 1
RRF RESUL,F ; RESUL B'00101100' -> B'10010110' 0 0
BCF C STATUS,C ; C=0 0 0
RRF RESUH,F ; RESUH B'00000000' -> B'00000000' 0 0
RRF RESUL,F ; RESUL B'10010110' -> B'01001011' 0 0
END
; Resultado: RESU = B'01001011' = D'75' !

```

Observe, no exemplo, a utilização de duas instruções RRF para realizar uma divisão por 4.

De fato, podemos resolver alguns tipos de equações rapidamente no PIC. Para isso devemos ter em mente que as multiplicações podem ser separadas em operações simples. Observe:

$$7^*5=35$$

Veja que a multiplicação por cinco possui uma implementação mais complexa nos PICs, o que significa um aumento no tempo de execução e consumo de memória de programa, contudo a mesma operação pode ser rescrita como:

$$7^*4+7=35$$

Esta operação, por sua vez, pode ser facilmente implementada no PIC:

```

LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF

```

A0 EQU 0x20

```

ORG 0x0000          Z DC C
MOVlw 0x07    ; W=0x07      X X X
MOVwf A0      ; A0=0x07      X X X
MULT2 A0,4    ; A0=0x1C (7x4=28) X X 0
ADDWF A0,F    ; A0=0x23=35 decimal 0 1 0
SLEEP

END

```

## 7.6 - Conversão Binário/Hexadecimal/Decimal

A seguir, apresentamos um conjunto de sub-rotinas para a conversão de base numérica.

### 7.6.1 - Conversão Binário em Decimal (8 bits)

A sub-rotina seguinte pode ser utilizada para a conversão de um valor binário armazenado no registrador W em um valor decimal armazenado nas variáveis A0, A1 e A2 da seguinte forma: A0 - unidades, A1 - dezenas e A2 - centenas.

```

; BIN8DEC - Conversão de binário 8 bits em decimal
; Entrada:
;   W - número em binário a ser convertido
; Saída:
;   D0 - unidades
;   D1 - dezenas
;   D2 - centenas
; As variáveis D0,D1 e D2 precisam ser definidas previamente no programa
BIN8DEC:
    MOVWF D0           ; copia o número a ser convertido para a variável
                        ; de unidades
    CLRF D1           ; limpa as dezenas
    CLRF D2           ; limpa as centenas
BIN8DEC_1:
    MOVLW D'100'       ; subtrai 100 decimal ...
    SUBWF D0,W         ; ... do valor a ser convertido e guarda em W
    BTFSS STATUS,C    ; o resultado é positivo ou zero ?
    GOTO BIN8DEC_2     ; não ? então vai para a próxima parte
    MOVWF D0           ; sim ? então copia o valor para as unidades ...
    INCF D2,F          ; incrementa um nas centenas
    GOTO BIN8DEC_1     ; e reinicia o ciclo
BIN8DEC_2:
    MOVLW D'10'         ; subtrai 10 decimal ...
    SUBWF D0,W         ; ... do valor a ser convertido e guarda o
                        ; resultado em W
    BTFSS STATUS,C    ; o resultado é positivo ou zero ?
    RETURN             ; não ? então retorna, a conversão terminou
    MOVWF D0           ; sim ? então copia o resultado em W para as
                        ; unidades ...
    INCF D1,F          ; incrementa um nas dezenas
    GOTO BIN8DEC_2     ; e reinicia o ciclo

```

## 7.6.2 - Conversão Decimal em Binário (8 bits)

A sub-rotina seguinte destina-se a converter um valor decimal armazenado em três variáveis (centenas, dezenas e unidades) em um valor binário armazenado em um único registrador de 8 bits:

```

; DEC8BIN - Conversão de decimal em binário 8 bits
; Entrada:
;   D0 - unidades
;   D1 - dezenas
;   D2 - centenas
; Saída:
;   D0 - resultado em binário/hexadecimal
; As variáveis D0,D1 e D2 precisam ser definidas previamente no programa e são
; alteradas pela sub-rotina
DEC8BIN:
    MOVLW 0x01      ; copia 1 em W
    SUBWF D2,F      ; subtrai 1 das centenas ...
    BTFSS STATUS,C  ; o resultado é positivo ou zero ?
    GOTO DEC8BIN_1  ; não ? então vai para a próxima fase
    MOVLW D'100'    ; sim ? então adiciona 100 decimal ...
    ADDWF D0,F      ; ... ao resultado binário armazenado em D0
    GOTO DEC8BIN    ; reinicia o ciclo
DEC8BIN_1:
    MOVLW 0x01      ; copia 1 em W
    SUBWF D1,F      ; subtrai 1 das dezenas ...
    BTFSS STATUS,C  ; o resultado é positivo ou zero ?
    RETURN          ; não ? então retorna, a conversão terminou
    MOVLW D'10'     ; sim ? então soma 10 decimal
    ADDWF D0,F      ; ao valor a resultado binário em D0
    GOTO DEC8BIN_1  ; reinicia o ciclo

```

## 7.6.3 - Conversão Binário em Decimal (16 bits)

A sub-rotina seguinte destina-se a converter um valor binário armazenado em duas variáveis (A0 e A1) em um valor decimal armazenado em cinco variáveis: D0 para as unidades, D1 para as dezenas, D2 para as centenas, D3 para as milhares e D4 para as dezenas de milhares.

```

; BIN16DEC - Conversão de binário 16 bits em decimal
; Entrada:
;   A0 - LSB binário
;   A1 - MSB binário
; Saída:
;   D0 - unidades
;   D1 - dezenas
;   D2 - centenas
;   D3 - milhares
;   D4 - dezenas de milhares
; As variáveis D0,D1,D2,D3,D4,B0,B1,A0,A1,TEMPO1 e TEMP1 precisam ser definidas
; previamente no programa.
BIN16DEC:
    CLRF D0          ; apaga as unidades
    CLRF D1          ; apaga as dezenas
    CLRF D2          ; apaga as centenas
    CLRF D3          ; apaga as milhares

```

```

CLRF    D4          ; apaga as dezenas de milhares
BIN16DEC_1:
MOVF    A0,W        ; copia o LSB binário ...
MOVWF   TEMPO0      ; ... para TEMPO0
MOVF    A1,W        ; copia o MSB binário ...
MOVWF   TEMP11      ; ... para TEMP1
MOVLW   0x10        ;
MOVWF   B0          ;
MOVLW   0x27        ;
MOVWF   B1          ; B=0x2710 (10000 decimal)
CALL    SUB16B      ; subtrai 10000 decimal do valor binário
BTFS S STATUS,C    ; testa se o resultado é positivo ou zero
GOTO   BIN16DEC_2   ; não ? então vai para a próxima fase
INCFL  D4,F        ; sim ? então incrementa as dezenas de milhares
GOTO   BIN16DEC_1   ; reinicia o ciclo

BIN16DEC_2:
MOVF    TEMPO0,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 binário ...
MOVWF   TEMPO0      ; ... para TEMPO0
MOVF    A1,W        ; copia o MSB binário ...
MOVWF   TEMP11      ; ... para TEMP1
MOVLW   0xE8         ;
MOVWF   B0          ;
MOVLW   0x03         ;
MOVWF   B1          ; B=03E8 (1000 decimal)
CALL    SUB16B      ; subtrai 1000 decimal do valor binário
BTFS S STATUS,C    ; testa se o resultado é positivo ou zero
GOTO   BIN16DEC_4   ; não ? então vai para a próxima fase
INCFL  D3,F        ; sim ? então incrementa as milhares
GOTO   BIN16DEC_3   ; reinicia o ciclo

BIN16DEC_4:
MOVF    TEMPO0,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 binário ...
MOVWF   TEMPO0      ; ... para TEMPO0
MOVF    A1,W        ; copia o MSB binário ...
MOVWF   TEMP11      ; ... para TEMP1
MOVLW   D'100'       ;
MOVWF   B0          ;
CLRF    B1          ; B=100 decimal
CALL    SUB16B      ; subtrai 100 decimal do valor binário (A)
BTFS S STATUS,C    ; testa se o resultado é positivo ou zero
GOTO   BIN16DEC_6   ; não ? então vai para a próxima fase
INCFL  D2,F        ; sim ? então incrementa as centenas
GOTO   BIN16DEC_5   ; reinicia o ciclo

BIN16DEC_6:
MOVF    TEMPO0,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 binário ...
MOVWF   TEMPO0      ; ... para TEMPO0
MOVF    A1,W        ; copia o MSB binário ...
MOVWF   TEMP11      ; ... para TEMP1

```

```

MOVLW D'10'          ;
MOVWF B0             ;
CLRF   B1             ; B=10 decimal
CALL   SUB16B         ; subtrai 10 decimal do valor binário (A)
BTFS S STATUS,C      ; testa se o resultado é positivo ou zero
GOTO  BIN16DEC_8      ; não ? então vai para fase final
INCF  D1,F           ; sim ? então incrementa as dezenas
GOTO  BIN16DEC_7      ; reinicia o ciclo
BIN16DEC_8:
    MOVF  TEMPO,W      ; copia o valor temporário restante ...
    MOVWF D0            ; ... para as unidades
    RETURN
SUB16B:
    MOVF  B0,W          ;
    SUBWF A0,F          ;
    MOVLW 0x01           ;
    BTFS S STATUS,C      ;
    SUBWF A1,F          ;
    BTFS S STATUS,C      ;
    GOTO  EMPRESTA
    MOVF  B1,W          ;
    SUBWF A1,F          ;
FIM_SUB16B:
    MOVF  A0,F          ;
    BTFSC STATUS,Z       ;
    MOVF  A1,F          ;
    RETURN
EMPRESTA:
    MOVF  B1,W          ;
    SUBWF A1,F          ;
    BCF   STATUS,C       ;
    GOTO  FIM_SUB16B

```

## 7.6.4 - Conversão Decimal em Binário (16 bits)

A sub-rotina seguinte pode ser utilizada para converter um valor decimal armazenado em cinco variáveis: D0 para as unidades, D1 para as dezenas, D2 para as centenas, D3 para as milhares e D4 para as dezenas de milhares, para um valor binário armazenado em duas variáveis: A0 (LSB) e A1(MSB).

```

; DEC16BIN - Conversão de decimal em binário 16 bits
; Entrada:
;     D0 - unidades
;     D1 - dezenas
;     D2 - centenas
;     D3 - milhares
;     D4 - dezenas de milhares
; Saída:
;     A0 - LSB binário
;     A1 - MSB binário
; As variáveis D0,D1,D2,D3,D4,B0,B1,A0,A1,TEMPO1 precisam ser definidas
; previamente no programa.
DEC16BIN:
    CLRF   A1             ; apaga o resultado MSB
    MOVF  D0,W           ; copia as unidades para W
    MOVWF A0             ; copia as unidades para o resultado LSB
DEC16BIN_1:
    MOVLW 0x01           ; subtrai 1 ...

```

```

SUBWF    D4,F           ; das dezenas de milhares
BTFS S  STATUS,C        ; verifica se o resultado é negativo
GOTO     DEC16BIN_2      ; se for negativo vai para DEC16BIN_2
MOVLW   0x27            ; soma ...
ADDWF   A1,F            ; ...
MOVLW   0x10            ; 0x2710 (10000 decimal) ...
ADDWF   A0,F            ; ao resultado
BTFS C  STATUS,C        ; verifica se o resultado 1SB transbordou
INCF    A1,F            ; se transbordou, incrementa o MSB
GOTO     DEC16BIN_1      ; 

DEC16BIN_2:
MOVLW   0x01            ; subtrai 1 ...
SUBWF   D3,F            ; das milhares
BTFS S  STATUS,C        ; verifica se o resultado é negativo
GOTO     DEC16BIN_3      ; se for negativo vai para DEC16BIN_3
MOVLW   0x03            ; soma ...
ADDWF   A1,F            ; ...
MOVLW   0xE8            ; ...
ADDWF   A0,F            ; 0x03E8 (10000 decimal) ao resultado
BTFS C  STATUS,C        ; verifica se o resultado LSB transbordou
INCF    A1,F            ; se transbordou, incrementa o MSB
GOTO     DEC16BIN_2      ; 

DEC16BIN_3:
MOVLW   0x01            ; subtrai 1 ...
SUBWF   D2,F            ; das centenas
BTFS S  STATUS,C        ; verifica se o resultado é negativo
GOTO     DEC16BIN_4      ; se for negativo vai para DEC16BIN_4
MOVLW   D'100'           ; soma ...
ADDWF   A0,F            ; 100 decimal ao resultado
BTFS C  STATUS,C        ; verifica se o resultado LSB transbordou
INCF    A1,F            ; se transbordou, incrementa o MSB
GOTO     DEC16BIN_3      ; 

DEC16BIN_4:
MOVLW   0x01            ; subtrai 1 ...
SUBWF   D1,F            ; das dezenas
BTFS S  STATUS,C        ; verifica se o resultado é negativo
RETURN
; se for negativo, retorna
MOVLW   D'10'            ; ...
ADDWF   A0,F            ; soma 10 ao resultado LSB
BTFS C  STATUS,C        ; verifica se transbordou
INCF    A1,F            ; se transbordou, incrementa o resultado MSB
GOTO     DEC16BIN_4      ; 

```

A seguir, apresentamos uma outra versão para a conversão decimal/binário que, apesar de menos intuitiva, é muito mais rápida que a versão anterior.

```

;*****
; DEC2BIN16 - Converte um número decimal para binário 16 bits
; Entrada
;     D0,D1,D2,D3,D4 - Dígitos decimais (unidades, dezenas, centenas, milhares
;     e dezenas de milhares
; Saída
;     A0,A1 - resultado binário (A0-LSB e A1-MSB)
;
; Tamanho=35 instruções Tempo : 2+34+2=38 ciclos de instrução
;
; bin = (((D1+D3+D4*256)*2+
;         D2+D2*16+D3*256)*2+
;         D1+D2*16-D3*16+D4*16*256)*2+
;         D0+D4*16-D4*256
;
```

; Janeiro 24, 2001 por Nikolai Golovchenko E-mail: golovchenko@mail.ru

```
*****
DEC2BIN16:
MOVE    D1,W      ; copia as dezenas em W
ADDWF   D3,W      ; soma as milhares com as dezenas e coloca em W
MOVWF   A0          ; copia a soma das milhares com as dezenas e guarda em A0
ADDWF   A0,F      ; soma A0 com ele mesmo ou seja, multiplica A0 por 2
RLF     D4,W      ; multiplica as dezenas de milhares por 2 e coloca em W
MOVWF   A1          ; armazena o resultado em A1
SWAPF   D2,W      ; inverte os nibbles das centenas e coloca em W
ADDWF   D2,W      ; soma as centenas e coloca em W
ADDWF   A0,F      ; soma o resultado ao A0
MOVF    D3,W      ; copia as milhares em W
ADDWF   A1,F      ; soma com A1
RLF     A0,F      ; multiplica o resultado geral ...
RLF     A1,F      ; ... por 2
SWAPF   D2,W      ; inverte os nibbles das centenas e coloca em W
ADDWF   D1,W      ; soma com as dezenas e coloca em W
ADDWF   A0,F      ; soma ao resultado em A0
SWAPF   D4,W      ; inverte os nibbles das dezenas de milhares
SKPNC
IORLW  0x01      ; seta o bit 0 de W
ADDWF   A1,F      ; soma o W com o resultado MSB
SWAPF   D3,W      ; inverte os nibbles das milhares e coloca em W
SUBWF   A0,F      ; subtrai do resultado LSB em A0
SKPC
DECF    A1,F      ; pula a próxima instrução se o flag C=1
CLRC
DECF    A1,F      ; decrementa o resultado MSB
CLRC
RLF     A0,F      ; limpa o flag C (C=0)
RLF     A0,F      ; multiplica o resultado atual ...
RLF     A1,F      ; ... por 2
SWAPF   D4,W      ; inverte os nibbles das dezenas de milhares e coloca em W
ADDWF   D0,W      ; soma com as unidades e coloca em W
ADDWF   A0,F      ; soma ao resultado temporário A0
MOVF    D4,W      ; copia as dezenas de milhares em W
SKPNC
DECF    D4,W      ; pula a próxima instrução se o flag C=0
SUBWF   A1,F      ; decrementa as dezenas de milhares e coloca em W (W=D4-1)
RETURN
*****
```

## 7.7 - Tabelas

Tabelas são estruturas de dados muito utilizadas para o armazenamento de informações constantes e que são apontadas por um índice.

Como exemplos podemos citar uma tabela de senos, em que para cada entrada de ângulo temos um valor correspondente ao seno daquele.

Um outro exemplo seria um sistema de ignição de motores, em que para cada faixa de rotação do motor teremos um valor correspondente ao avanço ou atraso da ignição.

O uso de tabelas nos PICs não é o que podemos chamar de simples. Ao contrário de outros microcontroladores, os PICs não possuem uma arquitetura que facilite o uso de tabelas; pelo contrário, o programador necessita de alguns artifícios para a utilização de tabelas com eles, mas nada que inviabilize ou dificulte grandemente a tarefa de programação.

Basicamente, todo o suporte à criação de tabelas de dados nos PICs é unicamente fornecido pela instrução RETLW.

Como já foi visto, esta instrução, além de suas atribuições normais de retorno de sub-rotina, também permite que seja carregado um valor no registrador W.

Além da instrução RETLW, outro aspecto interno do PIC que deve ser explorado na criação de tabelas é o desvio de programa com o auxílio do registrador PCL.

Como já foi visto, o registrador PCL corresponde aos 8 bits menos significativos do contador de programa (PC). Isto significa que uma escrita no registrador PCL pode provocar (como provoca) um desvio imediato do programa, para o endereço especificado pelo registrador PCL (em conjunto com os bits MSB armazenados no registrador PCLATH).

Como exemplo de implementação de uma tabela no PIC, vejamos um programa simples para determinar o quadrado de um número qualquer (qualquer neste caso é um número entre 1 e 15 !):

```
; Exemplo de quadrado
LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
ORG 0x0000
GOTO INICIO
QUADRADO:
    ADDWF PCL,F      ; soma o índice da tabela ao PCL e desvia
    RETLW 0          ; se W=0, retorna W=0
    RETLW D'1'        ; se W=1, retorna W=1
    RETLW D'4'        ; se W=2, retorna W=4
    RETLW D'9'        ; se W=3, retorna W=9
    RETLW D'16'       ; se W=4, retorna W=16 decimal
    RETLW D'25'       ; se W=5, retorna W=25 decimal
    RETLW D'36'       ; se W=6, retorna W=36 decimal
    RETLW D'49'       ; se W=7, retorna W=49 decimal
    RETLW D'64'       ; se W=8, retorna W=64 decimal
    RETLW D'81'       ; se W=9, retorna W=81 decimal
    RETLW D'100'      ; se W=10 decimal, retorna W=100 decimal
    RETLW D'121'      ; se W=11 decimal, retorna W=121 decimal
    RETLW D'144'      ; se W=12 decimal, retorna W=144 decimal
    RETLW D'169'      ; se W=13 decimal, retorna W=169 decimal
    RETLW D'196'      ; se W=14 decimal, retorna W=196 decimal
    RETLW D'225'      ; se W=15 decimal, retorna W=225 decimal
INICIO:
    MOVLW 0x05        ; coloca 5 decimal em W
    CALL QUADRADO    ; chama sub-rotina QUADRADO
    SLEEP             ; fim do programa
END
```

Observe no entanto que este método apresenta algumas limitações:

1. Não verifica se houve ou não cruzamento de página de programa;

2. Não verifica se o índice fornecido em W é maior do que 15 decimal (o que poderia provocar um colapso do programa);
3. Não permite que uma tabela resida em uma região de memória situada em área de transição do PCL, ou seja, em uma área em que aconteça um transbordo do registrador PCL, como, por exemplo, do endereço 0x00FF para o endereço 0x0100.

A seguir, apresentamos uma alternativa ao exemplo anterior e que implementa correções para evitar os problemas descritos anteriormente:

```

; Exemplo de quadrado
LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF

TEMP EQU 0x20
ORG 0x0000
GOTO INICIO

QUADRADO:
    MOVWF TEMP ; armazena o número na variável TEMP
    JPLS QUADRADO2,TEMP,D'15' ; testa se TEMP>15, se for desvia para QUA
    RETURN ;
QUADRADO2:
    MOVLW LOW TABELA+1 ; copia em W os 8 bits LSB do endereço da
    ; tabela (+1)
    ADDWF TEMP,F ; adiciona o valor à variável TEMP
    MOVLW HIGH TABELA ; copia em W os 5 bits superiores do endereço
    ; da TABELA
    BTFSC STATUS,C ; testa para ver se a soma anterior transbordou
    ADDLW 0x01 ; se transbordou, soma 1 ao W
    MOVWF PCLATH ; acerta o PCLATH de acordo com o endereço da
    ; TABELA
    MOVF TEMP,W ; copia o valor do TEMP para o W

TABELA:
    MOVLW PCL ; copia o W para o PCL (desvia para a tabela)
    RETLW 0 ; se W=0, retorna W=0
    RETLW D'1' ; se W=1, retorna W=1
    RETLW D'4' ; se W=2, retorna W=4
    RETLW D'9' ; se W=3, retorna W=9
    RETLW D'16' ; se W=4, retorna W=16 decimal
    RETLW D'25' ; se W=5, retorna W=25 decimal
    RETLW D'36' ; se W=6, retorna W=36 decimal
    RETLW D'49' ; se W=7, retorna W=49 decimal
    RETLW D'64' ; se W=8, retorna W=64 decimal
    RETLW D'81' ; se W=9, retorna W=81 decimal
    RETLW D'100' ; se W=10 decimal, retorna W=100 decimal
    RETLW D'121' ; se W=11 decimal, retorna W=121 decimal
    RETLW D'144' ; se W=12 decimal, retorna W=144 decimal
    RETLW D'169' ; se W=13 decimal, retorna W=169 decimal
    RETLW D'196' ; se W=14 decimal, retorna W=196 decimal
    RETLW D'225' ; se W=15 decimal, retorna W=225 decimal

INICIO:
    MOVLW 0x05 ; coloca 5 decimal em W
    CALL QUADRADO ; chama sub-rotina QUADRADO
    SLEEP ; fim do programa

END

```

## 7.8 - Acesso Indireto à Memória

Um dos modos de acesso à memória RAM do PIC é chamado de acesso indireto no qual se utilizam dois registradores SFR para realizar o acesso a qualquer endereço da memória RAM:

- ◆ FSR - utilizado para especificação do endereço do registrador a ser acessado indiretamente;
- ◆ INDF - utilizado para escrita/leitura do conteúdo do endereço especificado pelo registrador FSR.

Observe o exemplo em seguida:

```

LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF

ORG 0x0000
    MOVLW 0x85      ; seleciona o registrador 0x85 (TRISA)
    MOVWF FSR       ; apaga o registrador 0x85 (TRISA)
    CLRF INDF      ; incrementa FSR (0x86 - TRISB)
    INCF FSR,F     ; configura os 4 primeiros pinos como entradas ...
    MOVLW B'00001111' ; e os quatro MSB como saídas (registrador TRISB)
    MOVWF INDF      ; fim do programa
    SLEEP
END

```

Como o registrador FSR pode especificar 256 endereços diferentes, significa que no acesso indireto podemos acessar o conteúdo de dois bancos de memória de cada vez (0 e 1, ou 2 e 3).

Para alternar entre os bancos de memória, utilizamos o bit IRP do registrador STATUS.

```

LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF

ORG 0x0000
    BSF STATUS, IRP   ; seleciona os bancos 2 e 3 para acesso indireto
    MOVLW 0x81         ; endereço 0x81 (banco 3, registrador OPTION_REG)
    MOVWF FSR          ; seleciona o registrador 0x81 (OPTION_REG)
    CLRF INDF          ; apaga o registrador 0x81 (OPTION_REG)
    BCF STATUS, IRP    ; seleciona os bancos 0 e 1 para acesso indireto
    MOVLW 0x06         ; endereço 0x06 (banco 1, registrador PORTB)
    MOVWF FSR          ; seleciona o registrador 0x06 (PORTB)
    MOVF INDF,W        ; realiza a leitura do registrador PORTB e coloca
                       ; em W
    SLEEP              ; fim do programa
END

```

## 7.9 - Configurando os Timers

### 7.9.1 - Configurando o Timer 0

Veremos agora os procedimentos necessários para a configuração do timer 0 para trabalhar como timer para fornecer uma base de tempo de um segundo.

Podemos utilizar a fórmula seguinte para saber quantas interrupções o TMR0 irá gerar a cada segundo:

$$\text{Número de interrupções / segundo} = \text{Fosc} / 4 / \text{Prescaler} / \text{TMR0}$$

Ou seja, considerando que o Prescaler esteja configurado para dividir por 256, que o TMR0 seja utilizado diretamente e a freqüência de operação do PIC seja de 4Mhz, teremos:

$$\text{Número de interrupções / segundo} = 4\text{Mhz} / 4 / 256 / 256$$

O que resulta em aproximadamente 15,29 interrupções por segundo (15,29 Hz).

O programa para cronometrar a passagem de cada segundo poderia ser:

```
; Programa relógio
LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
SEGUNDOS EQU 0x20
TEMPO EQU 0x21
ORG 0x0000
GOTO INICIO
ORG 0x0004
    SALVA_CONTEXTO ;
BTFSZ INTCON,T0IF ; testa p/ ver se foi interrupção do TMR0
GOTO TIMER0 ; se T0IF=1 vai p/ TIMER0
RESTAURA_CONTEXTO ;
RETFIE ; se não retorna da interrupção
TIMER0:
    BCF INTCON,T0IF ; limpa sinalizador de interrupção do TMR0
DECFSZ TEMPO,F ; decremente um do tempo, se tempo=0 pula a
; próxima instrução se tempo não é zero retorna da
MOVLW D'15' ; interrupção restaura o valor
MOVWF TEMPO ; da variável TEMPO (TEMPO = 15)
INCF SEGUNDOS,F ; incrementa 1 segundo
RETFIE ; retorna da interrupção
INICIO:
    BSF STATUS,RP0 ; muda p/ banco 1
    MOVLW B'10000111'
    MOVWF OPTION_REG ; Configura o OPTION_REG: Resistores de Pull-up
; desligados, clock interno p/ TMR0, prescaler
; ligado ao TMR0 dividindo por 256
    BCF STATUS,RP0 ; volta p/ banco 0
    MOVLW B'10100000'
    MOVWF INTCON ; Configura interrupções: liga GIE e T0IE
    ...
    ...
END
```

## 7.9.2 - Configurando o Timer 1

A seguir, veremos os procedimentos para configurar o módulo timer 1 para operação como um timer, para fornecer uma base de tempo de um centésimo de segundo (ou 10 milissegundos).

Considerando que a freqüência de operação do PIC é de 4Mhz, temos que:

$$F_{INT} = CLOCK / PRESCALER / (65536 - TMR1)$$

Em que:  $F_{INT}$  - é a freqüência de interrupções do timer 1;

$CLOCK$  - é a freqüência de clock do timer 1 (no caso  $F_{osc}/4$ );

$PRESCALER$  - é o fator de divisão do prescaler do timer 1;

$TMR1$  - é o valor inicial do registrador TMR1.

Desta forma, considerando que o prescaler do timer 1 seja configurado para fator de divisão 1, teremos que:

$$100 = 1.000.000,00 / 1 / (65536-TMR1) \text{ ou,}$$

$$TMR1 = 55536$$

Isto significa que o timer 1 deve iniciar sua contagem em 55536 e a cada transbordo (interrupção TMR1IF) este valor deve novamente ser carregado no registrador TMR1.

Vejamos então o programa para realizar tal tarefa:

```
; Programa relógio
LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
CENTESIMOS EQU 0x20
TEMPO EQU 0x21

ORG 0x0000
GOTO INICIO

ORG 0x0004
SALVA_CONTEXTO
    BTFSC PIR1,TMR1IF ; testa p/ ver se foi interrupção do TMR1
    GOTO TIMER1 ; se TMR1IF=1 vai p/ TIMER1
FIM_INT:
    RESTAURA_CONTEXTO ;
    RETFIE ; se não retorna da interrupção
TIMER1:
    BCF T1CON,TMR1ON ; desliga o timer 1
    MOVLW 0xF0 ; copia o valor 55536 ...
    MOVWF TMR1L ; ... de volta ...
    MOVLW 0xD8 ; ... para o ...
    MOVWF TMR1H ; ... registrador TMR1
    BCF PIR1,TMR1IF ; limpa sinalizador de interrupção do TMR1
    BSF T1CON,TMR1ON ;
    INCF CENTESIMOS,F ; incrementa 1 na variável de centésimos
```

```

GOTO      FIM_INT      ;
INICIO:
    MOVLW  B'00000001'  ; configura o timer 1 para ativado, clock interno
    MOVWF  T1CON        ; ... prescaler 1:1
    BSF    STATUS,RP0   ; seleciona o banco 1
    MOVLW  B'00000001'  ; habilita interrupção individual ...
    MOVWF  PIE1         ; ... do timer 1
    BCF    STATUS,RP0   ; seleciona o banco 0
    BSF    INTCON,GIE  ; habilita o GIE
    ...
END

```

### 7.9.3 - Configurando o Timer 2

A seguir, veremos os procedimentos para configurar o módulo timer 2 para operação como um timer, para fornecer uma base de tempo de um milésimo de segundo (ou 1 milissegundo).

Considerando que a freqüência de operação do PIC é de 4Mhz, temos que:

$$F_{INT} = CLOCK / PRESCALER / (256 - PR2) / POSTSCALER,$$

Em que:  $F_{INT}$  - é a freqüência de interrupções do timer 2;

$CLOCK$  - é a freqüência de clock do timer 2 ( $F_{osc}/4$ );

$PRESCALER$  - é o fator de divisão do prescaler do timer 2;

$PR2$  - é o valor armazenado no registrador de período do timer 2;

$POSTSCALER$  - é o fator de divisão do postscaler do timer 2.

Desta forma, considerando que o prescaler esteja configurado para divisão por 4 e o postscaler para divisão por 2, teremos que:

$$1000 = 1.000.000,00 / 4 / (256-PR2) / 2 \text{ ou,}$$

$$PR2 = 131$$

Isto significa que o timer 2 deve iniciar sua contagem em 131 e a cada transbordo (interrupção TMR2IF) este valor deve novamente ser carregado no registrador TMR2, o que acontece automaticamente graças ao registrador PR2.

Vejamos então o programa para realizar tal tarefa:

```

; Programa relógio
LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
MILESIMOS    EQU      0x20
TEMPO        EQU      0x21

ORG 0x0000
GOTO INICIO
ORG 0x0004
SALVA_CONTEXTO ;

```

```

BTFS C PIR1,TMR2IF ; testa p/ ver se foi interrupção do TMR2
GOTO TIMER2 ; se TMR2IF=1 vai p/ TIMER2
FIM_INT:
RESTAURA_CONTEXTO ;
RETFIE ; se não retorna da interrupção
TIMER2:
BCF PIR1,TMR2IF ; limpa sinalizador de interrupção do TMR2
INCF MILESIMOS,F ; incrementa 1 à variável de milésimos
GOTO FIM_INT ;
INICIO:
MOVLW B'00001101' ; configura o timer 2 para ativado ...
MOVWF T2CON ; ... prescaler 1:4 e postscaler 1:2
BSF STATUS,RP0 ; seleciona o banco 1
MOVLW B'00000010' ; habilita interrupção individual ...
MOVWF PIE1 ; ... do timer 1
BCF STATUS,RP0 ; seleciona o banco 0
BSF INTCON,GIE ; habilita o GIE
...
END

```

## 7.10 - Medindo a Largura de um Pulso

Vejamos agora uma demonstração da utilização do módulo CCP para a medição da largura de um pulso aplicado ao pino CCP1 do microcontrolador.

Para a captura da largura do sinal aplicado à entrada do módulo CCP, devemos inicialmente configurar o módulo para o modo de captura 1 (captura a cada borda de subida do sinal de entrada).

Além disso, devemos configurar o timer 1 para trabalhar como timer (clock interno) e configurar o prescaler de acordo com a resolução desejada para a medição.

Observe que o clock do timer 1 possui influência direta na resolução e tamanho máximo do pulso a ser medido.

Considerando que o clock do microcontrolador seja de 4Mhz, se utilizarmos um fator de divisão do prescaler de 1:1, teremos uma frequência de 1Mhz na entrada do timer 1, o que significa que a resolução da medição será de 1 $\mu$ s, enquanto a maior largura de sinal que podemos medir (sem intervenção de software) será de 65535  $\mu$ s.

Vejamos então o software necessário para a realização da tarefa:

```

;Captura de Pulso
LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
CAP0 EQU 0x20 ; Registrador de captura LSB
CAP1 EQU 0x21 ; Registrador de captura MSB
ORG 0x0000
GOTO INICIO
ORG 0x0004
SALVA_CONTEXTO ; macro para o salvamento do contexto
BTFS C PIR1,CCP1IF ; testa o flag do módulo CCP1

```

```

GOTO      TRATA_CCP1      ; se for interrupção CCP1IF desvia para TRATA_CCP1
FIM_INT:
    RESTAURA_CONTEXTO      ; restaura o contexto anterior
    RETFIE                  ; retorno da interrupção
TRATA_CCP1:
    MOVF      CCPR1L,W      ; copia o valor LSB capturado para ...
    MOVWF     CAPO          ; ... o registrador CAPO
    MOVF      CCPR1H,W      ; copia o valor MSB capturado para ...
    MOVWF     CAP1          ; ... o registrador CAP1
    CLRF      TMR1L         ; apaga a contagem atual ...
    CLRF      TMR1H         ; ... do timer 1
    BCF       PIR1,CCP1IF    ; apaga flag de interrupção CCP1IF
    GOTO      FIM_INT       ; vai para o fim da interrupção
INICIO:
    CLRF      CCP1CON       ;
    CLRF      TMR1L         ;
    CLRF      TMR1H         ;
    CLRF      INTCON        ;
    CLRF      T1CON         ;
    BSF       STATUS,RP0     ; seleciona o banco 1
    BSF       TRISB,CCP1    ; configura o pino CCP1/RB3 como entrada
    MOVLW    'B'00000100'    ; habilita interrupção ...
    MOVWF     PIE1           ; ... do módulo CCP1
    BCF       STATUS,RP0     ; seleciona o banco 0
    CLRF      PIR1           ; apaga flags de interrupções periféricas
    MOVLW    '0x05            ; configura o módulo CCP1 para ...
    MOVWF     CCP1CON        ; ... modo de captura a cada borda de subida
    BSF       T1CON,TMR1ON   ; ativa contagem do timer 1
    BSF       INTCON,GIE     ; habilita interrupções globais
PRINCIPAL:
    ...
    ...
    ...
END

```

Caso seja necessário medir sinais de largura maiores que 65535µs, podemos lançar mão de um artifício de software

```

;Captura de Pulso
LIST  P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
CAP0      EQU 0x20      ; Registrador de captura LSB
CAP1      EQU 0x21      ; Registrador de captura
CAP2      EQU 0x22      ; Registrador de captura MSB
TMR1HI   EQU 0x23      ; Registrador MSB do timer 1
ORG 0x0000
    GOTO  INICIO
ORG 0x0004
    SALVA_CONTEXTO      ; macro para o salvamento do contexto
    BTFSC    PIR1,CCP1IF  ; testa o flag de interrupção do módulo CCP1
    GOTO    TRATA_CCP1    ; se for interrupção CCP1IF desvia para TRATA_CCP1
    BTFSC    PIR1,TMR1IF  ; testa o flag de interrupção do timer 1
    GOTO    TRATA_TMR1    ; se for interrupção TMR1IF desvia para TRATA_TMR1
FIM_INT:
    RESTAURA_CONTEXTO      ; restaura o contexto anterior
    RETFIE                  ; retorno da interrupção
TRATA_CCP1:

```

```
MOVF CCP1L,W      ; copia o valor LSB capturado para ...
MOVWF CAP0        ; ... o registrador CAP0
MOVF CCP1H,W      ; copia o valor MSB capturado para ...
MOVWF CAP1        ; ... o registrador CAP1
CLRF TMR1L        ; apaga a contagem atual ...
CLRF TMR1H        ; ... do timer 1
BCF PIR1,CCP1IF   ; apaga flag de interrupção CCP1IF
GOTO FIM_INT      ; vai para o fim da interrupção

TRATA_TMR1:
INCF TMR1HI,F    ; incrementa o novo registrador MSB do timer 1
GOTO FIM_INT      ; vai para o fim da interrupção

INICIO:
CLRF CCP1CON      ;
CLRF TMR1L        ;
CLRF TMR1H        ;
CLRF INTCON       ;
CLRF T1CON        ;
CLRF TMR1HI       ;
BSF STATUS,RP0    ; seleciona o banco 1
BSF TRISB,CCP1    ; configura o pino CCP1/RB3 como entrada
MOVLW B'00000100'  ; habilita interrupção ...
MOVWF PIE1         ; ... do módulo CCP1
BCF STATUS,RP0    ; seleciona o banco 0
CLRF PIR1          ; apaga flags de interrupções periféricas
MOVLW 0x05          ; configura o módulo CCP1 para ...
MOVLW CCP1CON     ; ... modo de captura a cada borda de subida
BSF T1CON,TMR1ON  ; ativa contagem do timer 1
BSF INTCON,GIE    ; habilita interrupções globais

PRINCIPAL:
...
...
...
END
```

## 7.11 - Conversor A/D Delta-Sigma

A medição de grandezas analógicas é sem dúvida uma capacidade sempre desejável em um microcontrolador.

O PIC em estudo não possui um conversor A/D interno, mas possui toda a estrutura necessária para a implementação de um conversor analógico / digital utilizando a técnica Delta-Sigma.

Esta técnica utiliza a curva de carga/descarga de uma rede RC para determinar com exatidão o valor de uma tensão analógica externa e apesar de não ser um conversor A/D que se possa chamar de rápido, possui uma velocidade suficiente para a medição de grandezas analógicas de variação lenta e/ou com pequenas taxas de amostragem.

Utilizando conversores Delta-Sigma, é possível obter altas resoluções (da ordem de 20 bits) a um custo muito baixo, quando comparados a outros tipos de conversores.

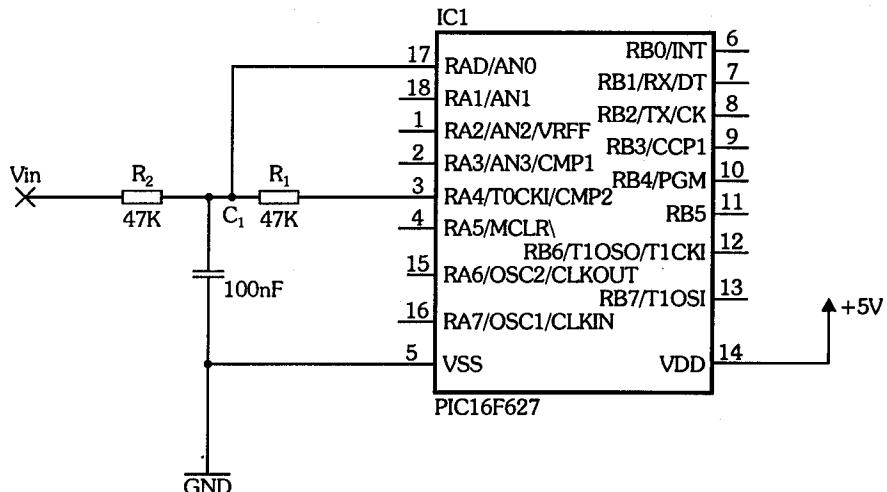
A implementação demonstrada aqui se baseia na nota de aplicação AN700 da Microchip de hardware desse conversor e pode ser vista na figura apresentada em seguida. Observe que são utilizados dois pinos do MCU: um como entrada analógica

(conectado ao comparador interno) e outro como saída, funcionando como um conversor D/A de 1 bit.

O software utiliza dois contadores de 16 bits, sendo um para a contagem do número de ciclos da conversão (4096 ciclos para uma conversão de 12 bits) e o outro para o armazenamento do resultado da conversão.

Na figura 7.2 temos o diagrama da implementação do conversor utilizando o PIC em estudo.

Maiores detalhes sobre o funcionamento dos conversores A/D Delta-Sigma podem ser obtidos na nota de aplicação AN-700 ou em outras literaturas especializadas.



**Figura 7.2**

; Conversor A/D Delta-Sigma (implementação de Dan Butler da Microchip Inc.)  
; Estas sub-rotinas realizam a conversão analógico-digital de um sinal externo.  
; Variáveis utilizadas: RESULTADO\_L, RESULTADO\_H, DELTA\_L, DELTA\_H  
**INICIALIZA\_CONV\_DELTA:**

```
BSF      STATUS, RP0          ; seleciona o banco 1
MOVLW   0xEC
MOVWF   VRCON
BCF     TRISA, 3            ; configura referência de tensão interna
BCF     STATUS, RP0          ; configura o pino RA3 para saída
MOVLW   0x05
MOVWF   CMCON
RETURN
```

**CONVERSOR\_DELTA:**

```
CLRF    RESULTADO_L
CLRF    RESULTADO_H
CLRF    DELTA_L
CLRF    DELTA_H
MOVLW   0x03
MOVWF   CMCON
; inicializa o módulo comparador analógico
```

**CICLO\_CONVERSAO:**

```
BTFSC   CMCON, C1OUT
GOTO    REF_MENOR
; verifica a saída do comparador 1
; se C1OUT=1 vai para REF_MENOR
```

**REF\_MAIOR:**

```
NOP
; gasta 1 ciclo
```

```

BCF      PORTA,3           ; coloca a saída RA3 em nível '1'
INCFSZ   RESULTADO_L,F    ; incrementa o resultado LSB
GOTO     GASTA2CICLOS     ; vai para GASTA2CICLOS
INCF     RESULTADO_H,F    ; incrementa o resultado MSB
GOTO     FINAL_CICLO      ; vai para o final do ciclo
REF_MENOR:
BSF      PORTA,3           ; coloca a saída RA3 em '1'
NOP
GOTO     GASTA2CICLOS     ; gasta 1 ciclo
GASTA2CICLOS:
GOTO     FINAL_CICLO      ; vai para o final do ciclo
FINAL_CICLO:
INCFSZ   DELTA_L,F        ; incrementa o contador de passos de conversão
                           ; LSB
GOTO     GASTA5CICLOS     ; gasta 5 ciclos
INCF     DELTA_H,F        ; incrementa o contador de passos de conversão
                           ; MSB
; o código a seguir determina se a contagem de ciclos ultrapassou 4096
MOVF    DELTA_H,W         ;
ANDLW   B'00010000'         ;
BTFSR   STATUS,Z          ;
GOTO    CICLO_CONVERSAO   ;
GOTO    FIM_CONVERSAO     ;
GASTA5CICLOS:
GOTO    $+1                 ;
NOP
GOTO    CICLO_CONVERSAO   ;
FIM_CONVERSAO:
MOVLW   0x06                ;
MOVWF   CMCON               ;
RETURN

```

## 7.12 - Interface com Displays

### 7.12.1 - Interface de Display 7 Segmentos com 4511

A forma mais simples e rápida de conectar um display LED de 7 segmentos a um microcontrolador é sem dúvida utilizar um CI decodificador externo.

No presente caso, apresentamos um pequeno circuito para demonstrar a forma de conexão entre o PIC e um CI decodificador, no caso um decodificador BCD 4511.

Na figura 7.3 podemos verificar um modo simples de conectar um decodificador 4511 a um PIC, de forma a utilizar um display de 7 segmentos.

É importante ressaltar que para facilitar a tarefa de programação, é interessante fazer com que as linhas de entrada de dados do decodificador (A, B, C e D) sejam conectadas às linhas 0, 1, 2 e 3 (no caso RB0, RB1, RB2 e RB3). Isto porque ao adotar esta aproximação, o programador pode simplesmente escrever no nibble inferior do registrador PORTB para fazer com que o respectivo número seja apresentado no display.

Observe ainda que o CI 4511 é um decodificador BCD, isto é, ele somente é capaz de apresentar no display dígitos entre 0 e 9. Se tentarmos apresentar um dígito hexadecimal, o display permanece apagado.

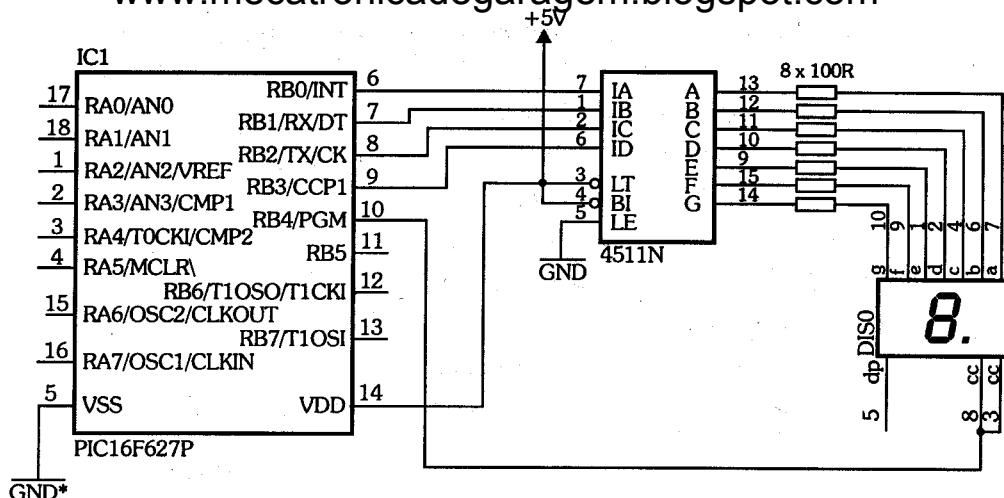


Figura 7.3

Para fazer com que o display seja efetivamente ativado, é necessário colocar o pino RB4 em nível lógico '0'. Isto porque o display utilizado é do tipo catodo comum e o pino RB4 está ligado ao catodo do display.

Dependendo do tipo do display utilizado, pode ser necessário utilizar um transistor (ou CI driver) para a função de ativação/desativação do catodo do display.

Vejamos agora um pequeno programa para apresentar o número 5 no display:

```
; teste de display com 4511
LIST P=16F627
INCLUDE <P16F627.INC>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
ORG 0x0000
; primeiramente vamos inicializar as portas como saídas
    MOVLW B'00000111'
    MOVWF CMCON           ; configura os comparadores analógicos para modo 7
    BANKSEL TRISA
    CLRF TRISA
    CLRF TRISB
    BANKSEL PORTB
; agora que as portas estão devidamente configuradas vamos ao display
    MOVLW D'5'             ; copia em W o número 5
    MOVWF PORTB            ; o número 5 é enviado para o 4511
    BSF    PORTB, 4          ; ativa o display
    SLEEP                  ; o programa termina aqui
END
```

Observe que este programa poderia ter copiado diretamente o valor 21 decimal (equivalente ao binário 00010101) para o registrador PORTB. Isto faria com que fosse desnecessária a instrução BSF PORTB,4. No entanto, o programa foi mantido como está por uma questão de didática.

A utilização de um CI decodificador de display apresenta vantagens e desvantagens:

- ◆ Como vantagem podemos destacar:
  1. Economia de pinos do PIC dedicados ao display;
  2. Facilidade de programação.
- ◆ Como desvantagens podemos destacar:
  1. Aumento do número de componentes;
  2. Aumento do custo;
  3. Aumento do consumo de corrente;
  4. Menor flexibilidade de operação do display (tendo em vista a limitação do decodificador utilizado).

Finalmente, um último comentário: o circuito descrito anteriormente representa apenas uma das formas de conectar um CI 4511 a um PIC. Existem diversas outras formas de implementação, que podem inclusive agregar outras funções (como fazer com que o PIC controle os pinos LE, LT e BI do 4511).

### 7.12.2 - Interface de Display 7 Segmentos Decodificada por Software

A seguir, temos uma outra forma de utilizar displays de 7 segmentos com o PIC, mas desta vez sem o auxílio de um circuito integrado decodificador de display.

A ausência do CI decodificador implica na necessidade de conexão de um pino de I/O do PIC para cada segmento do display. Desta forma, vamos necessitar inicialmente de sete pinos de I/O para cumprir a tarefa.

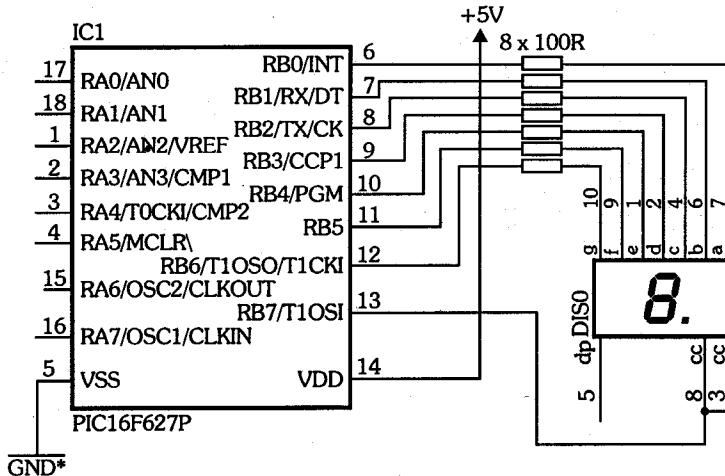


Figura 7.4

Conforme vemos nesta figura, optou-se por conectar o segmento 'a' do display ao pino RB0, o segmento 'b' ao pino RB1 e assim por diante até que o segmento 'f' foi conectado ao pino RB6.

Novamente, o cátodo do display foi conectado a um pino de I/O do PIC (no caso o pino RB7).

O hardware neste caso, é evidentemente mais simples que o circuito anterior, devido à ausência do decodificador, mas a tarefa de decodificação do display ficará agora a cargo do software em execução no PIC.

Vejamos o programa a ser utilizado para cumprir a tarefa descrita:

```
; teste de display decodificado por software
LIST P=16F627
INCLUDE <P16F627.INC>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF

DESLOCAMENTO EQU 0x20
ORG 0x0000

; primeiramente vamos inicializar as portas como saídas
MOVLW B'00000111'
MOVWF CMCON ; configura os comparadores analógicos para modo 7
BANKSEL TRISA
CLRF TRISA
CLRF TRISB
BANKSEL PORTB

; agora que as portas estão devidamente configuradas vamos ao display
MOVLW D'5' ; copia em W o número 5
CALL DECOD_DISPLAY ; chama a sub-rotina para decodificar o display
MOVWF PORTB ; o número 5 é enviado para o 4511
BSF PORTB, 4 ; ativa o display
SLEEP ; o programa termina aqui

DECOD_DISPLAY:
    MOVWF DESLOCAMENTO ; armazena o número na variável deslocamento
    MOVLW LOW TABELA ; copia em W os 8 bits LSB do endereço da TABELA
    ADDWF DESLOCAMENTO, F ; adiciona o valor à variável DESLOCAMENTO
    MOVLW HIGH TABELA ; copia em W os 5 bits superiores do endereço da
TABELA
    BTFSC STATUS, C ; testa para ver se a soma anterior transbordou
    ADDLW 0x01 ; se transbordou, soma 1 ao W
    MOVWF PCLATH ; acerta o PCLATH de acordo com o endereço da
; TABELA
    MOVF DESLOCAMENTO, W ; copia o valor do DESLOCAMENTO para o W
    MOVWF PCL ; copia o W para o PCL (desvia para a tabela)

TABELA:
    RETLW B'00111111' ; número 0
    RETLW B'00000110' ; número 1
    RETLW B'01011011' ; número 2
    RETLW B'01001111' ; número 3
    RETLW B'01100110' ; número 4
    RETLW B'01101101' ; número 5
    RETLW B'01111101' ; número 6
    RETLW B'00000111' ; número 7
    RETLW B'01111111' ; número 8
    RETLW B'01100111' ; número 9
    RETLW B'01110111' ; dígito A
```

```

RETLW  B'01111100' ; dígito B
RETLW  B'00111001' ; dígito C
RETLW  B'01011110' ; dígito D
RETLW  B'01111001' ; dígito E
RETLW  B'01110001' ; dígito F

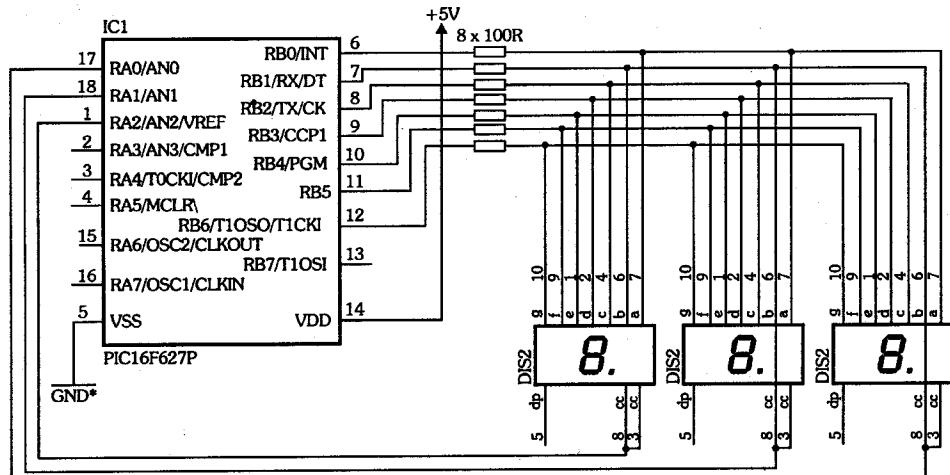
```

END

Observe que a decodificação de display por software também apresenta vantagens e desvantagens:

- ◆ Como vantagens podemos destacar:
  1. Maior flexibilidade de apresentação do display: a decodificação por software permite ao programador um controle preciso do que é apresentado no display. Isto permite a apresentação de símbolos, letras e quaisquer outras formas possíveis de ser representadas em 7 segmentos;
  2. Menor custo: devido à ausência do CI decodificador;
  3. Menor consumo: a ausência do decodificador aliada ao modo SLEEP do PIC permite um controle preciso sobre o consumo do circuito.
- ◆ Como desvantagens podemos citar:
  1. Aumento da complexidade do programa: as instruções adicionais para decodificação do display provocam um pequeno aumento no tamanho do programa;
  2. Aumento do tempo de execução: o tempo de execução do programa sofre um sensível aumento devido às instruções de decodificação do display;
  3. Utilização de muitos pinos de I/O: podemos dizer que um display decodificado por software irá utilizar, no mínimo, sete pinos de I/O para sua operação.

### 7.12.3 - Display Multiplexado de 7 Segmentos



*Figura 7.5*

A multiplexação de displays é uma técnica utilizada para minimizar o uso de pinos de I/O para operações com displays de 7 segmentos e baseia-se em uma característica do olho humano chamada de persistência retiniana.

A persistência retiniana é uma propriedade das células da retina do olho que faz com que cada célula atue como se fosse um pequeno capacitor. Ao haver a exposição à luz, a célula emite um sinal elétrico e continuará a emitir esse sinal por mais alguns milissegundos após o término da exposição. Este efeito faz com que o olho "memorize" uma imagem durante alguns instantes de tempo.

Esta característica da retina é utilizada para criar a ilusão de movimento em equipamentos como projetores de cinema, televisores, etc.

A utilização da persistência retiniana na multiplexação de displays funciona da seguinte forma: cada display é ativado por um certo tempo e em seguida desativado em seqüência, um após o outro. A repetição rápida deste processo causa a ilusão ao olho humano de que todos os displays estão acesos simultaneamente.

Se considerarmos o circuito da figura 7.5, utilizando uma implementação convencional (sem multiplexação), necessitariam de  $7 \times 3 + 3 = 24$  linhas de I/O para os três displays. No entanto, com o uso da multiplexação, podemos utilizar apenas  $7 + 3 = 10$  linhas de I/O para a mesma operação.

É claro que a contrapartida da multiplexação é que o programador deve providenciar o software necessário para efetuar todo o controle de multiplexação envolvido.

Em seguida temos um exemplo de software para realizar a multiplexação de displays do circuito da figura 7.5.

### Observação

Se for desejado um maior brilho dos displays, será necessário utilizar transistores ou drivers para realizar o controle dos catodos dos displays. A solução aqui apresentada visa demonstrar uma alternativa tão econômica quanto possível.

```
;Display multiplexado
LIST P=16F627
INCLUDE <P16F627.INC>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
DESLOCAMENTO EQU 0x20
D0 EQU 0x21
D1 EQU 0x22
D2 EQU 0x23
ORG 0x0000
GOTO INICIO
DECOD_DISPLAY:
    MOVWF DESLOCAMENTO ; armazena o número na variável deslocamento
    MOVLW LOW TABELA ; copia em W os 8 bits LSB do endereço da TABELA
    ADDWF DESLOCAMENTO, F ; adiciona o valor à variável DESLOCAMENTO
    MOVLW HIGH TABELA ; copia em W os 5 bits superiores do endereço da
TABELA
```

```

BTFSC STATUS,C      ; testa para ver se a soma anterior transbordou
ADDLW 0x01          ; se transbordou, soma 1 ao W
MOVWF PCLATH        ; acerta o PCLATH de acordo com o endereço da
                      ; TABELA
MOVF  DESLOCAMENTO,W ; copia o valor do DESLOCAMENTO para o W
MOVWF PCL            ; copia o W para o PCL (desvia para a tabela)

```

TABELA:

```

RETLW B'00111111'   ; número 0
RETLW B'00000110'   ; número 1
RETLW B'01011011'   ; número 2
RETLW B'01001111'   ; número 3
RETLW B'011000110'  ; número 4
RETLW B'01101101'   ; número 5
RETLW B'01111101'   ; número 6
RETLW B'000000111'  ; número 7
RETLW B'01111111'   ; número 8
RETLW B'011000111'  ; número 9
RETLW B'01110111'   ; dígito A
RETLW B'01111100'   ; dígito B
RETLW B'00111001'   ; dígito C
RETLW B'01011110'   ; dígito D
RETLW B'01111001'   ; dígito E
RETLW B'01110001'   ; dígito F

```

ATRASO:

```

ADDLW 1             ; Sub-rotina de atraso
BTFSC STATUS,Z     ; soma 1 ao W
RETURN             ; testa se W=0 ...
                  ; se W=0, retorna
GOTO   ATRASO       ; se W diferente de 0, desvia para ATRASO_LOOP

```

DISPLAY:

```

MOVF  D0,W          ; copia o dígito em W
CALL  DECOD_DISPLAY ; decodifica o dígito
MOVWF PORTB         ; ativa os segmentos do display
BCF   PORTA,2        ; ativa display 3
CLRW
CALL  ATRASO         ; W=0
                  ; espera
BSF   PORTA,2        ; desativa display 3
MOVF  D1,W          ; copia o próximo dígito em W
CALL  DECOD_DISPLAY ; decodifica o display
MOVWF PORTB         ; ativa os segmentos
BCF   PORTA,1        ; ativa display 2
CLRW
CALL  ATRASO         ; W=0
                  ; espera
BSF   PORTA,1        ; desativa display 2
MOVF  D2,W          ; copia o último dígito em W
CALL  DECOD_DISPLAY ; decodifica display
MOVWF PORTB         ; ativa os segmentos
BCF   PORTA,0        ; ativa o display 1
CLRW
CALL  ATRASO         ; W=0
                  ; espera
BSF   PORTA,0        ; desativa o display 1
RETURN             ; retorna

```

INICIO:

; primeiramente vamos inicializar as portas como saídas

```

MOVWL B'00000111'    ; configura os comparadores analógicos para modo 7
MOVWF CMCON           ;
BANKSEL TRISA          ; configura a porta A para saídas
CLRF   TRISA           ;
CLRF   TRISB           ; configura a porta B para saídas
BANKSEL PORTB          ;

```

```

MOVlw  B'00000111' ; desliga os catodos dos três ...
MOVwf  PORTA       ; ... displays
MOVlw  D'3'        ; display 1=3
MOVwf  D2          ;
MOVlw  D'2'        ; display 2=2
MOVwf  D1          ;
MOVlw  D'1'        ; display 3=1
MOVwf  D0          ;

LOOP_PRINCIPAL:
    CALL   DISPLAY      ; mostra display
    GOTO  LOOP_PRINCIPAL;
END

```

## 7.13 - Leitura de Teclado

Outros dispositivos normalmente utilizados para interação entre o mundo exterior e os microcontroladores são as teclas que podem ser utilizadas para uma infinidade de aplicações: início/fim de processos, programação, alteração de parâmetros, sensoriamento, etc.

As técnicas para a interface entre as teclas e o microcontrolador variam de acordo com diversos parâmetros, mas principalmente em função do número de teclas.

Outro aspecto importante quando da leitura de teclas reside no fato de que uma tecla, ao ser pressionada, gera um certo ruído e muitas vezes esse ruído é captado pelo software da aplicação, resultando em que uma única pressão da tecla gera uma grande quantidade de eventos de pressão de tecla.

Para contornar este problema, utilizam-se técnicas de software para a filtragem desses ruídos indesejados, as quais são conhecidas em inglês como DEBOUNCE.

Vejamos agora uma forma de realizar a leitura de um conjunto de três teclas sem a filtragem de ruído:

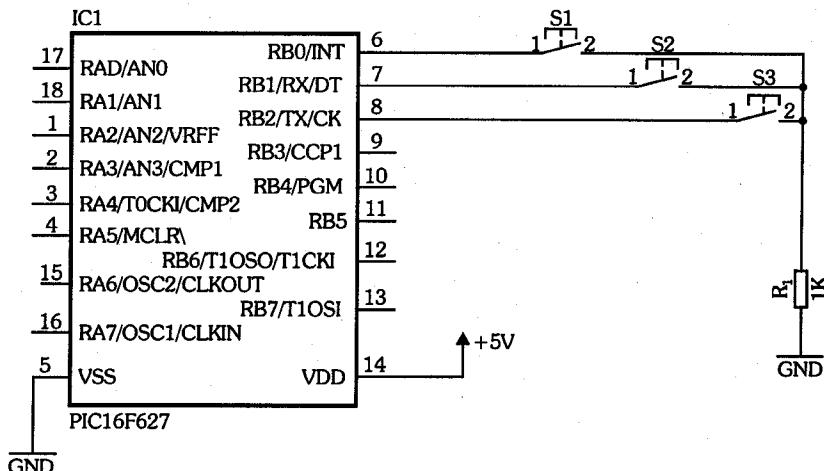


Figura 7.6

; os resistores de PULL-UP internos da porta B devem estar ativados  
OPTION\_REG:RBU='1'

...

**TRATA\_TECLAS:**

```

    BTFSS  PORTB,0      ; verifica se a tecla S1 está pressionada
    CALL   TECLA_S1
    BTFSS  PORTB,1      ; verifica se a tecla S2 está pressionada
    CALL   TECLA_S2
    BTFSS  PORTB,2      ; verifica se a tecla S3 está pressionada
    CALL   TECLA_S3
    RETURN
TECLA_S1:
    ...
    ...
    BTFSS  PORTB,0      ; verifica se a tecla S1 foi liberada
    GOTO   $-1
    RETURN
TECLA_S2:
    ...
    ...
    BTFSS  PORTB,1      ; verifica se a tecla S2 foi liberada
    GOTO   $-1
    RETURN
TECLA_S3:
    ...
    ...
    BTFSS  PORTB,2      ; verifica se a tecla S3 foi liberada
    GOTO   $-1
    RETURN

```

O mesmo circuito pode ser utilizado com o software seguinte que implementa a filtragem de ruído das teclas:

; esta sub-rotina utiliza uma variável chamada BUFFER que deve ser definida  
; previamente os resistores de PULL-UP internos da porta B devem estar ativados  
OPTION\_REG:RBU='1'

...

...

**TRATA\_TECLAS:**

```

    MOVF   PORTB,W      ; copia o conteúdo da porta B para o W (lê as teclas)
    MOVWF  BUFFER
    CLRW
    CALL   ATRASO
    MOVF   PORTB,W
    XORWF  BUFFER,W
    BTFSC  STATUS,Z
    RETURN
    BTFSS  PORTB,0      ; verifica se a tecla S1 está pressionada
    CALL   TECLA_S1
    BTFSS  PORTB,1      ; verifica se a tecla S2 está pressionada
    CALL   TECLA_S2
    BTFSS  PORTB,2      ; verifica se a tecla S3 está pressionada
    CALL   TECLA_S3
    RETURN
TECLA_S1:
    ...
    ...
    ...
    BTFSS  PORTB,0      ; verifica se a tecla S1 foi liberada
    GOTO   $-1
    RETURN

```

```

    RETURN ; retorna
TECLA_S2:
    ...
    ...
    ...
    BTFFS PORTB,1      ; verifica se a tecla S2 foi liberada
    GOTO $-1           ; se não foi, volta para instrução anterior
    RETURN             ; retorna
TECLA_S3:
    ...
    ...
    ...
    BTFFS PORTB,2      ; verifica se a tecla S3 foi liberada
    GOTO $-1           ; se não foi, volta para instrução anterior
    RETURN             ; retorna
ATRASO:
    ADDLW 1            ; Sub-rotina de atraso
    BTFSC STATUS,Z     ; soma 1 ao W
    RETURN             ; testa se W=0 ...
    GOTO ATRASO         ; se W=0, retorna
                        ; se W diferente de 0, desvia para ATRASO_LOOP

```

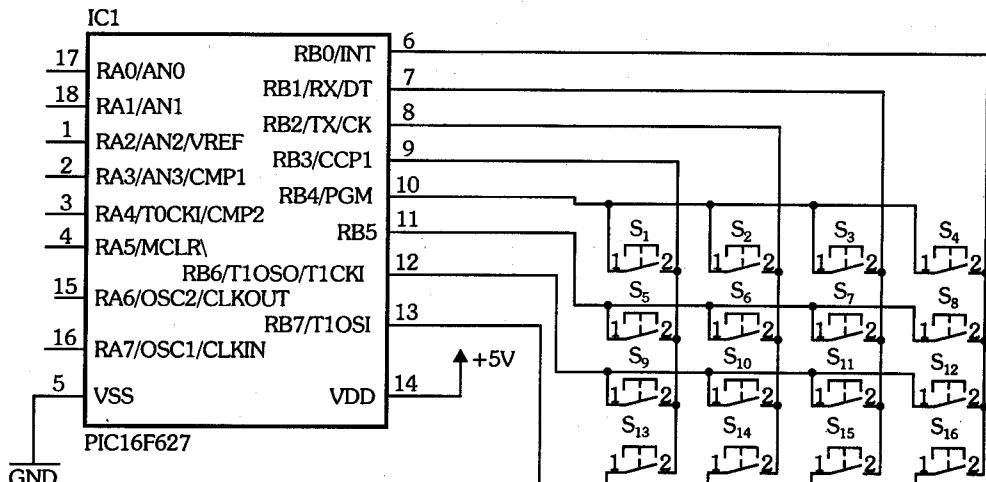
Em teclados com um número de teclas superior a oito, utiliza-se normalmente a técnica de varredura de matriz. Nessa técnica as teclas componentes do teclado são arranjadas para formar uma matriz com linhas e colunas.

A leitura do teclado é feita ativando e desativando uma coluna a cada vez.

Desta forma, a quantidade de I/Os necessária para a conexão do teclado é substancialmente reduzida.

O exemplo da figura 7.7 reflete bem esta situação. Temos um diagrama da conexão de um conjunto de 16 teclas utilizando apenas oito pinos de I/O.

Sem a utilização da varredura em matriz, seria necessário utilizar 16 I/Os para a mesma operação!



*Figura 7.7*

```

; TRATA_TECLAS - esta sub-rotina realiza a varredura do teclado a procura de
; uma tecla pressionada. Caso seja encontrada, retorna o valor
; da tecla na variável NOVA_TECLA.
; A variável TECLA_FLAGS é utilizada como um registrador de
; FLAGS, armazenando diversos FLAGS de estado do teclado:
;   - FILTRO:
;   - PRESSIONADA:
;   - TECLA:
; Uma outra sub-rotina no decorrer do programa deverá
; providenciar a leitura da tecla (ler a variável NOVA_TECLA) e
; zerar o FLAG TECLA.
; ****
; Os pinos RB0,RB1, RB2 e RB3 devem estar configurados como saídas
; Os pinos RB4, RB5, RB6 e RB7 devem estar configurados como entradas
; Variáveis: TECLA_FLAGS, VARREDURA, NOVA_TECLA, TEMP2, TEMP3, TEMPO_TECLA
;

TRATA_TECLAS:
; DEFINIÇÕES
FILTRO EQU 0
PRESSIONADA EQU 1
TECLA EQU 2
;
BTFS S TECLA_FLAGS, FILTRO ; testa se o filtro de DEBOUNCE está ativo
GOTO VARRE_TECLAS ; se não está, varre as teclas
DECFSZ TEMPO_TECLA, F ; se o filtro estiver ativo, decrementa
; tempo
RETURN ; retorna
BCF TECLA_FLAGS, FILTRO ; limpa o filtro de DEBOUNCE
RETURN ; retorna
VARRE_TECLAS
MOVLW B'11101111' ; prepara a varredura da matriz do teclado
MOVWF TEMP2
;
VARRE_PROXIMA:
MOVF PORTB,W ; lê a porta B
BCF INTCON, RBIF ; limpa o flag de mudança da porta B
RRF TEMP2, F ; rotaciona o índice de varredura de teclado
BTFS S STATUS, C ; testa se a varredura chegou ao final
GOTO NENHUMA_TECLA ; se chegou, nenhuma tecla foi pressionada
MOVF TEMP2, W ; se não copia o índice de varredura para W
MOVWF PORTB ; ... e então para a porta B
NOP ; aguarda ...
NOP ; ... dois ciclos
BTFS S INTCON, RBIF ; verifica se houve mudanças na porta B
GOTO VARRE_PROXIMA ; se não houve, varre a próxima coluna
BTFS C TECLA_FLAGS, PRESSIONADA ; se houve, verifica FLAG da tecla
; pressionada
RETURN ; se a tecla já estava pressionada, retorna
BSF TECLA_FLAGS, PRESSIONADA ; se não, ativa o flag de tecla
; pressionada
SWAPF PORTB, W ; inverte nibbles da porta B e coloca em W
MOVWF TEMP3 ; copia para variável temporária (TEMP3)
CALL PEGA_TECLA ; verifica qual a tecla pressionada
MOVWF NOVA_TECLA ; copia o código para a NOVA_TECLA
BSF TECLA_FLAGS, TECLA ; ativa flag de nova tecla pressionada
BSF TECLA_FLAGS, FILTRO ; ativa filtro de DEBOUNCE
MOVLW 0x04 ; reinicia o valor do temporizador
MOVWF TEMPO_TECLA ; de DEBOUNCE
NENHUMA_TECLA:
BCF TECLA_FLAGS, PRESSIONADA ; se nenhuma tecla foi pressionada, limpa
RETURN ; limpa o flag de tecla pressionada e
; retorna

```

```

PEGA_TECLA:
    CLRF    TEMP1      ; limpa tecla temporária (TEMP1)
    BTFS S TEMP2,3   ; verifica se é a primeira coluna
    GOTO    LINHA     ; sim ? então pula para o teste de linhas
    INCF    TEMP1,F   ; não ? então incrementa TEMP1
    BTFS S TEMP2,2   ; testa se é a segunda coluna
    GOTO    LINHA     ; sim ? então pula para o teste de linhas
    INCF    TEMP1,F   ; não ? então incrementa TEMP1
    BTFS S TEMP2,1   ; testa se é a terceira coluna
    GOTO    LINHA     ; sim ? então pula para o teste de linhas
    INCF    TEMP1,F   ; não ? então incrementa TEMP1
    ; então é a quarta coluna ...

LINHA:
    BTFS S TEMP3,0   ; verifica se foi a primeira linha
    GOTO    TABELA_TECLA ; sim ? então é uma das teclas: S1,S2,S3 ou S4
    BTFS S TEMP3,1   ; verifica se foi a segunda linha
    GOTO    TECLAS5678 ; sim ? então é uma das teclas: S5,S6,S7 ou S8
    BTFS S TEMP3,2   ; verifica se foi a terceira linha
    GOTO    TECLAS9_10_11_12 ; sim ? então é uma das teclas: S9,S10,S11,S12

TECLAS13_14_15_16:
    BSF    TEMP1,2    ; seta o bit 2 do índice da tabela
TECLAS9_10_11_12:
    BSF    TEMP1,3    ; seta o bit 3 do índice da tabela
    GOTO    TABELA_TECLA ; vai para a tabela
TECLAS5678:
    BSF    TEMP1,2    ; seta apenas o bit 2 do índice da tabela
TABELA_TECLA:
    MOVEF  TEMP1,W    ; copia o índice da tabela para W
    ADDW   PCL,F      ; adiciona W ao PCL
    RETLW  0x01        ; retorna valor da tecla S1
    RETLW  0x02        ; retorna valor da tecla S2
    RETLW  0x03        ; retorna valor da tecla S3
    RETLW  0x04        ; retorna valor da tecla S4
    RETLW  0x05        ; retorna valor da tecla S5
    RETLW  0x06        ; retorna valor da tecla S6
    RETLW  0x07        ; retorna valor da tecla S7
    RETLW  0x08        ; retorna valor da tecla S8
    RETLW  0x09        ; retorna valor da tecla S9
    RETLW  0x0A        ; retorna valor da tecla S10
    RETLW  0x0B        ; retorna valor da tecla S11
    RETLW  0x0C        ; retorna valor da tecla S12
    RETLW  0x0D        ; retorna valor da tecla S13
    RETLW  0x0E        ; retorna valor da tecla S14
    RETLW  0x0F        ; retorna valor da tecla S15

```

A tabela final encontrada na listagem anterior é responsável por gerar o código correspondente a cada tecla pressionada.

Alterando o valor retornado por cada instrução RETLW, o programador pode adequar a sub-rotina às suas necessidades.

## 7.14 - Detector de Paridade

Em seguida apresentamos um simples detector de paridade que pode ser utilizado em aplicações de transmissão de dados ou checagem de erros simples.

Ele calcula a quantidade de bits '1' existentes no valor passado em W e retorna no próprio W o valor da paridade: 0 se for par e 1 se for ímpar.

; Esta sub-rotina verifica a paridade de um número fornecido em W e retorna em  
; W o valor 0 se a paridade é par ou 1 se a paridade é ímpar.  
; O programa utiliza uma variável local: VTEMP.

PARIDADE:

MOVWF	VTEMP	; copia o valor a ser verificado para VTEMP
CLRW		; apaga W
BCF	STATUS,C	; limpa o flag de carry
RRF	VTEMP,F	; rotaciona VTEMP à direita
BTFS C	STATUS,C	; verifica o bit que transbordou do VTEMP (bit 0)
ADDLW	D'1'	; se for igual a '1', soma um ao W
RRF	VTEMP,F	; rotaciona VTEMP à direita
BTFS C	STATUS,C	; verifica o bit que transbordou do VTEMP (bit 1)
ADDLW	D'1'	; se for igual a '1', soma um ao W
RRF	VTEMP,F	; rotaciona VTEMP à direita
BTFS C	STATUS,C	; verifica o bit que transbordou do VTEMP (bit 2)
ADDLW	D'1'	; se for igual a '1', soma um ao W
RRF	VTEMP,F	; rotaciona VTEMP à direita
BTFS C	STATUS,C	; verifica o bit que transbordou do VTEMP (bit 3)
ADDLW	D'1'	; se for igual a '1', soma um ao W
RRF	VTEMP,F	; rotaciona VTEMP à direita
BTFS C	STATUS,C	; verifica o bit que transbordou do VTEMP (bit 4)
ADDLW	D'1'	; se for igual a '1', soma um ao W
RRF	VTEMP,F	; rotaciona VTEMP à direita
BTFS C	STATUS,C	; verifica o bit que transbordou do VTEMP (bit 5)
ADDLW	D'1'	; se for igual a '1', soma um ao W
RRF	VTEMP,F	; rotaciona VTEMP à direita
BTFS C	STATUS,C	; verifica o bit que transbordou do VTEMP (bit 6)
ADDLW	D'1'	; se for igual a '1', soma um ao W
RRF	VTEMP,F	; rotaciona VTEMP à direita
BTFS C	STATUS,C	; verifica o bit que transbordou do VTEMP (bit 7)
ADDLW	D'1'	; se for igual a '1', soma um ao W
ANDLW	0x01	; apaga todos os bits menos o bit 0 do W
RETURN		; retorna

## 7.15 - Expandindo o Conjunto de Instruções PIC

A seguir, apresentamos um conjunto de macroinstruções que podem ser muito úteis no dia-a-dia do programador. Estas macroinstruções estão inclusas no arquivo MACROS.ASM que pode ser baixado do site da Editora Érica.

### 7.15.1 - Operações com o Registrador W

#### INCW - Incremento de W

Formato: INCW

A macroinstrução INCW é utilizada para incrementar o registrador W em 1.

Observe que os bits C, DC e Z são alterados por esta macroinstrução:

```
INCW MACRO  
ADDLW 0x01  
ENDM
```

### ***Exemplo***

```
MOVLW 0x01 ; W=0x01  
INCW ; incrementa W. W=0x02. Z='0', DC='0' e C='0'  
...
```

## **DECW - Decremento de W**

Formato: DECW

Esta macroinstrução é utilizada para decrementar o valor do registrador W em 1.

Observe que os bits C, DC e Z são alterados por esta macroinstrução:

```
DECW MACRO  
ADDLW 0xFF  
ENDM
```

### ***Exemplo***

```
MOVLW 0x01 ; W=0x01  
DECW ; decrementa W. W=0x00. Z='1', DC='0' e C='0'  
...
```

## **COMW - Complementa W**

Formato: COMW

A macroinstrução COMW complementa o conteúdo do registrador W.

Esta macroinstrução altera o flag Z.

```
COMW MACRO  
XORWF 0xFF  
ENDM
```

### ***Exemplo***

```
MOVLW 0x0F ; W=0x0F  
COMW ; complementa W. W=0xF0. Z='0'  
...
```

## **BSW - Seta bit de W**

Formato: BSW <bit>

Esta macroinstrução é utilizada para setar o bit especificado pelo operando <bit> do registrador W.

Esta macroinstrução altera o flag Z.

```
BSW MACRO BIT
IF (BIT==0)
    IORLW B'00000001'
ENDIF
IF (BIT==1)
    IORLW B'00000010'
ENDIF
IF (BIT==2)
    IORLW B'00000100'
ENDIF
IF (BIT==3)
    IORLW B'00001000'
ENDIF
IF (BIT==4)
    IORLW B'00010000'
ENDIF
IF (BIT==5)
    IORLW B'00100000'
ENDIF
IF (BIT==6)
    IORLW B'01000000'
ENDIF
IF (BIT==7)
    IORLW B'10000000'
ENDIF
ENDM
```

### **Exemplo**

```
CLRW ; W=0x00. Z=1
BSW 2 ; seta o bit 2 de W. W=D'4' ou 00000100 em binário. Z=0
***
```

### **BCW - Apaga bit do registrador W**

Formato: BCW <bit>

Esta macroinstrução é utilizada para apagar o bit especificado pelo operando <bit> do registrador W.

Esta macroinstrução altera o flag Z.

```
BCW MACRO BIT
IF (BIT==0)
    ANDLW B'11111110'
ENDIF
IF (BIT==1)
    ANDLW B'11111101'
ENDIF
IF (BIT==2)
    ANDLW B'11111011'
ENDIF
IF (BIT==3)
```

```
        ANDLW    B'11110111'
ENDIF
IF (BIT==4)
    ANDLW    B'11101111'
ENDIF
IF (BIT==5)
    ANDLW    B'11011111'
ENDIF
IF (BIT==6)
    ANDLW    B'10111111'
ENDIF
IF (BIT==7)
    ANDLW    B'01111111'
ENDIF
ENDM
```

### ***Exemplo***

```
MOVLW    0xFF      ; W=0xFF
BCW      0          ; apaga o bit 0 de W. W=0xFE ou 11111110 em binário. Z=0
...
```

## **7.15.2 - Desvio Condisional**

### **JPE - Desvia se o registrador é igual a**

Formato: JPE <endereço>,<registrador>,<valor>

A macroinstrução JPE desvia o fluxo do programa para <endereço> caso o conteúdo do registrador especificado por <registraror> seja igual a <valor>.

Esta macroinstrução altera os flags Z, DC e C e também o registrador W.

```
JPE  MACRO    ENDER, REG, VAL
    MOVLW    VAL
    SUBWF    REG, W
    BTFSC   STATUS, Z
    GOTO    ENDER
ENDM
```

### ***Exemplo***

```
...
MOVLW    0x05      ; W=0x05
MOVWF    TESTE     ; TESTE=0x05
JPE     PONTOA, TESTE, 0x05 ; testa se TESTE=0x05 e pula para o
                           ; PONTOA se igual
...
PONTOA:                         ; o programa desviou para cá
...
```

### **JPNE - Desvia se o registrador é diferente de**

Formato: JPNE <endereço>,<registraror>,<valor>

A macroinstrução JPNE desvia o fluxo do programa para <endereço> caso o conteúdo do registrador especificado por <registrator> seja diferente de <valor>.

Esta macroinstrução altera os flags Z, DC e C e também o registrador W.

```
JPNE MACRO ENDER, REG, VAL
    MOVLW VAL
    SUBWF REG, W
    BTFSS STATUS, Z
    GOTO ENDER
ENDM
```

### **Exemplo**

```
...
MOVlw 0x04 ; W=0x04
MOVwf TESTE ; TESTE=0x04
JPNE PONTOA,TESTE,0x05 ; testa se TESTE é diferente de
... ; 0x05 e pula para o PONTOA se diferentes
... ; rentes
PONTOA: ; o programa desviou para cá
...
```

### **JPL - Desvia se o registrador é menor que**

Formato: JPL <endereço>,<registrator>,<valor>

A macroinstrução JPL desvia o fluxo do programa para <endereço> caso o conteúdo do registrador especificado por <registrator> seja menor que <valor>.

Esta macroinstrução altera os flags Z, DC e C e também o registrador W.

```
JPL MACRO ENDER, REG, VAL
    SUBLW VAL
    BTFSS STATUS, C
    GOTO ENDER
ENDM
```

### **Exemplo**

```
...
MOVlw 0x05 ; W=0x05
MOVwf TESTE ; TESTE=0x05
JPL PONTOA,TESTE,0x06 ; testa se TESTE é menor que 0x06
... ; e desvia para PONTOA caso verdadeiro
PONTOA: ; o programa desviou para cá
...
```

### **JPLS - Desvia se o registrador é menor ou igual a**

Formato: JPLS <endereço>,<registrator>,<valor>

A macroinstrução JPLS desvia o fluxo do programa para <endereço> caso o conteúdo do registrador especificado por <registrator> seja menor ou igual a <valor>.

JPLS	MACRO	ENDER, REG, VAL
SUBLW		VAL
BTFSS		STATUS, C
GOTO		ENDER
BTFSC		STATUS, Z
GOTO		ENDER
ENDM		

### ***Exemplo***

```
...
MOVLW 0x05          ; W=0x05
MOVWF TESTE         ; TESTE=0x05
JPLS  PONTOA,TESTE,0x05   ; testa se TESTE é < OU = a 0x05
...                   ; e desvia para PONTOA caso verdadeiro
PONTOA:             ; o programa desviou para cá
...
```

### **JPG - Desvia se o registrador é maior que**

Formato: JPG <endereço>,<registraror>,<valor>

A macroinstrução JPG desvia o fluxo do programa para <endereço> caso o conteúdo do registrador especificado por <registraror> seja maior que <valor>.

Esta macroinstrução altera os flags Z, DC e C e também o registrador W.

JPG	MACRO	ENDER, REG, VAL
SUBLW		VAL
BTFSC		STATUS, C
GOTO		JPG_FIM
BTFSS		STATUS, Z
GOTO		ENDER
JPG_FIM:		
ENDM		

### ***Exemplo***

```
...
MOVLW 0x05          ; W=0x05
MOVWF TESTE         ; TESTE=0x05
JPG  PONTOA,TESTE,0x06   ; testa se TESTE é < OU = a 0x05
...                   ; e desvia para PONTOA caso verdadeiro
...                   ; no caso, não ocorre o desvio
...
PONTOA:             ;
```

### **JPGS - Desvia se o registrador é maior ou igual a**

Formato: JPGS <endereço>,<registraror>,<valor>

A macroinstrução JPGS desvia o fluxo do programa para <endereço> caso o conteúdo do registrador especificado por <registraror> seja menor ou igual a <valor>.

Esta macroinstrução altera os flags Z, DC e C e também o registrador W.

```
JPGS MACRO    ENDER, REG, VAL
      SUBLW    VAL
      BTFSC    STATUS, C
      GOTO     ENDER
ENDM
```

### **Exemplo**

```
...
MOVLW 0x05          ; W=0x05
MOVWF TESTE         ; TESTE=0x05
JPL    PONTOA,TESTE,0x05 ; testa se TESTE é < OU = a 0x05
...                   ; e desvia para PONTOA caso verdadeiro
PONTOA:              ; o programa desviou para cá
...
```

## **JPWZ - Desvia se W é igual a zero**

Formato: JPWZ <endereço>

A macroinstrução JPWZ desvia para <endereço> caso o conteúdo do registrador W seja igual a zero.

Esta macroinstrução altera o flag Z.

```
JPWZ MACRO    ENDER
      ANDLW   0XFF
      BTFSC   STATUS, Z
      GOTO     ENDER
ENDM
```

### **Exemplo**

```
...
MOVLW 0x05      ; W=0x04
JPWZ  PONTOB    ; testa se W igual a zero e desvia
...             ; para o PONTOB
PONTOB:          ;
...
```

No exemplo acima, o programa não desviou pois W era diferente de zero

## **JPWNZ - Desvia se W é diferente de zero**

Formato: JPWNZ <endereço>

A macroinstrução JPWNZ desvia para <endereço> caso o conteúdo do registrador W seja diferente de zero.

Esta macroinstrução altera o flag Z.

```
JPWNZ MACRO    ENDER
ANDLW  0xFF
BTFS S STATUS, Z
GOTO   ENDER
ENDM
```

### **Exemplo**

```
...
MOV LW 0x05 ; W=0x04
JPWNZ PONTOB ; testa se W é diferente de zero e
               ; desvia para o PONTOB
PONTOB:
...
               ; o programa continua daqui
```

No exemplo acima, o programa desviou pois W era diferente de zero

### **7.15.3 - Outras Funções**

#### **COMBF - Complementa um bit do registrador**

Formato: COMBF <registraror>,<bit>

A macroinstrução COMBF é utilizada para complementar o bit <bit> do registrador <registraror>.

O flag Z é alterado. Observe que o registrador **W também é alterado!**

```
COMBF MACRO REG,BIT
  IF (BIT==0)
    MOVLW B'00000001'
  ENDIF
  IF (BIT==1)
    MOVLW B'00000010'
  ENDIF
  IF (BIT==2)
    MOVLW B'00000100'
  ENDIF
  IF (BIT==3)
    MOVLW B'00001000'
  ENDIF
  IF (BIT==4)
    MOVLW B'00010000'
  ENDIF
  IF (BIT==5)
    MOVLW B'00100000'
  ENDIF
  IF (BIT==6)
    MOVLW B'01000000'
  ENDIF
  IF (BIT==7)
    MOVLW B'10000000'
  ENDIF
  XORWF REG,F
ENDM
```

**Exemplo**

```

...
MOVLW B'10001100'      ; W= B'10001100'
MOVWF TESTE            ; TESTE=B'10001100'
COMBF  TESTE,7          ; complementa bit 7 de TESTE
                      ; TESTE=B'00001100'

```

## 7.16 - Convertendo 16F84 em 16F62x

A conversão dos programas escritos para o 16F84 para utilização nos 16F62x não oferece grandes desafios, já que os PICs 16F62x são versões melhoradas dos 16F84 e possuem compatibilidade de software e hardware, inclusive pino a pino.

No entanto, devemos lembrar de algumas diferenças existentes entre os chips:

1. Nos PICs 16F84, a área de memória RAM do usuário (os registradores GPR) tem início no endereço 0x0C. Já nos 16F62x os registradores GPR se iniciam no endereço 0x20. Por isso é necessário verificar todas as definições de variáveis e alterar as que estejam fora dos limites de memória do 16F62x;
2. No PIC 16F84, os registradores GPR são encontrados tanto no banco 0 como no banco 1 (devido ao fato de serem espelhados). Nos PICs 16F62x isto não acontece: os registradores GPR do banco 1 são diferentes dos do banco 0 (com exceção dos últimos 16 registradores GPR localizados entre os endereços 0x70 e 0x7F e que estão espelhados em todos os bancos). Desta forma, é necessário verificar se os bits RP0 e RP1 estão corretamente ajustados antes dos acessos aos registradores GPR;
3. É necessário desabilitar o módulo interno de comparadores analógicos (o que pode ser feito escrevendo o valor 0x07 no registrador CMCON), desta forma os pinos RA0 a RA4 podem ser utilizados para I/O digital;
4. Em relação à EEPROM interna foram modificados os bancos de memória dos registradores EEADR e EEDATA, além do bit de habilitação de interrupção que saiu do registrador INTCON e foi para o registrador PIE1, o bit EIF foi deslocado do registrador EECON1 para o registrador PIR1. Desta forma, antes de toda referência aos registradores EEADR e EEDATA deve ser providenciado o ajuste do banco de memória correto (agora é o banco 1). Observe também que se for desejado utilizar a interrupção da EEPROM, deve ser providenciada a habilitação das interrupções periféricas (o que pode ser feito por uma instrução BSF INTCON, PEIE);
5. A diretiva LIST ou PROCESSOR deve ser modificada de acordo com o processador utilizado;
6. A diretiva CONFIG deve incluir as chaves \_LVP\_OFF e \_BODEN\_OFF.

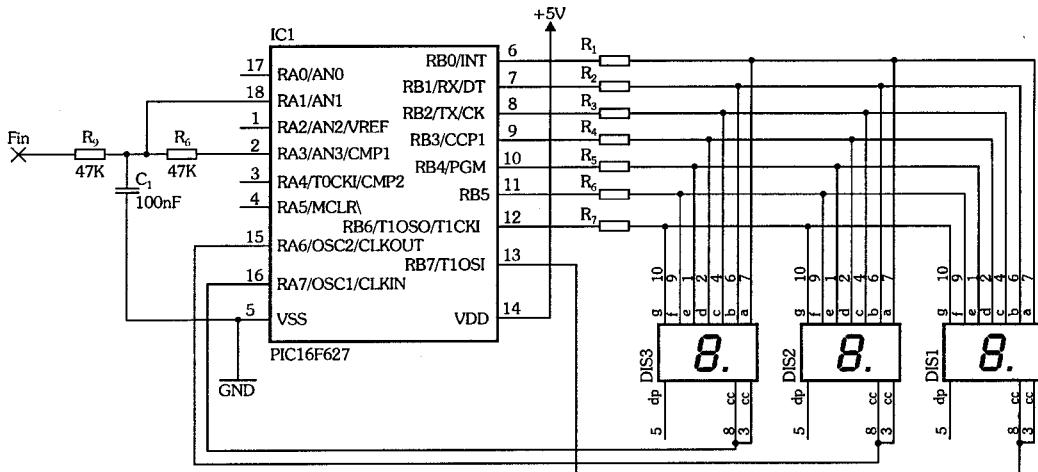
## Projetos

### 8.1 - Voltímetro Digital Simples

O primeiro projeto que vamos analisar é um voltímetro digital de 8 bits e três dígitos.

Conforme podemos ver na figura 8.1, o circuito é muito simples e permite uma faixa de medição de 0 a 5V.

Um detalhe importante é que o valor apresentado no display corresponde diretamente ao resultado decimal da conversão A/D. Seria ainda necessário o escalonamento do valor para a apresentação correta dos valores.



**Figura 8.1**

O programa a ser executado será constituído de três módulos já estudados:

- ◆ Decodificador multiplexado de display;
- ◆ Conversor A/D Delta-sigma modificado para 8 bits;
- ◆ Conversor binário/decimal 8 bits.

Vejamos então a listagem de um esboço do programa:

```

; Voltímetro Digital de 8 bits
LIST P=16F627
INCLUDE <P16F627.INC>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
DESLOCAMENTO EQU 0x20
D0 EQU 0x21
D1 EQU 0x22
D2 EQU 0x23
CONTA EQU 0x24
RESULTADO_L EQU 0x25
RESULTADO_H EQU 0x26
DELTA_L EQU 0x27
DELTA_H EQU 0x28
ORG 0x0000
GOTO INICIO
DECOD_DISPLAY:
    MOVWF DESLOCAMENTO ; armazena o número na variável deslocamento
    MOVLW LOW TABELA ; copia em W os 8 bits LSB do endereço da TABELA
    ADDWF DESLOCAMENTO,F ; adiciona o valor à variável DESLOCAMENTO
    MOVLW HIGH TABELA ; copia em W os 5 bits superiores do endereço da
TABELA
    BTFSC STATUS,C ; testa para ver se a soma anterior transbordou
    ADDLW 0x01 ; se transbordou, soma 1 ao W
    MOVWF PCLATH ; acerta o PCLATH de acordo com o endereço da TABELA
    MOVF DESLOCAMENTO,W ; copia o valor do DESLOCAMENTO para o W
    MOVWF PCL ; copia o W para o PCL (desvia para a tabela)
TABELA:
    RETLW B'10111111' ; número 0
    RETLW B'10000110' ; número 1
    RETLW B'11011011' ; número 2
    RETLW B'11001111' ; número 3
    RETLW B'11100110' ; número 4
    RETLW B'11101101' ; número 5
    RETLW B'11111101' ; número 6
    RETLW B'10000111' ; número 7
    RETLW B'11111111' ; número 8
    RETLW B'11100111' ; número 9
    RETLW B'11110111' ; dígito A
    RETLW B'11111100' ; dígito B
    RETLW B'10111001' ; dígito C
    RETLW B'11011110' ; dígito D
    RETLW B'11111001' ; dígito E
    RETLW B'11110001' ; dígito F
ATRASO:
    ADDLW 1 ; soma 1 ao W
    BTFSC STATUS,Z ; testa se W=0 ...
    RETURN ; se W=0, retorna
    GOTO ATRASO ; se W diferente de 0, desvia para ATRASO_LOOP
DISPLAY:
    MOVF D0,W ; copia o dígito em W
    CALL DECOD_DISPLAY ; decodifica o dígito
    MOVLW PORTB ; ativa os segmentos do display
    BCF PORTB,7 ; ativa display 1
    CLRW ; W=0
    CALL ATRASO ; espera
    BSF PORTB,7 ; desativa display 1
    MOVF D1,W ; copia o próximo dígito em W

```

```

CALL    DECOD_DISPLAY ; decodifica o display
MOVWF  PORTB          ; ativa os segmentos
BCF    PORTA, 6        ; ativa display 2
CLRW
       ; W=0
CALL    ATRASO         ; espera
BSF    PORTA, 6        ; desativa display 2
MOVF   D2,W            ; copia o último dígito em W
CALL    DECOD_DISPLAY ; decodifica display
MOVWF  PORTB          ; ativa os segmentos
BCF    PORTA, 7        ; ativa o display 3
CLRW
       ; W=0
CALL    ATRASO         ; espera
BSF    PORTA, 7        ; desativa o display 3
RETURN

BIN8DEC:
MOVWF  D0              ; copia o número a ser convertido para a variável de
                        ; unidades
CLRF   D1              ; limpa as dezenas
CLRF   D2              ; limpa as centenas

BIN8DEC_1:
MOVlw  D'100'           ; subtrai 100 decimal ...
SUBWF  D0,W             ; ... do valor a ser convertido e guarda em W
BTFSs  STATUS,C         ; o resultado é positivo ou zero ?
GOTO   BIN8DEC_2         ; não ? então vai para a próxima parte
MOVWF  D0              ; sim ? então copia o valor para as unidades ...
INCF   D2,F             ; incrementa um nas centenas
GOTO   BIN8DEC_1         ; e reinicia o ciclo

BIN8DEC_2:
MOVlw  D'10'             ; subtrai 10 decimal do valor ...
SUBWF  D0,W             ; ... a ser convertido e guarda o resultado em W
BTFSs  STATUS,C         ; o resultado é positivo ou zero ?
RETURN                         ; não ? então retorna, a conversão terminou
MOVWF  D0              ; sim ? então copia W para as unidades ...
INCF   D1,F             ; incrementa um nas dezenas
GOTO   BIN8DEC_2         ; e reinicia o ciclo

INICIALIZA_CONV_DELTA:
BSF    STATUS,RP0        ; seleciona o banco 1
MOVlw  0xEC
MOVWF  VRCON
BCF    TRISA,3
BCF    STATUS,RP0
MOVlw  0x05
MOVWF  CMCON
RETURN

CONVERSOR_DELTA:
CLRF   RESULTADO_L
CLRF   RESULTADO_H
CLRF   DELTA_L
CLRF   DELTA_H
MOVlw  0x03
MOVWF  CMCON
; inicializa o módulo comparador analógico

CICLO_CONVERSAO:
BTFS  CMCON,C1OUT
GOTO  REF_MENOR
; verifica a saída do comparador 1
; se C1OUT=1 vai para REF_MENOR

REF_MAIOR:
NOP
BCF    PORTA, 3
INCFsz RESULTADO_L,F
GOTO  GASTA2CICLOS
INCF   RESULTADO_H,F
GOTO  FINAL_CICLO
; gasta 1 ciclo
; coloca a saída RA3 em nível '1'
; incrementa o resultado LSB
; vai para GASTA2CICLOS
; incrementa o resultado MSB
; vai para o final do ciclo

REF_MENOR:

```

```

BSF      PORTA,3      ; coloca a saída RA3 em '1'
NOP      ; gasta 1 ciclo
GOTO    GASTA2CICLOS ; gasta mais 2 ciclos
GASTA2CICLOS:
GOTO    FINAL_CICLO   ; vai para o final do ciclo
FINAL_CICLO:
INCFSZ  DELTA_L,F    ; incrementa o contador de passos de conversão LSB
GOTO    GASTA5CICLOS ; gasta 5 ciclos
INCF    DELTA_H,F    ; incrementa o contador de passos de conversão MSB
; o código a seguir determina se a contagem de ciclos ultrapassou 256
MOVF   DELTA_H,W    ;
ANDLW  B'00000001'  ;
BTFSR  STATUS,Z    ;
GOTO    CICLO_CONVERSAO ;
GOTO    FIM_CONVERSAO ;
GASTA5CICLOS:
GOTO    $+1           ; gasta 2 ciclos (desvia para a próxima instrução)
NOP      ; gasta 1 ciclo
GOTO    CICLO_CONVERSAO ; inicia novo ciclode conversão
FIM_CONVERSAO:
MOVLW  0x06          ;
MOVWF  CMCON         ; configura o módulo comparador para modo 6
RETURN
INICIO:
; primeiramente vamos inicializar a porta B como saída e os pinos
; RA3, RA6 e RA7 como saídas, os demais pinos da porta A serão configurados
; como entradas
MOVLW  B'00000101'  ;
MOVWF  CMCON         ; configura os comparadores analógicos para
modo 5
BANKSEL TRISA        ;
MOVLW  B'00000110'  ; os pinos RA1 e RA2 como entradas e os demais
MOVWF  TRISA         ; como saídas
CLRF   TRISB         ; configura a porta B para saídas
BANKSEL PORTB        ;
MOVLW  B'11000000'  ; desliga os catodos dos dois ...
MOVWF  PORTA         ; ... displays
BSF    PORTB,7       ; desliga o catodo do terceiro display
CLRF   D0             ;
CLRF   D1             ;
CLRF   D2             ;
MOVLW  D'20'          ; inicializa o temporizador de conversão em
MOVWF  CONTA         ; 20 ciclos
LOOP_PRINCIPAL:
CALL    DISPLAY       ; apresenta o resultado atual no display
DECFSZ CONTA,F      ; decrementa temporizador de conversão
GOTO    LOOP_PRINCIPAL ; se CONTA>0 retorna para o LOOP_PRINCIPAL
MOVLW  D'20'          ;
MOVWF  CONTA         ; reinicia contador em 20 decimal
CALL    CONVERSOR_DELTA ; realiza uma conversão
MOVF   RESULTADO_L,W  ; copia o resultado de 8 bits em W
CALL    BIN8DEC       ; converte o resultado em decimal
GOTO    LOOP_PRINCIPAL ; retorna para o LOOP_PRINCIPAL
END

```

## 8.2 - Relógio Digital com Alarme

Em seguida, apresentamos o projeto de um relógio digital com alarme. Ele apresenta um conjunto de quatro displays, sendo capaz de apresentar as horas e minutos no formato HH:MM.

Os cinco leds possuem as seguintes funções:

- ◆ D1 e D2 - Pontos separadores das horas e minutos;
- ◆ D3 - Indicador do modo horário;
- ◆ D4 - Indicador do modo alarme;
- ◆ D5 - Indicador de alarme ligado.

Além disso, o relógio possui um conjunto de quatro teclas com as seguintes funções:

- ◆ S1 - Incrementa as horas (horário/alarme);
- ◆ S2 - Incrementa os minutos (horário/alarme);
- ◆ S3 - Mudança de modo (apresentação das horas ou horário do alarme);
- ◆ S4 - Liga/desliga o alarme (desliga o alarme quando disparado).

Utilizamos o timer 1 para gerar a base de tempo de 1 segundo para o funcionamento do relógio.

Um aspecto interessante sobre este projeto é a utilização de multiplexação de display e teclado nos mesmos pinos de I/O. Isto é conseguido graças à capacidade de alteração de direção dos pinos de I/O durante a execução do programa, ou seja: durante a apresentação do display, os pinos são configurados como saídas; já durante a leitura do teclado, os pinos RB0 a RB3 são configurados como entradas (os Pull-ups internos são também ativados).

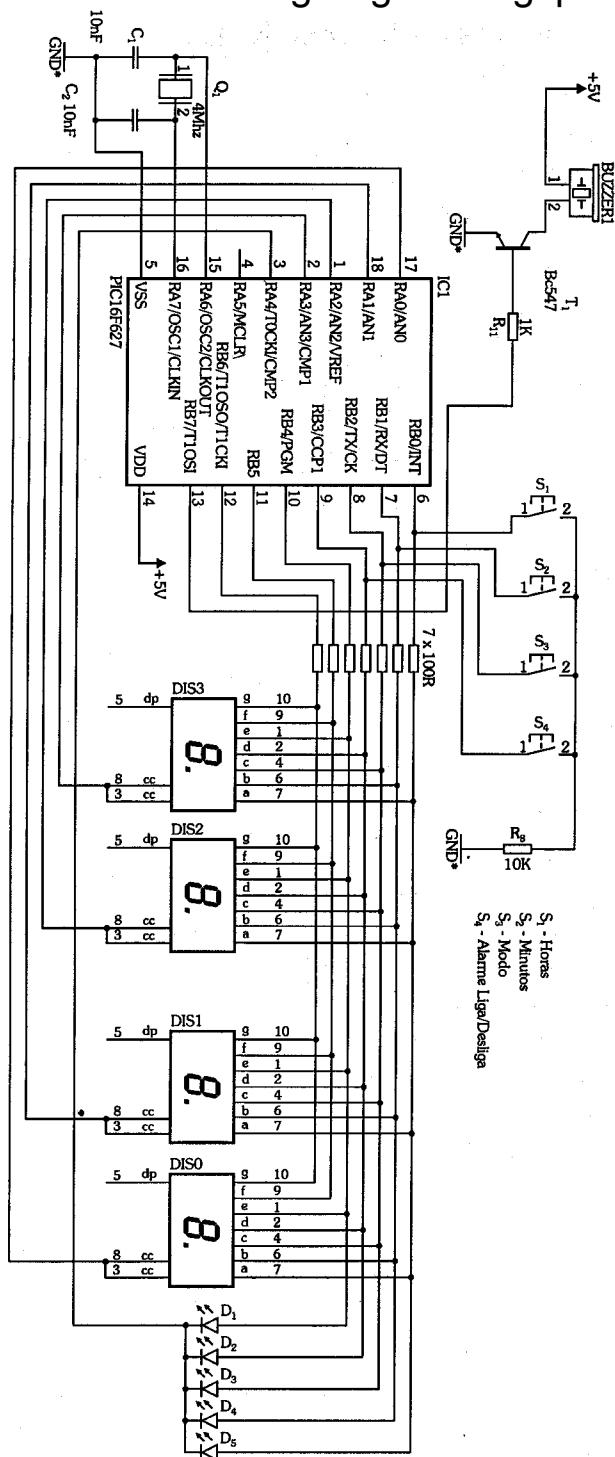


Figura 8.2

Em seguida temos a listagem do esboço do programa para o relógio:

```

; Relógio digital com alarme
LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
;
;
*****  

; Definição das variáveis
;  

HORA EQU 0x20 ; armazenamento das horas (00 a 23)
MINUTO EQU 0x21 ; armazenamento dos minutos (00 a 59)
H_AL EQU 0x22 ; armazenamento das horas do alarme (00 a 23)
M_AL EQU 0x23 ; armazenamento dos minutos do alarme (00 a 59)
TEMP EQU 0x24 ; registrador temporário de uso geral
TEMP1 EQU 0x25 ; registrador temporário de uso geral
FLAGS EQU 0x26 ; flags de estado do relógio
SEG EQU 0x27 ; segundos
D0 EQU 0x28 ;
D1 EQU 0x29 ;
W_TEMP EQU 0x2A ;
STATUS_TEMP EQU 0x2B ;
T1_CONTA EQU 0x2C ; contador auxiliar do timer 1
TECLA EQU 0x2D ; número da tecla pressionada
TECLA_C EQU 0x2E ; temporizador das teclas
#define AL_ON FLAGS,0 ; indicador de alarme ligado
#define AL_DIS FLAGS,1 ; indicador de alarme disparado
#define MODO FLAGS,2 ; modo (0-relógio,1-alarme)
#define PONTO FLAGS,3 ; Pontos separadores HH:MM
#define S1 PORTB,0 ; Horas
#define S2 PORTB,1 ; Minutos
#define S3 PORTB,2 ; Modo
#define S4 PORTB,3 ; Alarme
#define DIS0 PORTA,0 ; Catodo do display DIS0
#define DIS1 PORTA,1 ; Catodo do display DIS1
#define DIS2 PORTA,2 ; Catodo do display DIS2
#define DIS3 PORTA,3 ; Catodo do display DIS3
#define LEDS PORTA,4 ; Catodo dos Leds
#define PONTO1 0 ; Anodo do Led separador
#define PONTO2 1 ; Anodo do Led separador
#define LED_MH 2 ; Anodo do LED de modo hora
#define LED_MA 3 ; Anodo do LED de modo alarme
#define LED_AL 4 ; Anodo do LED de alarme ligado
#define BUZZER PORTB,7 ; Buzzer

ORG 0x0000
GOTO INICIO
ORG 0x0004
    SALVA_CONTEXTO ; salva o contexto atual
    BCF PIR1,TMR1IF ; apaga flag de interrupção do timer 1
    MOVlw 0xDC ;
    MOVwf TMR1L ;
    MOVlw 0x0B ;
    MOVwf TMR1H ; inicializa o timer 1 em 3036 decimal
    DECFSZ T1_CONTA,F ; decrementa a variável de contagem do timer 1
    GOTO FIM_INT ; se diferente de zero sai da interrupção
    MOVlw 2 ;
    MOVwf T1_CONTA ; reinicializa a variável de contagem em 2

```

```

INCF    SEG, F      ; incrementa 1 segundo
MOVLW  D'60'        ; verifica ...
XORWF  SEG,W       ; se passou de 59 segundos ...
BTFS S STATUS,Z    ;
GOTO   FIM_INT     ; se não, sai da interrupção
CLRF   SEG          ; se passou ...
INCF   MINUTO,F    ; incrementa 1 nos minutos
MOVLW  D'60'        ; verifica se passou ...
XORWF  MINUTO,W    ; ...
BTFS S STATUS,Z    ; de 59 minutos ...
GOTO   FIM_INT     ; se não passou, sai da interrupção
CLRF   MINUTO      ; se passou, apaga os minutos
INCF   HORA,F      ; incrementa 1 hora
MOVLW  D'24'        ; verifica se passou de
XORWF  HORA,W      ; ...
BTFS S STATUS,Z    ; 23 horas ...
GOTO   FIM_INT     ; copia o W para o PCL (desvia para a tabela)

```

## TABELA:

```

RETLW  B'00111111' 
RETLW  B'000000110'
RETLW  B'01011011'
RETLW  B'01001111'
RETLW  B'01100110'
RETLW  B'01101101'
RETLW  B'01111101'
RETLW  B'000000111'
RETLW  B'01111111'
RETLW  B'01100111'
RETLW  B'01101111'
RETLW  B'01111110'
RETLW  B'01111100'
RETLW  B'00111001'
RETLW  B'01011110'
RETLW  B'011111001'
RETLW  B'01110001'

```

```

; número 0
; número 1
; número 2
; número 3
; número 4
; número 5
; número 6
; número 7
; número 8
; número 9
; dígito A
; dígito B
; dígito C
; dígito D
; dígito E
; dígito F

```

## ATRASO:

```

ADDLW  1            ; soma 1 ao W
BTFS C STATUS,Z    ; testa se W=0 ...
RETURN                         ; se W=0, retorna
GOTO   ATRASO         ; se W diferente de 0, desvia para ATRASO_LOOP

```

## DISPLAY:

```

BTFS C MODO          ;
GOTO   MOSTRA_ALARME ;
MOVF   MINUTO,W      ; copia o dígito em W
CALL   BIN8DEC        ;
MOVF   D0,W           ;
CALL   DECOD_DISPLAY ; decodifica o dígito
MOVWF  PORTB          ; ativa os segmentos do display
BTFS C AL_DIS         ; testa se o alarme está disparado
BSF    BUZZER         ; ativa o buzzer
BCF    DIS0            ; ativa display 0
CLRW
CALL   ATRASO         ; espera
BSF    DIS0            ; desativa display 0
MOVE   D1,W           ;
CALL   DECOD_DISPLAY ; decodifica o dígito
MOVWF  PORTB          ; ativa os segmentos do display
BTFS C AL_DIS         ; testa se o alarme está disparado
BSF    BUZZER         ; ativa o buzzer
BCF    DIS1            ; ativa display 0
CLRW

```

```

CALL ATRASO ; espera
BSF DIS1 ; desativa display 0
MOVF HORA,W ; copia o dígito em W
CALL BIN8DEC ;
MOVF D0,W ;
CALL DECOD_DISPLAY ; decodifica o dígito
MOVWF PORTB ; ativa os segmentos do display
BTFS C AL_DIS ; testa se o alarme está disparado
BSF BUZZER ; ativa o buzzer
BCF DIS2 ; ativa display 0
CLRW ; W=0
CALL ATRASO ; espera
BSF DIS2 ; desativa display 0
MOVF D1,W ;
CALL DECOD_DISPLAY ; decodifica o dígito
MOVWF PORTB ; ativa os segmentos do display
BTFS C AL_DIS ; testa se o alarme está disparado
BSF BUZZER ; ativa o buzzer
BCF DIS3 ; ativa display 0
CLRW ; W=0
CALL ATRASO ; espera
BSF DIS3 ; desativa display 0
GOTO MOSTRA_LEDs ;

```

**MOSTRA\_ALARME:**

```

MOVF M_AL,W ; copia o dígito em W
CALL BIN8DEC ;
MOVF D0,W ;
CALL DECOD_DISPLAY ; decodifica o dígito
MOVWF PORTB ; ativa os segmentos do display
BTFS C AL_DIS ; testa se o alarme está disparado
BSF BUZZER ; ativa o buzzer
BCF DIS0 ; ativa display 0
CLRW ; W=0
CALL ATRASO ; espera
BSF DIS0 ; desativa display 0
MOVF D1,W ;
CALL DECOD_DISPLAY ; decodifica o dígito
MOVWF PORTB ; ativa os segmentos do display
BTFS C AL_DIS ; testa se o alarme está disparado
BSF BUZZER ; ativa o buzzer
BCF DIS1 ; ativa display 0
CLRW ; W=0
CALL ATRASO ; espera
BSF DIS1 ; desativa display 0
MOVF H_AL,W ; copia o dígito em W
CALL BIN8DEC ;
MOVF D0,W ;
CALL DECOD_DISPLAY ; decodifica o dígito
MOVWF PORTB ; ativa os segmentos do display
BTFS C AL_DIS ; testa se o alarme está disparado
BSF BUZZER ; ativa o buzzer
BCF DIS2 ; ativa display 0
CLRW ; W=0
CALL ATRASO ; espera
BSF DIS2 ; desativa display 0
MOVF D1,W ;
CALL DECOD_DISPLAY ; decodifica o dígito
MOVWF PORTB ; ativa os segmentos do display
BTFS C AL_DIS ; testa se o alarme está disparado
BSF BUZZER ; ativa o buzzer
BCF DIS3 ; ativa display 0
CLRW ; W=0

```

```

CALL    ATRASO      ; espera
BSF     DIS3        ; desativa display 0
MOSTRA_LEDs:
CLRW
BTFS C AL_ON       ; apaga o W
BSW   LED_AL       ; AL_ON=1 ?
BTFS S MODO        ; sim -> ativa led LED_AL
BSW   LED_MH       ; modo relógio ?
BTFS C MODO        ; sim -> acende led de modo hora
BSW   LED_MA       ; modo alarme ?
BTFS C PONTO       ; sim -> acende led de modo alarme
IORLW  B'00000011'  ; pontos ativados ?
BTFS C AL_DIS      ; sim -> acende os leds separadores
BSW   7             ; testa se o alarme está disparado
MOVWF PORTB
BCF   LEDS         ; ativa o buzzer
CLRW
CALL  ATRASO       ;
BSF   LEDS         ; copia o valor final de W para o PORTB
RETURN
BIN8DEC:
MOVWF D0           ; copia o número a ser convertido para a variável de
                     ; unidades
CLRF  D1           ; limpa as dezenas
BIN8DEC_2:
MOVLW D'10'
SUBWF D0,W
BTFS S STATUS,C
RETURN
MOVWF D0           ; subtrai 10 decimal ...
                     ; ... do valor a ser convertido e guarda em W
                     ; o resultado é positivo ou zero ?
                     ; não ? então retorna, a conversão terminou
                     ; sim ? então copia o resultado em W para as
unidades
INCF  D1,F         ; incrementa um nas dezenas
GOTO  BIN8DEC_2    ; e reinicia o ciclo
TECLAS:
BANKSEL TRISB
MOVLW B'00001111'
MOVWF TRISB
BANKSEL T1CON
BTFS S S1          ; configura RB0 a RB3 como entradas
GOTO  TECLAS_S1
BTFS S S2          ; testa se é a tecla S1
GOTO  TECLAS_S2
BTFS S S3          ; vai para tratamento da tecla S1
GOTO  TECLAS_S3
BTFS S S4          ; testa se é a tecla S2
GOTO  TECLAS_S4
CLRF  TECLA         ; vai para tratamento da tecla S3
                     ; testa se é a tecla S4
                     ; vai para tratamento da tecla S4
                     ; apaga a tecla (nenhuma foi pressionada)
TECLAS_FIM:
MOVLW D'30'
MOVWF TECLA_C
TECLAS_FIM2:
BANKSEL TRISB
CLRF  TRISB
BANKSEL T1CON
RETURN
TECLAS_S1:
MOVLW 1             ; configura porta B para saída
XORWF TECLA,F      ; retorna
MOVWF TECLA
BTFS S STATUS,Z
GOTO  TECLAS_FIM

```

```

MOVEF    TECLA_C,F      ; testa se o temporizador das teclas ...
BTFSZ    STATUS,Z       ; ... está em zero ...
GOTO     TECLAS_FIM2   ; se estiver, vai para o final
DECFSZ   TECLA_C,F      ; decrementa o temporizador das teclas
GOTO     TECLAS_FIM2   ; se maior que zero vai para o final
BTFFS    MODO            ; se o MODO=0
INCF    HORA,F          ; incrementa a hora atual
BTFSZ    MODO            ; se o MODO=1
INCF    H_AL,F          ; incrementa a hora do alarme
MOVLW   D'24'           ; verifica ...
XORWF   HORA,W          ; se a hora é igual a 24
BTFSZ    STATUS,Z       ; se for ...
CLRF    HORA            ; zera a hora
MOVLW   D'24'           ; verifica ...
XORWF   H_AL,W          ; se a hora do alarme é igual a 24
BTFSZ    STATUS,Z       ; se for ...
CLRF    H_AL             ; zera a hora do alarme
GOTO     TECLAS_FIM2   ; vai para o final

TECLAS_S2:
MOVLW   2                ; compara ...
XORWF   TECLA,F          ; a tecla pressionada com tecla anterior ...
MOVWF   TECLA            ; tecla atual é a 2
BTFFS    STATUS,Z       ; se a tecla atual é diferente da ...
GOTO     TECLAS_FIM      ; ... anterior, vai para o final
MOVF    TECLA_C,F        ; testa se o temporizador das teclas ...
BTFSZ    STATUS,Z       ; ... está em zero ...
GOTO     TECLAS_FIM2   ; se estiver, vai para o final
DECFSZ   TECLA_C,F      ; decrementa o temporizador das teclas
GOTO     TECLAS_FIM2   ; se maior que zero vai para o final
BTFFS    MODO            ; se o MODO=0
INCF    MINUTO,F         ; incrementa o minuto atual
BTFSZ    MODO            ; se o MODO=1
INCF    M_AL,F           ; incrementa o minuto do alarme
MOVLW   D'60'            ; verifica ...
XORWF   MINUTO,W          ; se os minutos são iguais a 60
BTFSZ    STATUS,Z       ; se forem ...
CLRF    HORA            ; zera os minutos
MOVLW   D'60'            ; verifica ...
XORWF   M_AL,W           ; se os minutos do alarme são iguais a 60
BTFSZ    STATUS,Z       ; se forem ...
CLRF    H_AL             ; zera os minutos do alarme
GOTO     TECLAS_FIM2   ; vai para o final

TECLAS_S3:
MOVLW   3                ; compara ...
XORWF   TECLA,F          ; a tecla pressionada com tecla anterior ...
MOVWF   TECLA            ; tecla atual é a 3
BTFFS    STATUS,Z       ; se a tecla atual é diferente da ...
GOTO     TECLAS_FIM      ; ... anterior, vai para o final
MOVF    TECLA_C,F        ; testa se o temporizador das teclas ...
BTFSZ    STATUS,Z       ; ... está em zero ...
GOTO     TECLAS_FIM2   ; se estiver, vai para o final
DECFSZ   TECLA_C,F      ; decrementa o temporizador das teclas
GOTO     TECLAS_FIM2   ; se maior que zero vai para o final
COMBF    MODO            ; muda o modo
GOTO     TECLAS_FIM2   ; vai para o final

TECLAS_S4:
MOVLW   4                ; compara ...
XORWF   TECLA,F          ; a tecla pressionada com tecla anterior ...
MOVWF   TECLA            ; tecla atual é a 4
BTFFS    STATUS,Z       ; se a tecla atual é diferente da ...
GOTO     TECLAS_FIM      ; ... anterior, vai para o final
MOVF    TECLA_C,F        ; testa se o temporizador das teclas ...

```

```

BTFSC STATUS,Z ; ... está em zero ...
GOTO TECLAS_FIM2 ; se estiver, vai para o final
DECFSZ TECLA_C,F ; decrementa o temporizador das teclas
GOTO TECLAS_FIM2 ; se maior que zero vai para o final
BTFSC AL_DIS ; o alarme está disparado ?
GOTO DESLIGA_ALARME ; sim -> vai para DESLIGA_ALARME
COMBF AL_ON ; inverte o estado do alarme
GOTO TECLAS_FIM2 ; vai para o final

DESLIGA_ALARME:
BCF BUZZER ; desliga buzzer
GOTO TECLAS_FIM2 ; vai para o final

VERIFICA_ALARME:
BTFSS AL_ON ; verifica se o alarme está ligado
RETURN ; não ? Então retorna
MOVF HORA,W ; testa se a hora ...
XORWF H_AL,W ; é igual a do alarme ...
BTFSS STATUS,Z ;
RETURN ; não ? Então retorna
MOVF MINUTO,W ; testa se os minutos ...
XORWF M_AL,W ; são iguais aos do alarme ...
BTFSS STATUS,Z ;
RETURN ; não ? Então retorna
; se o programa chegou até aqui, então o horário do alarme é igual ao
; horário atual. O alarme então irá disparar
BSF AL_DIS ; ativa o flag do alarme
BSF BUZZER ; ativa o buzzer
RETURN ; retorna

INICIO:
MOVLW B'00000111' ; configura os comparadores analógicos para modo 7
MOVWF CMCON ;
BANKSEL TRISA ; porta A como saída
CLRF TRISA ;
CLRF TRISB ; porta B como saída
MOVLW B'00000001' ;
MOVWF PIE1 ; habilita interrupção do timer 1
CLRF OPTION_REG ; ativa resistores de PULL-UP
BANKSEL T1CON ;
MOVLW B'00110000' ;
MOVWF T1CON ; configura o timer 1 para clock interno e
; prescaler 1:8
MOVLW 0xDC ;
MOVWF TMR1L ;
MOVLW 0x0B ;
MOVWF TMR1H ; inicializa o timer 1 em 3036 decimal
CLRF SEG ; apaga variáveis
CLRF MINUTO ;
CLRF HORA ;
CLRF H_AL ;
CLRF M_AL ;
CLRF FLAGS ;
BSF T1CON, TMR1ON ; ativa a contagem do timer 1
MOVLW B'11000000' ;
MOVWF INTCON ; habilita GIE e PEIE

LOOP_PRINCIPAL:
CALL DISPLAY ; mostra display
CALL TECLAS ; verifica teclas
CALL VERIFICA_ALARME ; verifica o horário
BTFS C AL_DIS ; ativa o alarme
GOTO LOOP_PRINCIPAL ; retorna para o LOOP_PRINCIPAL

END

```

## 8.3 - Freqüencímetro/Contador de 7 Dígitos

Em seguida, veremos a construção de um freqüencímetro/contador digital de 7 dígitos capaz de medir freqüências de até 40 Mhz e contar impulsos ou eventos até 9.999.999.

Novamente, graças aos diversos periféricos internos do PIC 16F627, não utilizaremos qualquer outro componente ativo, a não ser o PIC e os 7 displays do tipo catodo comum.

O princípio de funcionamento é bastante simples: o sinal a ser medido é aplicado à entrada de clock externo do timer 1 (T1CKI), fazendo com que ele conte a quantidade de pulsos externos.

A contagem do timer 1 é inicializada em 55536 e a cada estouro da sua contagem (10000 contagens) a variável FT4 é incrementada. Se o valor dela for igual a 10, é zerada e a variável FT5 é incrementada. Novamente, se a variável FT5 for igual a 10, ela é zerada e a variável FT6 é incrementada.

Caso o valor da variável FT6 seja igual a 10, todas as variáveis de contagem são apagadas e o timer 1 é reiniciado na sua contagem inicial (55536 decimal).

No modo freqüencímetro, a contagem é totalizada a cada segundo, o valor da contagem atual é repassado para todos os displays e os contadores são reiniciados do zero para uma nova medição.

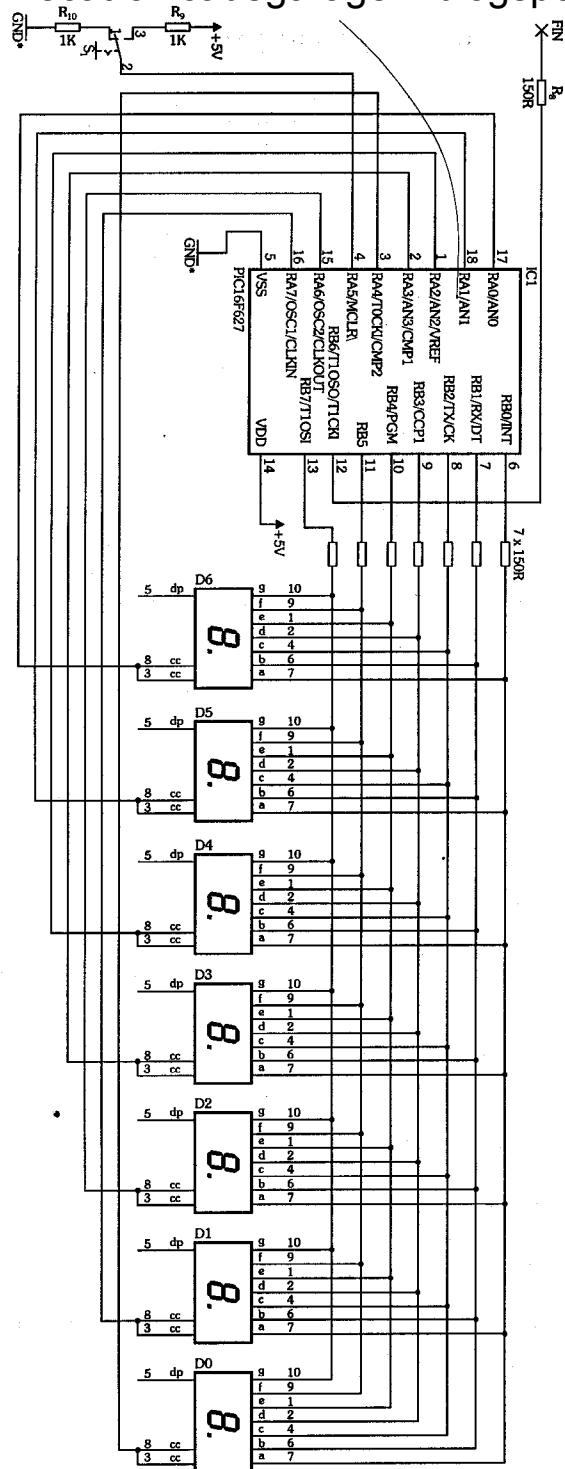
No modo contador de eventos, a contagem nunca é zerada e o valor apresentado no display representa a quantidade de impulsos recebidos desde o último reset do sistema.

Uma característica interessante do software utilizado é que foi usado um flag de software (ATUALIZA) para notificar o programa principal de que o valor das variáveis deve ser atualizado para apresentação no display. Esse flag é setado a cada nova medição totalizada (pela rotina do timer 0).

A cada novo ciclo do programa principal, o flag é verificado e caso esteja em nível '1', é executada a sub-rotina de atualização dos valores.

Esta sub-rotina irá copiar os valores da última contagem (armazenados nas variáveis F4, F5, F6, A0 e A1) da seguinte forma: F4 para o display 4 (DISP4), F5 para o display 5 (DISP5), F6 para o display 6 (DISP6). Os quatro dígitos menos significativos do display (DISP0 a DISP4) são obtidos da conversão binário-decimal do valor armazenado em A0 e A1.

Observe que para uma maior precisão na medição, deveríamos utilizar um cristal oscilador externo. Neste caso, seria necessário adicionar um circuito integrado (um decodificador, por exemplo) ao projeto, ou diminuir a quantidade de displays utilizados.



*Figura 8.3*

Em seguida temos o esboço do programa:

```

; Freqüencímetro Digital e Contador de Eventos
; O timer 0 é utilizado para fornecer a base de tempo de 1 segundo
; O timer 1 é utilizado para fazer a contagem primária dos pulsos de entrada
LIST P=16F627
INCLUDE <P16F627.INC>
INCLUDE <MACROS.ASM>
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_OFF & _BODEN_OFF & _MCLRE_OFF &
_INTRC_OSC_NOCLKOUT & _LVP_OFF
;
; ****
; Definição das variáveis
;

FLAGS EQU 0x20 ; armazenamento de flags (MODO e ATUALIZA)
FT4 EQU 0x21 ; freqüência temporária 4
FT5 EQU 0x22 ; freqüência temporária 5
FT6 EQU 0x23 ; freqüência temporária 6
A0 EQU 0x24 ; freqüência temporária LSB do timer 1
A1 EQU 0x25 ; freqüência temporária MSB do timer 1
F4 EQU 0x26 ; freqüência 4
F5 EQU 0x27 ; freqüência 5
F6 EQU 0x28 ; freqüência 6
DISP0 EQU 0x29 ; dígito 0 do display
DISP1 EQU 0x2A ; dígito 1 do display
DISP2 EQU 0x2B ; dígito 2 do display
DISP3 EQU 0x2C ; dígito 3 do display
DISP4 EQU 0x2D ; dígito 4 do display
DISP5 EQU 0x2E ; dígito 5 do display
DISP6 EQU 0x2F ; dígito 6 do display
TEMP EQU 0x30 ; variável temporária
W_TEMP EQU 0x31 ; W temporário
STATUS_TEMP EQU 0x32 ; STATUS temporário
TEMP0 EQU 0x33 ; variável temporária
TEMP1 EQU 0x34 ; variável temporária
B0 EQU 0x35 ; variável temporária utilizada na subtração
B1 EQU 0x36 ; variável temporária utilizada na subtração
T0_CONTA EQU 0x37 ; contador de interrupções do timer 0
; símbolos
#define D0 PORTB,7 ; catodo do display D0
#define D1 PORTA,7 ; catodo do display D1
#define D2 PORTA,6 ; catodo do display D2
#define D3 PORTA,3 ; catodo do display D3
#define D4 PORTA,2 ; catodo do display D4
#define D5 PORTA,1 ; catodo do display D5
#define D6 PORTA,0 ; catodo do display D6
#define ATUALIZA FLAGS,0 ; flag de atualização
#define MODO FLAGS,1 ; flag de modo
T0_VAL EQU D'18' ; valor inicial da variável de contagem do timer 0

T0_INI EQU D'40' ; valor inicial do timer 0
ORG 0x0000
GOTO INICIO
ORG 0x0004
    SALVA_CONTEXTO ; macro para salvar o conteúdo de W e STATUS
    BTFSC INTCON,T0IF ; testa a interrupção do timer 0
    GOTO TRATA_TIMER0 ; se T0IF=1, desvia para TRATA_TIMER0
    BTFSC PIR1,TMR1IF ; testa a interrupção do timer 1
    GOTO TRATA_TIMER1 ; se TMR1IF=1, desvia para TRATA_TIMER1
FIM_INT:
    RESTAURA_CONTEXTO ; macro para restaurar os valores de W e STATUS
    BSF T1CON,TMR1ON ; liga o timer 1
    RETFIE ; retorna da interrupção
TRATA_TIMER1:
    BCF PIR1,TMR1IF ; limpa o flag de interrupção do timer 1

```

```

BCF      T1CON, TMR1ON    ; desliga o timer 1
MOVLW   0xD8             ; inicializa o timer 1 ...
MOVWF   TMR1H            ; ...
MOVWF   0xFO              ; ...
MOVF    TMR1L            ; com o valor 0xD8FO (ou 55536 decimal)
INCF    FT4, F            ; incrementa FT4
MOVLW   D'10'             ; verifica se FT4 ...
XORWF   FT4, W            ; ... é igual ...
BTFS S STATUS, Z          ; ... a 10 ...
GOTO   FIM_INT            ; se não é, vai para o final da interrupção
CLRF    FT4               ; se é, apaga o FT4 ...
INCF    FT5, F            ; ... e incrementa o FT5
MOVLW   D'10'             ; verifica se FT5 ...
XORWF   FT5, W            ; ... é igual ...
BTFS S STATUS, Z          ; ... a 10 ...
GOTO   FIM_INT            ; se não é, vai para o final da interrupção
CLRF    FT5               ; se é, apaga o FT5 ...
INCF    FT6, F            ; ... e incrementa o FT6
MOVLW   D'10'             ; verifica se FT6 ...
XORWF   FT6, W            ; ... é igual ...
BTFS S STATUS, Z          ; ... a 10 ...
GOTO   FIM_INT            ; se não é, vai para o final da interrupção
CLRF    TMR1L            ; se é, apaga o timer 1 ...
CLRF    TMR1H            ; ...
CLRF    FT4               ; FT4 ...
CLRF    FT5               ; FT5 ...
CLRF    FT6               ; FT6 ...
GOTO   FIM_INT            ; vai para o final da interrupção

TRATA_TIMER0:
BCF      INTCON, TOIF    ; apaga o flag da interrupção do timer 0
MOVLW   T0_INI             ; ...
MOVWF   TMR0               ; reinicia o timer 0 para 39 decimal
DECFL   T0_CONTA           ; decrementa a variável de contagem do timer 0
BTFS S STATUS, Z          ; verifica se T0_CONTA=0
MOVLW   T0_VAL             ; se não é, vai para o fim da interrupção
MOVWF   T0_CONTA           ; se é, reinicial a variável em 18
MOVF    TMR1L, W           ; copia o valor LSB do timer 1 para ...
MOVF    A0                 ; ... o A0
MOVF    TMR1H, W           ; copia o valor MSB do timer 1 para ...
MOVF    A1                 ; ... o A1
MOVF    FT4, W             ; copia o valor de FT4 ...
MOVF    F4                 ; para F4
MOVF    FT5, W             ; copia o valor de FT5 ...
MOVF    F5                 ; para F5
MOVF    FT6, W             ; copia o valor de FT6 ...
MOVF    F6                 ; para F6
BSF     ATUALIZA.          ; ativa flag de atualização dos valores
BTFS C MODO                ; testa qual o modo de funcionamento
GOTO   FIM_INT             ; se modo=1 (contador) então retorna
                           ; se modo=0, (frequencímetro)
MOVLW   0xD8               ; inicializa o timer 1 ...
MOVWF   TMR1H             ; ...
MOVWF   0xFO               ; ...
MOVF    TMR1L             ; com o valor 0xD8FO (ou 55536 decimal)
CLRF    FT4               ; apaga contadores
CLRF    FT5               ; ...
CLRF    FT6               ; ...
GOTO   FIM_INT             ; vai para o final da interrupção

DECOD_DISPLAY:
MOVF    TEMP               ; armazena o número na variável deslocamento
MOVLW   LOW TABELA          ; copia em W os 8 bits LSB do endereço da TABELA
ADDWF   TEMP, F             ; adiciona o valor à variável DESLOCAMENTO
MOVLW   HIGH TABELA         ; copia em W os 5 bits superiores do endereço da
                           ; TABELA
BTFS C STATUS, C           ; testa para ver se a soma anterior transbordou

```

ADDLW 0x01 ; se transbordou, soma 1 ao W  
 MOVWF PCLATH ; acerta o PCLATH de acordo com o endereço da TABELA  
 MOVF TEMP,W ; copia o valor do DESLOCAMENTO para o W  
 MOVWF PCL ; copia o W para o PCL (desvia para a tabela)  
**TABELA:**  
 ; a tabela seguinte foi modificada devido ao uso do pino RB6 como uma entrada  
 RETLW B'00111111' ; número 0  
 RETLW B'00000110' ; número 1  
 RETLW B'10011011' ; número 2  
 RETLW B'10001111' ; número 3  
 RETLW B'10100110' ; número 4  
 RETLW B'10101101' ; número 5  
 RETLW B'10111101' ; número 6  
 RETLW B'00000111' ; número 7  
 RETLW B'10111111' ; número 8  
 RETLW B'10100111' ; número 9  
 RETLW B'10110111' ; dígito A  
 RETLW B'10111100' ; dígito B  
 RETLW B'00111001' ; dígito C  
 RETLW B'10011110' ; dígito D  
 RETLW B'10111001' ; dígito E  
 RETLW B'10110001' ; dígito F  
**ATRASO:**  
 ADDLW 1 ; Sub-rotina de atraso  
 BTFSC STATUS,Z ; soma 1 ao W  
 RETURN ; testa se W=0 ...  
 GOTO ATRASO ; se W=0, retorna  
**DISPLAY:**  
 ; apresenta o primeiro dígito no display D0  
 MOVF DISP0,W ; copia o dígito em W  
 CALL DECOD\_DISPLAY ; decodifica o dígito  
 MOVWF PORTB ; ativa os segmentos do display  
 BCF D0 ; ativa display D0  
 CLRW ; W=0  
 CALL ATRASO ; espera  
 BSF D0 ; desativa display D0  
 ; apresenta o segundo dígito no display D1  
 MOVF DISP1,W ; copia o próximo dígito em W  
 CALL DECOD\_DISPLAY ; decodifica o display  
 MOVWF PORTB ; ativa os segmentos  
 BCF D1 ; ativa display D1  
 CLRW ; W=0  
 CALL ATRASO ; espera  
 BSF D1 ; desativa display D1  
 ; apresenta o terceiro dígito no display D2  
 MOVF DISP2,W ; copia o último dígito em W  
 CALL DECOD\_DISPLAY ; decodifica display  
 MOVWF PORTB ; ativa os segmentos  
 BCF D2 ; ativa o display D2  
 CLRW ; W=0  
 CALL ATRASO ; espera  
 BSF D2 ; desativa o display D2  
 ; apresenta o quarto dígito no display D3  
 MOVF DISP3,W ; copia o último dígito em W  
 CALL DECOD\_DISPLAY ; decodifica display  
 MOVWF PORTB ; ativa os segmentos  
 BCF D3 ; ativa o display D3  
 CLRW ; W=0  
 CALL ATRASO ; espera  
 BSF D3 ; desativa o display D3  
 ; apresenta o quinto dígito no display D4  
 MOVF DISP4,W ; copia o último dígito em W  
 CALL DECOD\_DISPLAY ; decodifica display  
 MOVWF PORTB ; ativa os segmentos

```

BCF      D4          ; ativa o display D4
CLRW     ; W=0
CALL     ATRASO      ; espera
BSF      D4          ; desativa o display D4
; apresenta o sexto dígito no display D5
MOVF    DISP5,W      ; copia o último dígito em W
CALL    DECOD_DISPLAY ; decodifica display
MOVWF   PORTB         ; ativa os segmentos
BCF      D5          ; ativa o display D5
CLRW     ; W=0
CALL     ATRASO      ; espera
BSF      D5          ; desativa o display D5
; apresenta o sétimo dígito no display D6
MOVF    DISP6,W      ; copia o último dígito em W
CALL    DECOD_DISPLAY ; decodifica display
MOVWF   PORTB         ; ativa os segmentos
BCF      D6          ; ativa o display D6
CLRW     ; W=0
CALL     ATRASO      ; espera
BSF      D6          ; desativa o display D6
RETURN
; Sub-rotina de atualização dos valores a serem apresentados no
; display.
ATUALIZACAO:
BCF      ATUALIZA     ; apaga flag de solicitação de
                      ; atualização de valores
MOVF    F6,W         ; copia a contagem de F6
MOVWF   DISP6         ; para o DISP6
MOVF    F5,W         ; copia a contagem de F5
MOVWF   DISP5         ; para o DISP5
MOVF    F4,W         ; copia a contagem de F4
MOVWF   DISP4         ; para o DISP4
; primeiramente, devemos subtrair 55536 do valor em A0 e A1:
MOVLW   0xD8         ;
MOVWF   B1           ;
MOVLW   0xF0         ;
MOVWF   B0           ;
CALL    SUB16B       ;
; a rotina a seguir converte o valor armazenado em A0 e A1 para decimal
; e armazena os dígitos decimais em DISP0 a DISP3
CLRF    DISP0         ; apaga as unidades
CLRF    DISP1         ; apaga as dezenas
CLRF    DISP2         ; apaga as centenas
CLRF    DISP3         ; apaga as milhares
BIN16DEC_3:
MOVF    A0,W         ; copia o LSB binário ...
MOVWF   TEMPO0        ; ... para TEMPO0
MOVF    A1,W         ; copia o MSB binário ...
MOVWF   TEMP1          ; ... para TEMP1
MOVLW   0xE8         ;
MOVWF   B0           ;
MOVLW   0x03         ;
MOVWF   B1           ; B=03E8 (1000 decimal)
CALL    SUB16B       ; subtrai 1000 decimal do valor binário
BTFS S STATUS,C      ; testa se o resultado é positivo ou zero
GOTO   BIN16DEC_4     ; não ? então vai para a próxima fase
INCF   DISP3,F        ; sim ? então incrementa as milhares
GOTO   BIN16DEC_3     ; reinicia o ciclo
BIN16DEC_4:
MOVF    TEMPO0,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 binário ...
MOVWF   TEMPO0        ; ... para TEMPO0

```

```

MOVF    A1,W           ; copia o MSB binário ...
MOVWF   TEMP1          ; ... para TEMP1
MOVLW   D'100'
MOVWF   B0              ;
CLRF    B1              ; B=100 decimal
CALL    SUB16B          ; subtrai 100 decimal do valor binário (A)
BTFS S STATUS,C        ; testa se o resultado é positivo ou zero
GOTO    BIN16DEC_6      ; não ? então vai para a próxima fase
INCF    DISP2,F         ; sim ? então incrementa as centenas
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 binário ...
MOVWF   TEMP0          ; ... para TEMP0
MOVF    A1,W           ; copia o MSB binário ...
MOVWF   TEMP1          ; ... para TEMP1
MOVLW   D'10'
MOVWF   B0              ;
CLRF    B1              ; B=10 decimal
CALL    SUB16B          ; subtrai 10 decimal do valor binário (A)
BTFS S STATUS,C        ; testa se o resultado é positivo ou zero
GOTO    BIN16DEC_8      ; não ? então vai para fase final
INCF    DISP1,F         ; sim ? então incrementa as dezenas
GOTO    BIN16DEC_7      ; reinicia o ciclo
BIN16DEC_8:
MOVF    TEMP0,W         ; copia o valor temporário restante ...
MOVWF   DISP0            ; ... para as unidades
RETURN
SUB16B:
MOVF    B0,W           ; retorna
SUBWF   A0,F           ;
MOVLW   0x01
BTFS S STATUS,C
SUBWF   A1,F           ;
BTFS S STATUS,C
GOTO    EMPRESTA
MOVF    B1,W           ;
SUBWF   A1,F           ;
FIM_SUB16B:
MOVF    A0,F           ;
BTFS C STATUS,Z
MOVF    A1,F           ;
RETURN
EMPRESTA:
MOVF    B1,W           ;
SUBWF   A1,F           ;
BCF    STATUS,C
GOTO    FIM_SUB16B
***** Aqui se inicia o programa. Primeiramente as variáveis e dispositivos internos
; são inicializados ...
INICIO:
MOVLW   B'00000111'
MOVWF   CMCON          ; configura os comparadores analógicos para modo 7
BANKSEL TRISA           ;
CLRF    TRISA          ; configura a porta A como saída
CLRF    TRISB          ; configura a porta B como saída
BSF     TRISA,5         ; RA5 é entrada
BSF     TRISB,6         ; RB6 é entrada
BANKSEL PORTB          ;
MOVLW   B'11111111'
MOVWF   PORTA          ; desliga os catodos dos ...
; ... displays

```

```
CLRF    DISP0      ; apaga as variáveis
CLRF    DISP1      ;
CLRF    DISP2      ;
CLRF    DISP3      ;
CLRF    DISP4      ;
CLRF    DISP5      ;
CLRF    DISP6      ;
CLRF    FT4       ;
CLRF    FT5       ;
CLRF    FT6       ;
BCF     ATUALIZA   ; apaga o flag de notificação de atualização
BCF     MODO       ; seleciona o modo 0 (frequencímetro)
MOVLW   T0_VAL    ;
MOVWF   T0_CONTA   ;

; a temporização de 1 segundo é conseguida utilizando-se o timer 0: ele é
; programado para utilizar o prescaler com divisão 1:256, além disso, o timer 0
; é inicializado com o valor 39 decimal, o que faz com ele realize apenas
; 217 contagens por ciclo. Desta forma teremos um total de 4Mhz/4/256/217=18,0011
; interrupções por segundo. A rotina de tratamento do timer 0 deve então pro-
; videnciar a contagem até 18, antes de efetuar a totalização dos valores.

MOVLW   B'00000011'; configura o timer 1 para clock externo, presca-
MOVWF   T1CON     ; ler 1:1 e modo síncrono
BANKSEL OPTION_REG ;
MOVLW   B'10000111' ; configura o timer 0 para clock interno com
MOVWF   OPTION_REG ; prescaler dividindo por 256
MOVLW   B'00000001' ;
MOVWF   PIE1      ;
BANKSEL INTCON    ;
MOVLW   B'11100000' ;
MOVWF   INTCON    ;
MOVLW   0xD8       ; inicializa o timer 1 ...
MOVWF   TMR1H    ; ...
MOVLW   0xF0       ; ...
MOVWF   TMR1L     ; com o valor 0xD8F0 (ou 55536 decimal)
MOVLW   T0_INI    ;
MOVWF   TMR0      ;

; no loop principal o programa permanecerá apresentando os valores no display
LOOP_PRINCIPAL:
    BTFSC  ATUALIZA   ; verifica se o valor deve ser atualizado
    CALL   ATUALIZACAO ; sim ? então atualiza os valores
    CALL   DISPLAY    ; mostra display
; Agora verifica qual a posição da chave de seleção de modo, e seleciona o modo
; de acordo.
; Se RA5=1 modo contador, neste caso o flag MODO é setado
; Se RA5=0 modo freqüencímetro, neste caso o flag MODO é apagado
    BCF    MODO       ; apaga MODO
    BTFSC  PORTA,5    ;
    BSF    MODO       ; se RA5=1, seta MODO
    GOTO   LOOP_PRINCIPAL ; retorna para o LOOP_PRINCIPAL
END
```

## APÊNDICE

A

**Bancos de Memória****Banco 0**

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valor no reset POR	Valor nos outros reset
Banco 0											
00h	INDF									xxxx xxxx	xxxx xxxx
		Registrador de acesso indexado (indireto). Acessa o endereço especificado pelo registrador FSR. O registrador INDF não existe fisicamente									
01h	TMRO									xxxx xxxx	aaaa aaaa
02h	PCL									0000 0000	0000 0000
03h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	000q quuu
04h	FSR									xxxx xxxx	aaaa aaaa
		Ponteiro de acesso indireto à memória									
05h	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxx 0000	xxxx 0000
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	aaaa aaaa
07h	Não implementado									-	-
08h	Não implementado									-	-
09h	Não implementado									-	-
0Ah	PCLATH									--0 0000	---0 0000
0Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	EEIE	CMIF	RCIF	TXIF	-	CCP1IF	TMR2IF	TMR1IF	0000 -000	0000 -000
0Dh	Não implementado									-	-
0Eh	TMR1L									xxxx xxxx	aaaa aaaa
		Registrador de acesso ao byte menos significativo (LSB) do timer 1									
0Fh	TMR1H									xxxx xxxx	aaaa aaaa
10h	T1CON			T1CKPS1	TICKPS0	T10SCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000	--uu uuuu
11h	TMR2									0000 0000	0000 0000
12h	T2CON		TOUTPS 3	TOUTPS2	TOUTPS1	TOUTPS0	TMR20N	T2CKPS0	T2CKPS0	-000 0000	-uuu uuuu
13h	Não implementado									-	-
14h	Não implementado									-	-
15h	CCPR1L									xxxx xxxx	aaaa aaaa
16h	CCPR1H									xxxx xxxx	aaaa aaaa
17h	CCP1CON			CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG									0000 0000	0000 0000
1Ah	RCREG									0000 0000	0000 0000
1Bh	Não implementado									-	-
1Ch	Não implementado									-	-
1Dh	Não implementado									-	-
1Eh	Não implementado									-	-
1Fh	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0000	0000 0000

**Tabela A.1 - Não implementado (lido como '0'), u - inalterado, x - desconhecido, q - depende de condição. Os endereços sombreados não estão implementados.**

**Banco 1**

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valor no reset POR	Valor nos outros reset
Banco 1											
80h	INDF									xxxx xxxx	xxxx xxxx
81h	OPTION	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82h	PCL									0000 0000	0000 0000
83h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1XXX	000q quuu
84h	FSR									xxxx xxxx	uuuu uuuu
85h	TRISA	TRISA7	TRISA6		TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	11-1 1111	11-1 1111
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
87h	Não implementado									--	--
88h	Não implementado									--	--
89h	Não implementado									--	--
8Ah	PCLATH									---0 0000	---0 0000
8Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE		CCP1IE	TMR2IE	TMR1IE	0000 -000	0000 -000
8Dh	Não implementado									--	--
8Eh	PCON					OSCF			POR	BOD	---- 1-0x
8Fh	Não implementado										
90h	Não implementado										
91h	Não implementado										
92h	PR2									11111111	11111111
93h	Não implementado									--	--
94h	Não implementado									--	--
95h	Não implementado									--	--
96h	Não implementado									--	--
97h	Não implementado									--	--
98h	TXSTA	CSRC	TX9	TXEN	SYNC		BRGH	TRMT	TX9D	0000 -010	0000 -010
99H	SPBRG									0000 0000	0000 0000
9Ah	EEDATA									xxxx xxxx	uuuu uuuu
9Bh	EEADR									xxxx xxxx	uuuu uuuu
9Ch	EECON1					WRERR	WREN	WR	RD	---- x000	---- q000
9Dh	EECON2									----	----
9Eh	Não implementado										
9Fh	VRCN	VRE	VROE	VRR	-	VR3	VR2	VR1	VR0	000- 0000	000- 0000

**Tabela A.2 - Não implementado (lido como '0'), u - inalterado, x - desconhecido, q - depende de condição. Os endereços sombreados não estão implementados.**

**Banco 2**

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valor no reset POR	Valor nos outros reset
Banco 2											
100h	INDF									xxxx xxxx	xxxx xxxx
101h	TMRO	RBU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
102h	PCL									0000 0000	0000 0000
103h	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1XXX	000q quuu
104h	FSR									xxxx xxxx	uuuu uuuu
105h	Não implementado										
106h	PORT B	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
107h	Não implementado										
108h	Não implementado										
109h	Não implementado										
10Ah	PCLATH	-	-	-						--0 0000	--0 0000
10Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
10Ch	Não implementado										
10Dh	Não implementado										
10Eh	Não implementado										
10Fh	Não implementado										
110h	Não implementado										
111h	Não implementado										
112h	Não implementado										
113h	Não implementado										
114h	Não implementado										
115h	Não implementado										
116h	Não implementado										
117h	Não implementado										
118h	Não implementado										
119h	Não implementado										
11Ah	Não implementado										
11Bh	Não implementado										
11Ch	Não implementado										
11Dh	Não implementado										
11Eh	Não implementado										
11Fh	Não implementado										

**Tabela A.3 - Não implementado (lido como '0'), u - inalterado, x - desconhecido, q - depende de condição. Os endereços sombreados não estão implementados.**

**Banco 3**

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valor no reset POR	Valor nos outros reset
Banco 3											
180h	INDF									xxxx xxxx	xxxx xxxx
181h	OPTION	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
182h	PCL									0000 0000	0000 0000
183h	STATUS	IRP	RP1	RPO	TO	PD	Z	DC	C	0001 1XXX	000q quuu
184h	FSR									xxxx xxxx	uuuu uuuu
185h											
186h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
187h											
188h											
189h											
18Ah	PCLATH									-0 0000	-0 0000
18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
18Ch											
18Dh											
18Eh											
18Fh											
190h											
191h											
192h											
193h											
194h											
195h											
196h											
197h											
198h											
199h											
19Ah											
19Bh											
19Ch											
19Dh											
19Eh											
19Fh											

**Tabela A.4 - Não implementado (lido como '0'), u - inalterado, x - desconhecido, q - depende de condição. Os endereços sombreados não estão implementados.**

## APÊNDICE

B

**Tabelas de Baud Rate****Modo Síncrono**

Velocidade (Kbps)	Clock = 20 Mhz			Clock = 16 Mhz			Clock = 10 Mhz		
	Clock = Velocidade Real	Erro (%)	SPBRG	Clock = Velocidade Real	Erro (%)	SPBRG	Clock = Velocidade Real	Erro (%)	SPBRG
0,3	-	-	-	-	-	-	-	-	-
0,6	-	-	-	-	-	-	-	-	-
1,2	-	-	-	-	-	-	-	-	-
2,4	-	-	-	-	-	-	-	-	-
4,8	-	-	-	-	-	-	-	-	-
9,6	-	-	-	-	-	-	9,77	1,73	255
19,2	19,53	1,73	255	19,23	0,16	207	19,23	0,16	129
38,4	38,46	0,16	129	38,46	0,16	103	38,46	0,16	64
76,8	76,92	0,16	64	76,92	0,16	51	75,76	-1,36	32
96	96,15	0,16	51	95,24	-0,79	41	96,15	0,16	25
115,2	116,28	0,94	42	114,29	-0,79	34	113,64	-1,36	21
230,4	227,27	-1,36	21	235,29	2,12	16	227,27	-1,36	10
300	294,12	-1,96	16	307,69	2,56	12	312,50	4,17	7
500	500,00	0,00	9	500,00	0,00	7	500,00	0,00	4
Máxima	5000,00	-	0	4000,00	-	0	2500,00	-	0
Mínima	19,53	-	255	15,63	-	255	9,77	-	255

Velocidade (Kbps)	Clock = 7,15909 Mhz			Clock = 5,0688 Mhz			Clock = 4 Mhz		
	Clock = Velocidade Real	Erro (%)	SPBRG	Clock = Velocidade Real	Erro (%)	SPBRG	Clock = Velocidade Real	Erro (%)	SPBRG
0,3	-	-	-	-	-	-	-	-	-
0,6	-	-	-	-	-	-	-	-	-
1,2	-	-	-	-	-	-	-	-	-
2,4	-	-	-	-	-	-	-	-	-
4,8	-	-	-	4,95	3,13	255	4,81	0,16	207
9,6	9,62	0,23	185	9,60	0,00	131	9,62	0,16	103
19,2	19,24	0,23	92	19,20	0,00	65	19,23	0,16	51
38,4	38,08	-0,83	46	38,40	0,00	32	38,46	0,16	25
76,8	77,82	1,32	22	74,54	-2,94	16	76,92	0,16	12
96	94,20	-1,88	18	97,48	1,54	12	100,00	4,17	9
115,2	111,86	-2,90	15	115,20	0,00	10	111,11	-3,55	8
230,4	223,72	-2,90	7	211,20	-8,33	5	250,00	8,51	3
300	298,30	-0,57	5	316,80	5,60	3	-	-	-
500	-	-	-	-	-	-	500,00	0,00	1
Máxima	1789,77	-	0	1267,20	-	0	1000,00	-	0
Mínima	6,99	-	255	4,95	-	255	3,91	-	255

Velocidade (Kbps)	Clock = 3,57959 Mhz			Clock = 1 Mhz			Clock = 32768 Hz		
	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG
0,3	-	-	-	-	-	-	0,30	1,14	26
0,6	-	-	-	-	-	-	0,59	-2,48	13
1,2	-	-	-	1,20	0,16	207	1,17	-2,48	6
2,4	-	-	-	2,40	0,16	103	-	-	-
4,8	4,81	0,23	185	4,81	0,16	51	-	-	-
9,6	9,62	0,23	92	9,62	0,16	25	-	-	-
19,2	19,04	-0,83	46	19,23	0,16	12	-	-	-
38,4	38,91	1,32	22	35,71	-6,99	6	-	-	-
76,8	74,57	-2,90	11	83,33	8,51	2	-	-	-
96	99,43	3,58	8	-	-	-	-	-	-
115,2	111,86	-2,90	7	125,00	8,51	1	-	-	-
230,4	223,72	-2,90	3	250,00	8,51	0	-	-	-
300	-	-	-	-	-	-	-	-	-
500	-	-	-	-	-	-	-	-	-
Máxima	894,90	-	0	250,00	-	0	8,19	-	0
Mínima	3,50	-	255	0,98	-	255	0,03	-	255

## Modo Assíncrono Baixa Velocidade (BRGH='0')

Velocidade (Kbps)	Clock = 20 Mhz			Clock = 16 Mhz			Clock = 10 Mhz		
	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG
0,3	-	-	-	-	-	-	0,81	1,73	255
0,6	-	-	-	-	-	-	1,20	0,16	129
1,2	1,22	1,73	255	1,20	0,16	207	1,20	0,16	64
2,4	2,40	0,16	129	2,40	0,16	103	2,40	0,16	32
4,8	4,81	0,16	64	4,81	0,16	51	4,73	-1,36	15
9,6	9,47	-1,36	32	9,62	0,16	25	9,77	1,73	7
19,2	19,53	1,73	15	19,23	0,16	12	19,63	1,73	3
38,4	39,06	1,73	7	35,71	-6,99	6	39,06	1,73	1
76,8	78,13	1,73	3	83,33	8,51	2	78,13	1,73	-
96	104,17	8,51	2	-	-	-	-	-	-
115,2	-	-	-	125,00	8,51	1	-	-	-
230,4	-	-	-	250,00	8,51	0	-	-	-
300	-	-	-	-	-	-	-	-	-
500	-	-	-	-	-	-	-	-	-
Máxima	312,50	-	0	250,00	-	0	156,25	-	0
Mínima	1,22	-	255	0,98	-	255	0,61	-	255

Velocidade (Kbps)	Clock = 7,15909 Mhz			Clock = 5,0688 Mhz			Clock = 4 Mhz		
	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG
0,3	-	-	-	0,31	3,13	255	0,30	1,13	205
0,6	0,60	0,23	185	0,60	0,00	131	0,61	1,13	102
1,2	1,20	0,23	92	1,20	0,00	65	1,20	0,16	51
2,4	2,38	-0,83	46	2,40	0,00	32	2,40	0,16	25
4,8	4,86	1,32	22	4,66	-2,94	16	4,81	0,16	12
9,6	9,32	-2,90	11	9,90	3,13	7	8,93	-6,99	6
19,2	18,64	-2,90	5	19,80	3,13	3	20,83	8,51	2
38,4	37,29	-2,90	2	39,60	3,13	1	-	-	-
76,8	-	-	-	79,20	3,13	0	-	-	-
96	-	-	-	-	-	-	-	-	-
115,2	111,86	-2,90	0	-	-	-	-	-	-
230,4	-	-	-	-	-	-	-	-	-
300	-	-	-	-	-	-	-	-	-
500	-	-	-	-	-	-	-	-	-
Máxima	111,86	-	0	79,20	-	0	62,50	-	0
Mínima	0,44	-	255	0,31	-	255	0,24	-	255

Velocidade (Kbps)	Clock = 3,57959 Mhz			Clock = 1 Mhz			Clock = 32768 Hz		
	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG
0,3	0,30	1,32	183	0,30	0,16	51	-	-	-
0,6	0,61	1,32	91	0,60	0,16	25	-	-	-
1,2	1,22	1,32	45	1,20	0,16	12	-	-	-
2,4	2,33	-2,90	23	2,23	-6,99	6	-	-	-
4,8	4,66	-2,90	11	5,21	8,51	2	-	-	-
9,6	9,32	-2,90	5	-	-	-	-	-	-
19,2	18,64	-2,90	2	-	-	-	-	-	-
38,4	-	-	-	-	-	-	-	-	-
76,8	-	-	-	-	-	-	-	-	-
96	-	-	-	-	-	-	-	-	-
115,2	-	-	-	-	-	-	-	-	-
230,4	-	-	-	-	-	-	-	-	-
300	-	-	-	-	-	-	-	-	-
500	-	-	-	-	-	-	-	-	-
Máxima	55,93	-	0	15,63	-	0	0,51	-	0
Mínima	0,22	-	255	0,06	-	255	0,00	-	255

## Modo Assíncrono Alta Velocidade (BRGH='1')

Velocidade (Kbps)	Clock = 20 Mhz			Clock = 16 Mhz			Clock = 10 Mhz		
	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG
0,3	-	-	-	-	-	-	-	-	-
0,6	-	-	-	-	-	-	-	-	-
1,2	-	-	-	-	-	-	-	-	-
2,4	-	-	-	-	-	-	2,44	1,73	255
4,8	4,88	1,73	255	4,81	0,16	207	4,81	0,16	129
9,6	9,62	0,16	129	9,62	0,16	103	9,62	0,16	64
19,2	19,23	0,16	64	19,23	0,16	51	18,94	-1,36	32
38,4	37,88	-1,36	32	38,46	0,16	25	39,06	1,73	15
76,8	78,13	1,73	15	76,92	0,16	12	78,13	1,73	7
96	96,15	0,16	12	100,00	4,17	9	89,29	-6,99	6
115,2	113,64	-1,36	10	111,11	-3,55	8	125,00	8,51	4
230,4	250,00	8,51	4	250,00	8,51	3	208,33	-9,58	2
300	312,50	4,17	3	-	-	-	312,50	4,17	1
500	-	-	-	500,00	0,00	1	-	-	-
Máxima	1250,00	-	0	1000,00	-	0	625,00	-	0
Mínima	4,88	-	255	3,91	-	255	2,44	-	255

Velocidade (Kbps)	Clock = 7,15909 Mhz			Clock = 5,0688 Mhz			Clock = 4 Mhz		
	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG
0,3	-	-	-	-	-	-	-	-	-
0,6	-	-	-	-	-	-	-	-	-
1,2	-	-	-	-	-	-	1,20	0,16	207
2,4	2,41	0,23	185	-	-	-	2,40	0,16	103
4,8	4,81	0,23	92	1,24	-74,22	255	4,81	0,16	51
9,6	9,52	-0,83	46	9,60	0,00	131	9,62	0,16	25
19,2	19,45	1,32	22	19,20	0,00	65	19,23	0,16	12
38,4	37,29	-2,90	11	38,40	0,00	32	35,71	-6,99	6
76,8	74,57	-2,90	5	74,54	-2,94	16	83,33	8,51	2
96	89,49	-6,78	4	97,48	1,54	12	-	-	-
115,2	111,86	-2,90	3	115,20	0,00	10	125,00	8,51	1
230,4	223,72	-2,90	1	211,20	-8,33	5	250,00	8,51	0
300	-	-	-	79,20	-73,60	3	-	-	-
500	-	-	-	-	-	-	-	-	-
Máxima	447,44	-	0	1267,20	-	0	250,00	-	0
Mínima	1,75	-	255	4,95	-	255	0,98	-	255

Velocidade (Kbps)	Clock = 3,57959 Mhz			Clock = 1 Mhz			Clock = 32768 Hz		
	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG	Velocidade Real	Erro (%)	SPBRG
0,3	-	-	-	0,30	0,16	207	0,29	-2,48	6
0,6	-	-	-	0,60	0,16	103	-	-	-
1,2	1,20	0,23	185	1,20	0,16	51	-	-	-
2,4	2,41	0,23	92	2,40	0,16	25	-	-	-
4,8	4,76	-0,83	46	4,81	0,16	12	-	-	-
9,6	9,73	1,32	22	8,93	-6,99	6	-	-	-
19,2	18,64	-2,90	11	20,83	8,51	2	-	-	-
38,4	37,29	-2,90	5	-	-	-	-	-	-
76,8	74,57	-2,90	2	-	-	-	-	-	-
96	-	-	-	-	-	-	-	-	-
115,2	111,86	-2,90	1	-	-	-	-	-	-
230,4	223,72	-2,90	0	-	-	-	-	-	-
300	-	-	-	-	-	-	-	-	-
500	-	-	-	-	-	-	-	-	-
Máxima	223,72	-	0	62,50	-	0	2,05	-	0
Mínima	0,87	-	255	0,24	-	255	0,01	-	255

## APÊNDICE

C

**Conjunto de Instruções 14 Bits**

Mnemônico	Descrição	Ciclos	Código de 14 bits				Flags afetados
			MSb	...	...	LSb	
<b>Operações com registrador orientadas a byte</b>							
ADDWF f,d	Adiciona o valor de W ao valor de f (d=f+W)	1	00	0111	ffff	ffff	C,DC,Z
ANDWF f,d	AND lógico de W com f (d=f and W)	1	00	0101	ffff	ffff	Z
CLRF f	Limpa conteúdo de f (zera bits) (f=0)	1	00	0001	ffff	ffff	Z
CLRW	Limpa conteúdo de W (zera bits) (W=0)	1	00	0001	0xxx	xxxx	Z
COMF f,d	Complementa f (inverte bits) (d=~f)	1	00	1001	ffff	ffff	Z
DECWF f,d	Decrementa f (d=f-1)	1	00	0011	ffff	ffff	Z
DECFSZ f,d	Decrementa f (d=f-1) e pula se igual a zero	1 (2)	00	1011	ffff	ffff	Nenhum
INCF f,d	Incrementa f (d=f+1)	1	00	1010	ffff	ffff	Z
INCFSZ f,d	Incrementa f (d=f+1) e pula se igual a zero	1 (2)	00	1111	ffff	ffff	Nenhum
IORWF f,d	OR lógico entre W e f (d=f or W)	1	00	0100	ffff	ffff	Z
MOVF f,d	Copia o valor de f para o destino d (d=f)	1	00	1000	ffff	ffff	Z
MOVWF f	Copia o valor de W para f (f=W)	1	00	0000	ffff	ffff	Nenhum
NOP	Nenhuma operação	1	00	0000	0x00	0000	Nenhum
RLF f,d	Desloca f uma posição à esquerda	1	00	1101	ffff	ffff	C
RRF f,d	Desloca f uma posição à direita	1	00	1100	ffff	ffff	C
SUBWF f,d	Subtrai W de F (d=f-W)	1	00	0010	ffff	ffff	C,DC,Z
SWAPF f,d	Troca Nibbles de f (xxxxxx -> yyyyxxxx)	1	00	1110	ffff	ffff	Nenhum
XORWF f,d	XOR lógico entre W e f (d=f xor W)	1	00	0110	ffff	ffff	Z
<b>Operações com registrador orientadas a bit</b>							
BCF f,d	Limpa bit b do registrador f	1	01	00bb	bfff	ffff	Nenhum
BSF f,d	Seta bit b do registrador f	1	01	01bb	bfff	ffff	Nenhum
BTFSZ f,d	Testa bit b do registrador f, pula se igual a zero	1 (2)	01	10bb	bfff	ffff	Nenhum
BTFSZ f,d	Testa bit b do registrador f, pula se igual a um	1 (2)	01	11bb	bfff	ffff	Nenhum
<b>Operações de controle e com constantes</b>							
ADDLW k	Soma a constante k ao registrador W (W=W+k)	1	11	111x	kkkk	kkkk	C,DC,Z
ANDLW k	AND lógico entre k e W (W=W and k)	1	11	1001	kkkk	kkkk	Z
CALL k	Chamada da sub-rotina especificada por k	2	10	0kkk	kkkk	kkkk	Nenhum
CLRWDT	Limpa a contagem do watchdog	1	00	000	0110	0100	TO , PD
GOTO k	Desvia o programa para o endereço k	2	10	1kkk	kkkk	kkkk	Nenhum
IORLW k	OR lógico entre k e W (W=W or k)	1	11	1000	kkkk	kkkk	Z
MOVLW k	Copia a constante k para o registrador W (W=k)	1	11	00xx	kkkk	kkkk	Nenhum
RETFIE	Retorno da interrupção (seta GIE para 1)	2	00	0000	0000	1001	Nenhum
RETLW k	Retorna da sub-rotina, copia k para registrador W	2	11	01xx	kkkk	kkkk	Nenhum
RETURN	Retorno de sub-rotina	2	00	0000	0000	1000	Nenhum
SLEEP	Ativa modo de baixa potência	1	00	0000	0110	0011	TO , PD
SUBLW k	Subtrai W de k (W=k-W)	1	11	110x	kkkk	kkkk	C,DC,Z
XORLW k	XOR lógico entre k e W (W=W xor k)	1	11	1010	kkkk	kkkk	Z

## Ciclos

O valor representa a quantidade de ciclos de máquina necessários para a CPU executar a instrução. No PIC as instruções utilizam sempre um ciclo de máquina (ou seja, quatro ciclos de clock), com exceção das instruções de desvio, chamada e retorno de sub-rotina. Essas instruções, como provocam desvio no programa, irão levar dois ciclos de máquina para serem executadas. Já as instruções de desvio condicional levarão dois ciclos para serem executadas quando provocarem o desvio, e um ciclo de máquina se não realizarem desvio.

## Código de 14 Bits

Nesta coluna encontramos a representação binária da instrução, isto é, o código binário gerado pelo montador durante a fase de montagem de um programa e que é efetivamente armazenado na memória de programa do chip.

## Flags Afetados

Esta coluna indica os flags afetados pela execução da instrução. Tais flags estão localizados no registrador STATUS conforme já estudado.

Devemos lembrar que as instruções que afetam os flags fazem isto na última parte do ciclo de máquina. Isto é, após todo o processamento necessário e escrita do resultado no local de destino, então a CPU irá escrever nos flags afetados.

## Diretivas do MPASM

**\_BADRAM** - define endereços inválidos de memória RAM

Utilização: **\_BADRAM <expressão> [<expressão>]**

A diretiva BADRAM é utilizada para definir endereços de memória RAM inválidos, ou seja, os endereços de memória RAM que não estão implementados fisicamente no chip. O uso desta diretiva não é necessário, já que os arquivos de inclusão fornecidos pela Microchip para cada MCU já possuem as definições de memória inválida. Observe ainda que o uso da diretiva BADRAM deve sempre ser precedido pela diretiva MAXRAM.

### **Exemplo**

```
_maxram H'0BF  
_badram H'07'-H'09', H'0D'-H'1E' ;os endereços 0x07 a 0x09 e 0x0D a 0x1E são inválidos  
_badram H'87'-H'89', H'8D', H'8F'-H'9E' ;os endereços 0x87 a 0x89, 0x8D e 0x8F a 0x9E são inválidos
```

**BANKISEL** - gera código para seleção de banco de memória para acesso indireto.

Utilização : **BANKISEL <nome do registrador>**

A diretiva BANKISEL irá fazer com que o montador gere automaticamente o código necessário para o acesso indireto ao registrador especificado. Nos PICs de 14 bits isto é feito por intermédio do bit IRP no registrador STATUS. Nos PICs de 16 bits, a seleção de bancos é feita diretamente pelos registradores FSR.

### **Exemplo**

VAR1	EQU 0x20	; VAR1 está localizada no endereço 0x20 da memória RAM
MOVLW	VAR1	; o endereço do registrador INTCON é armazenado em W (W=0x20)
MOVWF	FSR	; o endereço de VAR1 é passado para o FSR para acesso indireto
BANKISEL	VAR1	; o montador irá gerar uma instrução BCF STATUS, IRP, pois o registrador VAR1
		; está localizado no banco de memória 0
MOVF	INDF, W	; o conteúdo do registrador IND (que no momento é igual ao do VAR1) é
		; armazenado em W

**BANKSEL** - gera código para seleção de banco de memória para acesso direto

Utilização : BANKSEL <nome do registrador>

A diretiva BANKSEL é utilizada para que o compilador gere automaticamente o código necessário para selecionar o banco de memória no local em que reside o registrador. Nos PICs de 14 bits, esta instrução irá alterar os bits RP0 e RP1 no registrador STATUS, de acordo com o endereço do registrador especificado.

**Exemplo**

BANKSEL CMCON ; o montador irá gerar as instruções BCF STATUS,RP0 e BCF STATUS, RP1, para selecionar o banco de memória correto para acesso ao registrador CMCON (no caso o banco 0).

**CBLOCK** - define um bloco de constantes.

Utilização : CBLOCK <endereço>

<expressões>

ENDC

Esta diretiva é utilizada para fazer a definição de uma série de constantes/variáveis sem a atribuição direta de endereço a cada uma delas. O fim do bloco de constantes deve ser especificado com a diretiva ENDC.

**Exemplo**

```
CBLOCK 0x20
        VAR1           ; esta variável está localizada no endereço 0x20
        VAR2           ; VAR2 está localizada no endereço 0x21
        VAR3:2         ; VAR3 possui 2 bytes e está localizada no endereço 0x22
        VAR5:0, VAR5H, VAR5L ; VAR5 está localizada no endereço 0x24, VAR5H no
                           ; endereço 0x24
                           ; e VAR5L no endereço 0x25
        VAR6: Tamanho   ; VAR6 está localizada no endereço 0x26 e possui tamanho
                           ; definido
                           ; pela constante TAMANHO
ENDC
```

**CODE** - define o início de um bloco de programa.

Utilização: <rótulo> CODE <endereço de memória>

A diretiva CODE é utilizada na geração de arquivos de objeto (programas parcialmente montados e que podem ser unidos a outros objetos para formar um programa mais complexo). O <rótulo> define o nome pelo qual será conhecido o bloco de código do objeto. Se não houver um rótulo, o montador assumirá como rótulo

".code". Caso não seja fornecido um endereço para montagem, o LINKER (programa responsável por ligar um objeto ao outro) irá atribuir um endereço durante a "linkagem" dos objetos.

### Exemplo

TESTE	CODE	0x0100
GOTO PTA		

\_CONFIG - define os bits da palavra de configuração do PIC.

Utilização: \_CONFIG <expressão>

A diretiva config é utilizada para programar a palavra de configuração. Observe que o arquivo de inclusão de cada chip fornecido pela Microchip, já contém as definições a serem utilizadas por esta diretiva.

Exemplo:

<u>_CONFIG _BODEN_ON &amp; _CP_OFF &amp; _PWRTE_ON &amp; _WDT_OFF &amp; _LVP_OFF &amp; _MCLRE_OFF &amp; _INTRC_OSC_NOCLKOUT</u>
---

Esta diretiva configura o chip da seguinte forma: reset por Brown-out ativado, proteção de código desligada, timer watchdog, programação por baixa tensão desligada, pino de MCLR desativado e clock interno RC sem saída clock-out.

Em seguida temos as definições encontradas para o PIC 16F628:

<u>_BODEN_ON</u>	EQU	H'3FFF
<u>_BODEN_OFF</u>	EQU	H'3FBF
<u>_CP_ALL</u>	EQU	H'03FF
<u>_CP_75</u>	EQU	H'17FF
<u>_CP_50</u>	EQU	H'2BFF
<u>_CP_OFF</u>	EQU	H'3FFF
<u>_PWRTE_OFF</u>	EQU	H'3FFF
<u>_PWRTE_ON</u>	EQU	H'3FF7
<u>_WDT_ON</u>	EQU	H'3FFF
<u>_WDT_OFF</u>	EQU	H'3FFB
<u>_LVP_ON</u>	EQU	H'3FFF
<u>_LVP_OFF</u>	EQU	H'3F7F
<u>_MCLRE_ON</u>	EQU	H'3FFF
<u>_MCLRE_OFF</u>	EQU	H'3FDF
<u>_ER_OSC_CLKOUT</u>	EQU	H'3FFF
<u>_ER_OSC_NOCLKOUT</u>	EQU	H'3FFE
<u>_INTRC_OSC_CLKOUT</u>	EQU	H'3FFD
<u>_INTRC_OSC_NOCLKOUT</u>	EQU	H'3FFC
<u>_EXTCLK_OSC</u>	EQU	H'3FEF
<u>_LP_OSC</u>	EQU	H'3FEC

_XT_OSC	EQU H'3FED'
_HS_OSC	EQU H'3FEE'

**CONSTANT** - define um valor constante.

Utilização : CONSTANT <expressões>

A diretiva CONSTANT é utilizada para criar definições de valores constantes a serem utilizados pelo assembler durante a montagem do programa.

**Exemplo**

CONSTANT	TAMANHO = 5
CONSTANT UNIDADE = TAMANHO + 2, VALOR = TAMANHO, PADRAO = 1	

**#DEFINE** - cria um símbolo de definição de texto.

Utilização: DEFINE <nome> <texto>

A diretiva define é utilizada para definir um símbolo que será substituído por um texto ou expressão no programa. Desta forma, cada vez que o montador encontrar o símbolo <nome>, vai substituí-lo pelo texto <texto>.

**Exemplo**

#define tecla	PORTE,1
#define alarme_disparado	FLAG,1
#define LED1	PORTEB,0
#define liga_LED1	BSF LED1
#define desliga_LED1	BCF LED1

**DA** - armazena strings (filas) de caracteres na memória ROM (FLASH).

Utilização: DA <string>

Esta diretiva permite que sejam armazenados dados em forma de filas de caracteres na memória de programa do PIC. Os caracteres ASCII (7 bits) são armazenados dois a dois em cada posição de 14 bits de memória. Observe que esta diretiva deve ser precedida de um ORG para definir o ponto inicial da memória em que serão armazenados os caracteres.

**Exemplo**

ORG 0x050	
DA "testando 123",0	
A diretiva irá fazer com que o montador inicie o armazenamento dos caracteres dois a dois começando no endereço 0x050.	
ORG 0x060	
DA 'PIC16F628','JOINVILLE','SC'	

**DATA** - inicializa uma ou mais posições de memória.

Utilização: DATA <dados>

A diretiva DATA é utilizada para inicializar uma ou mais posições da memória de programa com strings de caracteres, valores, constantes, expressões, labels (etiquetas) e objetos reallocáveis. Observe que os dados podem ser expressões, valores, caracteres simples (cada caractere deve ser colocado entre '). Ex.: 'A'), ou filas de caracteres (strings), que devem ser iniciadas e terminadas por aspas (""). Ex.: "TEXTO". No caso de strings, armazena-se um conjunto de dois caracteres em cada posição de memória de 14. bits. Se o número de caracteres for ímpar, a última posição de memória irá conter o último caractere e mais sete bits '0'. Nas séries 12, 14 e 16, o primeiro caractere é armazenado nos sete bits mais significativos da posição de memória. Na série 18, o primeiro caractere é armazenado nos 7 bits menos significativos da posição de memória.

Observe que, tal qual a diretiva DA, a diretiva DATA deve ser precedida de um ORG para definir o ponto inicial da memória em que serão armazenados os caracteres.

**Exemplo**

```
ORG    0x050
DATA   25,CONTADOR,"testando 123",'a','A'
```

A diretiva irá armazenar o valor 25 na posição 0x050 de memória, o valor representado pela constante CONTADOR no endereço 0x051 e assim por diante.

**DB** - inicializa a memória de programa byte a byte.

Utilização: DB <expressões>

A diretiva DB preenche a memória de programa byte a byte. No caso de um número ímpar de expressões, o último byte é preenchido com zero.

**Exemplo**

```
ORG    0x050
DB     25,CONTADOR,'a','A'
```

A diretiva irá armazenar o valor 25 na posição 0x050 de memória, o valor representado pela constante CONTADOR no endereço 0x051 e assim por diante.

**DE** - Inicializa o conteúdo da memória EEPROM interna

Utilização: DE <expressões>

A diretiva DE permite inicializar valores na EEPROM interna do PIC durante a programação dele. Observe que esta diretiva aloca 1 byte por expressão ou caractere. Caso a diretiva DE seja utilizada para inicializar dados na memória de programa, os sete bits mais significativos (MSB) serão colocados em '0'. Observe que para efeitos de programação, o endereço inicial da memória EEPROM interna é 0x2100.

### **Observação**

Nos chips mais antigos com EEPROM interna (como o 12CE518, 12CE519, 12CE673 e 12CE674), não é possível inicializar o conteúdo da EEPROM, pois ela não é acessível durante a programação.

### **Exemplo**

ORG	0x2100
DE	"memória EEPROM",0x12,0x55

A diretiva irá armazenar no endereço 0x00 da memória EEPROM o valor 0x6D (correspondente à letra 'm'), o endereço 0x01 dela irá conter o valor 0x65 (letra 'e'), e assim por diante.

**DT** - define uma tabela na memória de programa.

Utilização: DT <expressões>

A diretiva DT é utilizada para criar uma estrutura de tabela de valores na memória de programa utilizando instruções RETLW. Essa tabela será composta de tantas instruções quantos forem as expressões definidas pela diretiva. Observe que podemos utilizar a diretiva ORG para definir o ponto inicial de montagem da tabela.

### **Exemplo**

ADDWF	PCL,F
DT	1,4,5,8,12,34,67,34,32

Esta diretiva fará com que o montador insira nove instruções RETLW após a instrução ADDWF. Cada RETLW irá conter um valor das expressões. (RETLW 1, RETLW 4, etc.)

**DW** - inicializa posições da memória de programa palavra a palavra

Utilização: DW <expressões>

A diretiva DW funciona de forma parecida com a diretiva DB, mas com a diferença de que as informações são agrupadas para serem armazenadas em cada posição da memória de programa.

### **Exemplo**

ORG	0x050
DW	25,CONTADOR,'a','A'

**ELSE** - bloco a ser executado no caso de a condição testada pelo IF ser falsa

Utilização: ELSE

A diretiva ELSE é utilizada em conjunto com o IF, IFDEF e IFNDEF para determinar a execução de instruções / diretivas alternativas no caso de a condição testada ser falsa.

Maiores detalhes na diretiva IF.

**END** - indica ao montador que o final do programa foi atingido.

Utilização: END.

A diretiva END é utilizada para indicar o fim de um programa. Observe que quaisquer instruções após a diretiva não serão montadas pelo MPASM.

**ENDC** - indica o fim de um bloco de constantes.

Utilização: ENDC

A diretiva ENDC é utilizada para determinar o final de um bloco de definição de constantes iniciado pela diretiva CBLOCK.

**ENDIF** - indica o final de um bloco de teste condicional.

Utilização : ENDIF

A diretiva ENDIF é utilizada para indicar ao montador o final de um bloco de teste condicional.

**ENDM** - indica ao montador que o final de uma macro foi alcançado.

Utilização: ENDM

A diretiva ENDM é utilizada para determinar o final de uma macro.

### **Exemplo**

```
ORG      0x050
DB       25,CONTADOR,'a','A'
```

A diretiva irá armazenar o valor 25 na posição 0x050 de memória, o valor representado pela constante CONTADOR no endereço 0x051 e assim por diante.

**ENDW** - indica o final de um bloco de repetição.

Utilização: ENDW

A diretiva ENDW é utilizada para determinar o final de um bloco de repetição iniciado pela diretiva WHILE.

**EQU** - definição de valores.

Utilização: <nome> EQU <expressão>

A diretiva EQU é utilizada para criar uma definição de valores para o montador. Assim, o montador irá substituir todas as ocorrências de <nome> pelo valor definido

pelo valor de <expressão>. Esta diretiva é muito utilizada para a criação de variáveis nos registradores GPR.

### **Exemplo**

```
VAR_1    EQU 0x10 ; a variável VAR_1 está localizada no registrador GPR de endereço 0x10  
TEMP     EQU 0x11 ; a variável TEMP está localizada no registrador GPR de endereço 0x11
```

**ERROR** - gera uma mensagem de erro para o usuário.

Utilização: ERROR <texto>

A diretiva ERROR é utilizada para gerar uma mensagem de erro durante a montagem do programa. Essa mensagem pode conter até 80 caracteres e será apresentada após a montagem do programa na janela "Build Results".

### **Exemplo**

```
TESTE MACRO ARG1  
    IF ARG1 >= 55      ; Se o valor de ARG1 for maior ou igual a 55, gera mensagem de erro.  
        ERROR "ERRO - Argumento fora do alcance"  
    ENDIF  
ENDM
```

**ERRORLEVEL** - determina o nível de geração das mensagens.

Utilização: ERRORLEVEL 0/1/2[ , +/- <número>]

A diretiva ERRORLEVEL é utilizada para estabelecer que tipos de mensagem serão mostrados ao usuário durante a compilação.

Parâmetro	Definição
0	Exibe erros, alertas e mensagens
1	Exibe apenas erros e alertas
2	Exibe somente erros
+ <número>	Habilita a mensagem <número>
- <número>	Desabilita a mensagem <número>

### **Exemplo**

```
ERRORLEVEL 2
```

**EXITM** - força a finalização imediata de uma macro.

Utilização: EXITM

A diretiva EXITM, quando executada pelo montador, força a finalização de uma macro, uma saída prematura dela. Possui os mesmos efeitos de uma diretiva ENDM.

**EXPAND** - adiciona o código completo das macros ao arquivo de listagem do programa.

Utilização: EXPAND

A diretiva EXPAND é utilizada para que o código completo das macros seja adicionado ao arquivo de listagem do programa. Esse arquivo possui extensão 'LST' e é utilizado para verificar o código final, símbolos, etc. gerados pelo montador. A diretiva NOEXPAND é utilizada para obter o efeito contrário, isto é, fazer com que no arquivo de listagem não apareça o código da macro; apenas a sua chamada.

**EXTERN** - declara os símbolos definidos em outros objetos.

Utilização: EXTERN <nome>[,<nome>]

A diretiva EXTERN é utilizada para indicar que os símbolos declarados estão definidos em outros arquivos de objeto que serão ligados ao atual posteriormente. Para que um símbolo esteja disponível a todos os objetos, deve ser declarado como GLOBAL.

**Exemplo**

```
EXTERN MULTIPLICA  
...  
...  
CALL MULTIPLICA
```

**FILL** - preenche a memória com um valor ou instrução.

Utilização: FILL <expressão>,<quantidade>

A diretiva FILL preenche a memória de programa com um valor determinado por <expressão> e por <quantidade> endereços. Observe que a expressão pode conter uma instrução assembly, desde que a ela esteja entre parênteses.

**Exemplo**

```
ORG 0x50  
FILL 0x0001, 10 ; preenche dez posições de memória a partir do endereço 0x50 com o valor  
; 0x0001  
FILL (SLEEP), 5 ; preenche cinco posições de memória a partir do endereço atual com a  
; instrução  
; : SLEEP  
FILL 0x0000, NUM ; preenche NUM posições de memória a partir do endereço atual com o valor  
; 0x0000
```

**GLOBAL** - declara símbolos acessíveis por outros objetos.

Utilização: GLOBAL <nome>[,<nome>]

A diretiva GLOBAL é utilizada para definir símbolos dentro do objeto atual que serão acessíveis externamente por qualquer outro objeto.

**Exemplo**

```
UDATA
    VAR1      RES 1
    VAR2      RES 1
    RESULTADO RES 1
    GLOBAL    VAR1,VAR2,RESULTADO,MULTIPLICA
MULTIPLICA CODE
    CLRF      RESULTADO
    ...
    ...
    RETURN
```

**\_IDLOCS** - define valores de identificação do dispositivo.

Utilização: \_IDLOCS <expressão>

Esta diretiva é utilizada para programar as posições de identificação do chip (endereço de memória 0x2000 a 0x2003). A Microchip recomenda que somente se utilizem os quatro bits menos significativos de cada posição de identificação.

O resultado da avaliação da <expressão> é armazenado com o MSB no endereço 0x2000 consecutivamente até que o dígito LSB da expressão seja armazenado no endereço 0x2003.

**Exemplo**

```
ORG 0x2000
_IDLOCS 0x0001 ; armazena 0 no endereço 0x2000,0x2001 e 0x2002 e 1 no endereço 0x2003.
```

**IDATA** - declara um conjunto de dados inicializados.

Utilização: <nome> IDATA [<endereço da memória RAM>]

A diretiva IDATA permite que se declarem dados inicializados para serem utilizados com objetos.

**Exemplo**

```
IDATA
    RPM_MAX  RES 2
    RPM_MIN  RES 2
```

**IF** - testa uma condição e executa um bloco se ela for verdadeira.

Utilização: IF <expressão>

BLOCO1

ELSE

BLOCO2

ENDIF

Esta diretiva é utilizada para criar blocos de montagem condicional. Caso a condição determinada por <expressão> seja verdadeira, será montado o BLOCO1; caso seja falsa, será montado o BLOCO2.

### **Exemplo**

IF REG>5

(instruções para o caso de REG ser maior que 5)

ELSE

(instruções para o caso de REG ser menor ou igual a 5)

ENDIF

**IFDEF** - testa a existência de um símbolo.

Utilização: IFDEF <nome>

A diretiva IFDEF é utilizada para determinar se existe um símbolo definido com o nome <nome>. Caso exista, será executado o código entre o IFDEF e a diretiva ENDIF.

**IFNDEF** - testa a não-existência de um símbolo.

Utilização: IFNDEF <nome>

A diretiva IFNDEF é utilizada para determinar se não existe um símbolo definido com o nome <nome>. Caso ele não exista, será executado o código entre o IFNDEF e a diretiva ENDIF.

**#INCLUDE** - inclui na montagem um arquivo de definições.

Utilização: #INCLUDE <nome do arquivo> ou "nome do arquivo"

A diretiva INCLUDE é utilizada para indicar ao montador que deve ser incluído um arquivo de definições. Arquivos de definições têm o propósito de definir símbolos utilizados em um determinado processador ou sub-rotina.

### **Exemplo**

#INCLUDE <P16F628.INC>

**LIST** - define opções para o arquivo de listagem.

Utilização: LIST <opção>[, <opção2>, etc.]

Esta diretiva é utilizada para comandar a montagem do programa, alterar a formatação do arquivo de listagem gerado, etc.

Opção	Valor padrão	Descrição
b=nnn	8	Define o espaço das tabulações
c=nnn	132	Define a largura em colunas
f=<formato>	INHX8M	Define o formato do arquivo de saída. Podemos selecionar entre três formatos: INHX32, INHX8M ou INHX8S
free	fixed	Interpretador em formato livre
fixed	fixed	Interpretador em formato fixo
mm=ON/OFF	ON	Apresenta mapa de memória no arquivo de listagem
n=nnn	60	Define o número de linhas por página
p=<tipo>	---	Define o tipo de processador
r=<radix>	HEX	Define a base numérica padrão (HEX, DEC ou OCT)
st=ON/OFF	ON	Mostra a tabela de símbolos no arquivo de listagem
t=ON/OFF	OFF	Trunca linhas maiores que o valor definido de colunas
w=0/1/2	0	Define o nível de mensagens (Veja ERRORLEVEL)
x=ON/OFF	ON	Define se o conteúdo das macros será mostrado ou não no arquivo de listagem

### Exemplo

```
LIST P=16F628, r=HEX, x=ON
```

**LOCAL** - define uma variável acessível apenas dentro da macro.

Utilização: LOCAL <nome>[, <nome>]

A diretiva LOCAL é utilizada para definir variáveis que estarão disponíveis apenas dentro da macro, não alterando o valor de variáveis externas à macro, mesmo que existam duas variáveis (uma local e outra global).

### Exemplo

```
VAL    EQU    0x20
TEMP   EQU    0x21
...
...
TESTE  MACRO
        VAR1
        LOCAL TEMP
        TEMP   SET    0x10
ENDM
```

Neste exemplo a variável TEMP somente possuirá o valor 0x10 dentro da macro TESTE. Fora dela o seu valor será igual a 0x21.

**MACRO** - define uma macro.

Utilização: <nome> MACRO [<argumento1>, etc.]

A diretiva MACRO permite a criação de macrofunções, ou seja, seqüências de instruções que são inseridas no programa quando invocadas. Repare que ao contrário das sub-rotinas, que podem ser chamadas por instruções CALL, as macros fazem com que seu código seja inserido, no instante da montagem, no local em que foram chamadas. As macros podem também ser armazenadas em arquivos separados e incluídas na montagem por meio da diretiva INCLUDE, além disso, podem possuir variáveis locais, chamar outras macros, inclusive recursivamente. As macros podem ainda ser utilizadas para automatizar tarefas repetitivas na programação, como: checagem de banco de registrador, montagem condicional, etc.

**Exemplo**

```
SALVA_CONTEXTO MACRO
    MOVWF    W_TEMP      ; SALVA O CONTEÚDO DO W EM W_TEMP
    SWAPF    STATUS,W    ; SALVA O CONTEÚDO DO STATUS ...
    MOVWF    STATUS_TEMP ; ... EM STATUS_TEMP
ENDM
```

**\_MAXRAM** - define o endereço final da memória RAM.

Utilização: \_MAXRAM <expressão>

A diretiva MAXRAM é utilizada para definir o endereço final da memória RAM do processador. O uso desta diretiva pode ser dispensado em virtude de que ela já está definida nos arquivos de inclusão da Microchip.

**MESSG** - gera uma mensagem para o usuário durante a montagem.

Utilização: MESSG "texto"

A diretiva MESSG é utilizada normalmente dentro de macros para informar ao usuário determinados eventos de montagem. O tamanho máximo da string "texto" é de 80 caracteres.

**Exemplo**

```
IF POSICAO > 100      ; Se o valor de POSICAO for maior que 100, gera mensagem
    MESSG "A POSIÇÃO É MAIOR QUE 100"
ENDIF
```

**NOEXPAND** - não expande o código das macros.

Utilização: NOEXPAND

A diretiva NOEXPAND determina ao montador que não adicione o código das macros ao arquivo de listagem.

**NOLIST** - desliga a geração do arquivo de listagem.

Utilização: NOLIST

A diretiva NOLIST determina ao montador que não gere o arquivo de listagem.

**ORG** - esta diretiva é usada para especificar o endereço a partir do qual o programa será montado.

**PAGE** - insere uma quebra de página no arquivo de listagem.

Utilização: PAGE

A diretiva PAGE é utilizada para fazer a separação de páginas no arquivo de listagem.

**PAGESEL** - seleciona a página da memória de programa.

Utilização: PAGESEL <destino>

A diretiva PAGESEL é utilizada para gerar automaticamente as instruções necessárias para programar o registrador PCLATH, de forma que os desvios realizados por instruções CALL e GOTO funcionem adequadamente.

Observe que nos dispositivos que possuem 2kword ou menos de memória de programa, não é gerado código, já que tais dispositivos não necessitam de intervenções no registrador PCLATH para chamadas GOTO e CALL.

### **Exemplo**

```
ORG      0x0000
MOVLW    0xA
...
PAGESEL  PONTO_A
CALL     PONTO_A
```

```
...
ORG      0x1000
PONTO_A:
BCF     PORTA,0
...
RETURN
```

**PROCESSOR** - especifica ao montador qual o processador utilizado.

Utilização: PROCESSOR <modelo>

A diretiva PROCESSOR é utilizada para especificar o tipo do processador para o qual deve ser montado o programa.

**RADIX** - especifica a base numérica padrão a ser utilizada.

Utilização: RADIX <base>

A diretiva RADIX é utilizada para determinar a base numérica a ser utilizada como padrão durante o programa.

É possível selecionar entre três bases numéricas:

- ◆ HEX - para base hexadecimal;
- ◆ DEC - para base decimal;
- ◆ OCT - para base octal.

Observe que é possível ao programador especificar a qualquer momento a base numérica utilizando as seguintes definições:

Base	Definição
Binário	B'XXXXXX'
Octal	O'XX'
Decimal	D'XX' .XX
Hexadecimal	H'XX' 0xXX
ASCII	A'XX'

**RES** - reserva memória.

Utilização: [<nome>] RES <tamanho>

A diretiva RES é utilizada para reservar posições de memória RAM para uso posterior.

O bloco reservado inicia-se na posição atual e possui <tamanho> bytes de memória.

**SET** - define uma variável para o montador.

Utilização: <nome> SET <expressão>

A diretiva SET é utilizada para criar variáveis a serem utilizadas pelo montador durante a montagem do programa.

**SPACE** - insere linhas em branco no arquivo de listagem.

Utilização: SPACE <expressão>

A diretiva SPACE insere uma quantidade definida por <expressão> de linhas em branco ao arquivo de listagem.

**SUBTITLE** - define o subtítulo do programa.

Utilização: SUBTITLE "<texto>"

A diretiva SUBTITLE é utilizada para especificar o subtítulo do programa. Esse subtítulo é apresentado após o título do programa no arquivo de listagem.

**TITLE** - especifica o título do programa.

Utilização: TITLE "<texto>"

A diretiva TITLE é utilizada para especificar o título do programa para o montador. Esse título é apresentado na primeira linha do arquivo de listagem.

**UDATA** - Declara o início de uma seção de dados não inicializados para objetos

Utilização: [<nome>] UDATA [<endereço RAM >]

A diretiva UDATA é utilizada em objetos para declarar uma área cujos dados não são inicializados. Essa seção tem início no endereço da memória especificado por <endereço RAM >. Caso ele não seja especificado, é utilizado o endereço 0.

**UDATA\_OVR** - declara o início de uma seção de dados sobrecarregados.

Utilização: [<nome>] UDATA\_OVR [<endereço RAM>]

A diretiva UDATA\_OVR é utilizada em objetos para declarar uma seção de dados sobrecarregados, ou seja, dados (variáveis) que vão compartilhar os mesmos endereços. Este tipo de dado é muito interessante quando se definem muitas variáveis temporárias em um programa.

**UDATA\_SHR** - declara o início de uma seção de dados compartilhados.

Utilização: [<nome>] UDATA\_SHR [<endereço RAM>]

A diretiva UDATA\_SHR é utilizada em objetos para definir uma seção de dados que podem ser compartilhados entre diversos bancos de memória do PIC. Essa diretiva faz uso da área de registradores GPR espelhados disponível em alguns PICs.

**#UNDEFINE** - desfaz uma definição de símbolo já declarada

Utilização: # UNDEFINE <nome>

Esta diretiva é utilizada para desfazer uma definição declarada anteriormente no programa.

**VARIABLE** - definição de variável (ou variáveis).

Utilização: VARIABLE <nome> [= <expressão>, <nome> = <expressão>]

Esta diretiva é utilizada para definir variáveis a serem utilizadas durante a montagem do programa. Ela não gera qualquer código binário para o PIC.

Esta diretiva é similar à diretiva SET, mas ao contrário desta, não necessita de uma atribuição de valor para a variável, ou seja, permite apenas a declaração da variável.

**WHILE** - laço condicional enquanto a condição é verdadeira.

Utilização: WHILE <expressão>

...

...

ENDW

As diretivas WHILE e ENDW são utilizadas para definir um loop (laço) condicional que irá se repetir enquanto a condição estipulada pela <expressão> for avaliada como verdadeira. Note que as instruções compreendidas entre a diretiva WHILE e ENDW serão executadas a cada iteração do loop.

## Pinagem dos PICs

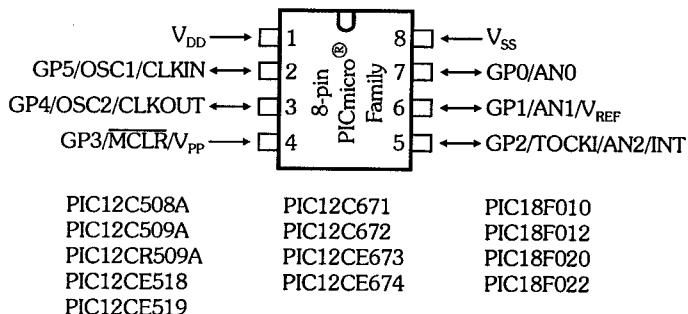


Figura E.1

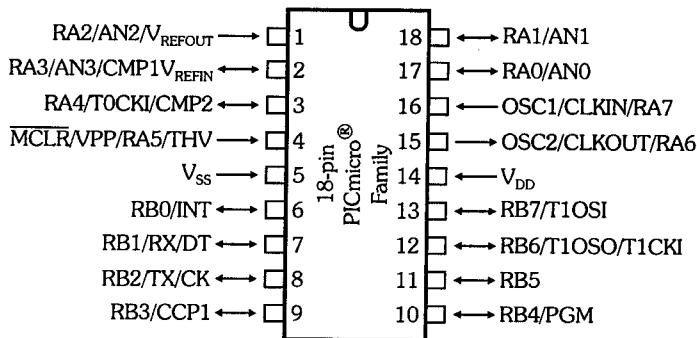
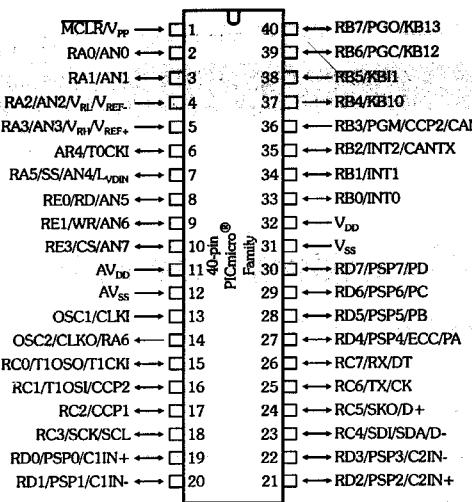
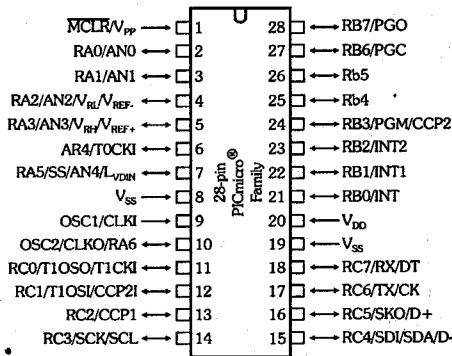


Figura E.2



PIC16CR65	PIC16F77	PIC16F877
PIC16C65B	PIC16C765	PIC16F877A
PIC16C67	PIC16C774	PIC18C442
PIC16C662	PIC16F871	PIC18C452
PIC16C74B	PIC16F874	PIC16F442
PIC16F74	PIC16F874A	PIC18F452
		PIC18F458

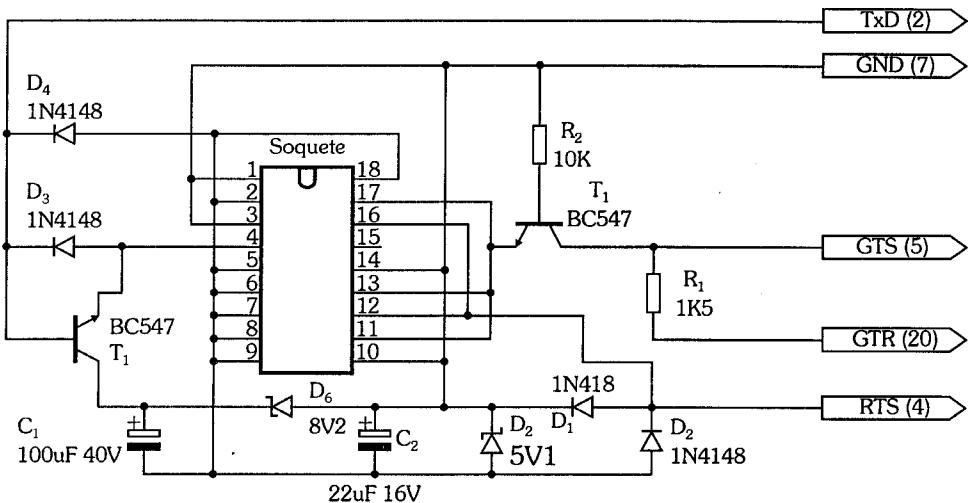
**Figura E.3**



PIC16C62B	PIC16F73	PIC16F873A
PIC16CR63	PIC16F76	PIC16F876
PIC16C63A	PIC16C642	PIC16F876A
PIC16C66	PIC16C745	PIC18C242
PIC16CR72	PIC16C773	PIC18C252
PIC16F72A	PIC16F870	PIC18F242
PIC16C73B	PIC16F872	PIC18F252
PIC16C76	PIC16F873	PIC18F258

**Figura E.4**

## Programador JDM



**Figura F.1**

Este diagrama é o circuito do programador conhecido como JDM.

Trata-se de um dispositivo muito simples e de baixíssimo custo que pode ser utilizado para leitura e gravação de PICs de 8 e 18 pinos, além de memórias EEPROM I2C, PICs de 28 pinos e 40 pinos (com uso de adaptador, ou pela modificação física do esquema).

O programador utiliza alguns sinais disponíveis na porta serial de qualquer micro compatível com IBM-PC (porta COM).

O diagrama apresentado refere-se à pinagem do conector DB25.

O software de programação IC-PROG pode ser encontrado em:

<http://www.ic-prog.com>

## Tabelas de Conversão

Decimal	Hexadecimal	Binário	ASCII	Negativo
0	00	00000000	NUL	0
1	01	00000001	SOH	-255
2	02	00000010	STX	-254
3	03	00000011	ETX	-253
4	04	00000100	EOT	-252
5	05	00000101	ENQ	-251
6	06	00000110	ACK	-250
7	07	00000111	BEL-BEEP	-249
8	08	00001000	BS-BACK SPACE	-248
9	09	00001001	HT	-247
10	0A	00001010	LF-LINE FEED	-246
11	0B	00001011	VT	-245
12	0C	00001100	FF	-244
13	0D	00001101	CR-RETURN	-243
14	0E	00001110	SO	-242
15	0F	00001111	SI	-241
16	10	00010000	DLE	-240
17	11	00010001	DC1	-239
18	12	00010010	DC2	-238
19	13	00010011	DC3	-237
20	14	00010100	DC4	-236
21	15	00010101	NAK	-235
22	16	00010110	SYN	-234
23	17	00010111	ETB	-233
24	18	00011000	CAN	-232
25	19	00011001	EM	-231
26	1A	00011010	SUB	-230
27	1B	00011011	ESC-ESCAPE	-229
28	1C	00011100	FS	-228
29	1D	00011101	GS	-227
30	1E	00011110	RS	-226
31	1F	00011111	US	-225
32	20	00100000	SP-ESPAÇO	-224
33	21	00100001	!	-223
34	22	00100010	"	-222
35	23	00100011	#	-221
36	24	00100100	\$	-220
37	25	00100101	%	-219
38	26	00100110	&	-218
39	27	00100111	,	-217
40	28	00101000	(	-216
41	29	00101001	)	-215
42	2A	00101010	*	-214
43	2B	00101011	+	-213
44	2C	00101100	,	-212
45	2D	00101101	-	-211
46	2E	00101110	.	-210
47	2F	00101111	/	-209
48	30	00110000	0	-208
49	31	00110001	1	-207
50	32	00110010	2	-206
51	33	00110011	3	-205

Decimal	Hexadecimal	Binário	ASCII	Negativo
52	34	00110100	4	-204
53	35	00110101	5	-203
54	36	00110110	6	-202
55	37	00110111	7	-201
56	38	00111000	8	-200
57	39	00111001	9	-199
58	3A	00111010	:	-198
59	3B	00111011	;	-197
60	3C	00111100	<	-196
61	3D	00111101	=	-195
62	3E	00111110	>	-194
63	3F	00111111	?	-193
64	40	01000000	@	-192
65	41	01000001	A	-191
66	42	01000010	B	-190
67	43	01000011	C	-189
68	44	01000100	D	-188
69	45	01000101	E	-187
70	46	01000110	F	-186
71	47	01000111	G	-185
72	48	01001000	H	-184
73	49	01001001	I	-183
74	4A	01001010	J	-182
75	4B	01001011	K	-181
76	4C	01001100	L	-180
77	4D	01001101	M	-179
78	4E	01001110	N	-178
79	4F	01001111	O	-177
80	50	01010000	P	-176
81	51	01010001	Q	-175
82	52	01010010	R	-174
83	53	01010011	S	-173
84	54	01010100	T	-172
85	55	01010101	U	-171
86	56	01010110	V	-170
87	57	01010111	W	-169
88	58	01011000	X	-168
89	59	01011001	Y	-167
90	5A	01011010	Z	-166
91	5B	01011011	[	-165
92	5C	01011100	\	-164
93	5D	01011101	]	-163
94	5E	01011110	^	-162
95	5F	01011111	-	-161
96	60	01100000	-	-160
97	61	01100001	a	-159
98	62	01100010	b	-158
99	63	01100011	c	-157
100	64	01100100	d	-156
101	65	01100101	e	-155
102	66	01100110	f	-154
103	67	01100111	g	-153
104	68	01101000	h	-152
105	69	01101001	i	-151
106	6A	01101010	j	-150
107	6B	01101011	k	-149
108	6C	01101100	l	-148
109	6D	01101101	m	-147
110	6E	01101110	n	-146
111	6F	01101111	o	-145
112	70	01110000	p	-144
113	71	01110001	q	-143
114	72	01110010	r	-142
115	73	01110011	s	-141
116	74	01110100	t	-140
117	75	01110101	u	-139
118	76	01110110	v	-138
119	77	01110111	w	-137
120	78	01111000	x	-136
121	79	01111001	y	-135

Decimal	Hexadecimal	Binário	ASCII	Negativo
122	7A	01111010	z	-134
123	7B	01111011	(	-133
124	7C	01111100	)	-132
125	7D	01111101	)	-131
126	7E	01111110	~	-130
127	7F	01111111	DEL-DELETE	-129
128	80	10000000		-128
129	81	1000000001		-127
130	82	100000010		-126
131	83	100000011		-125
132	84	100000100		-124
133	85	100000101		-123
134	86	100000110		-122
135	87	100000111		-121
136	88	100001000		-120
137	89	100001001		-119
138	8A	100001010		-118
139	8B	100001011		-117
140	8C	100001100		-116
141	8D	100001101		-115
142	8E	100001110		-114
143	8F	100001111		-113
144	90	10010000		-112
145	91	10010001		-111
146	92	10010010		-110
147	93	10010011		-109
148	94	10010100		-108
149	95	10010101		-107
150	96	10010110		-106
151	97	10010111		-105
152	98	10011000		-104
153	99	10011001		-103
154	9A	10011010		-102
155	9B	10011011		-101
156	9C	10011100		-100
157	9D	10011101		-99
158	9E	10011110		-98
159	9F	10011111		-97
160	A0	10100000		-96
161	A1	10100001		-95
162	A2	10100010		-94
163	A3	10100011		-93
164	A4	10100100		-92
165	A5	10100101		-91
166	A6	10100110		-90
167	A7	10100111		-89
168	A8	10101000		-88
169	A9	10101001		-87
170	AA	10101010		-86
171	AB	10101011		-85
172	AC	10101100		-84
173	AD	10101101		-83
174	AE	10101110		-82
175	AF	10101111		-81
176	B0	10110000		-80
177	B1	10110001		-79
178	B2	10110010		-78
179	B3	10110011		-77
180	B4	10110100		-76
181	B5	10110101		-75
182	B6	10110110		-74
183	B7	10110111		-73
184	B8	10111000		-72
185	B9	10111001		-71
186	BA	10111010		-70
187	BB	10111011		-69
188	BC	10111100		-68
189	BD	10111101		-67
190	BE	10111110		-66
191	BF	10111111		-65

Decimal	Hexadecimal	Binário	ASCII	Negativo
192	C0	11000000		-64
193	C1	11000001		-63
194	C2	11000010		-62
195	C3	11000011		-61
196	C4	11000100		-60
197	C5	11000101		-59
198	C6	11000110		-58
199	C7	11000111		-57
200	C8	11001000		-56
201	C9	11001001		-55
202	CA	11001010		-54
203	CB	11001011		-53
204	CC	11001100		-52
205	CD	11001101		-51
206	CE	11001110		-50
207	CF	11001111		-49
208	D0	11010000		-48
209	D1	11010001		-47
210	D2	11010010		-46
211	D3	11010011		-45
212	D4	11010100		-44
213	D5	11010101		-43
214	D6	11010110		-42
215	D7	11010111		-41
216	D8	11011000		-40
217	D9	11011001		-39
218	DA	11011010		-38
219	DB	11011011		-37
220	DC	11011100		-36
221	DD	11011101		-35
222	DE	11011110		-34
223	DF	11011111		-33
224	E0	11100000		-32
225	E1	11100001		-31
226	E2	11100010		-30
227	E3	11100011		-29
228	E4	11100100		-28
229	E5	11100101		-27
230	E6	11100110		-26
231	E7	11100111		-25
232	E8	11101000		-24
233	E9	11101001		-23
234	EA	11101010		-22
235	EB	11101011		-21
236	EC	11101100		-20
237	ED	11101101		-19
238	EE	11101110		-18
239	EF	11101111		-17
240	F0	11110000		-16
241	F1	11110001		-15
242	F2	11110010		-14
243	F3	11110011		-13
244	F4	11110100		-12
245	F5	11110101		-11
246	F6	11110110		-10
247	F7	11110111		-9
248	F8	11111000		-8
249	F9	11111001		-7
250	FA	11111010		-6
251	FB	11111011		-5
252	FC	11111100		-4
253	FD	11111101		-3
254	FE	11111110		-2
255	FF	11111111		-1

# Índice Remissivo

## A

acesso indireto à memória, 268  
ADDEN, 172, 177  
adição, 106  
adição de 16 bits, 242  
adição no PIC, 240  
AND, 21  
apagamento, 200, 201  
assembler, 204  
assembly, 204, 212

## B

Baud Rate, 167  
BCD, 32  
BCW, 291  
BOD, 191  
BODEN, 191  
BRGH, 171, 173  
brown-out, 111, 188, 191  
BSW, 290  
buffer, 23, 24, 25, 26

## C

C1OUT, 157  
C2OUT, 157  
CALL, 43  
Captura, 147, 148  
CCP1, 144, 272  
chamada de sub-rotina seletiva, 234  
CIS, 158  
clock externo, 191  
CM0, 158  
CM1, 158  
CM2, 158  
COMBF, 295  
Comparação, 147, 152  
Comparador  
modo 5, 161

Comparador Analógico, 157  
Comparadores  
modo 0, 159  
modo 1, 159  
modo 2, 160  
modo 3, 160  
modo 4, 161  
modo 6, 162

modo 7, 162  
COMW, 290  
contador, 309  
contador de programa, 41  
conversão, 260, 261, 263, 264  
conversão 16F84, 296  
conversão A/D, 297  
conversão de base numérica, 260  
conversor A/D, 275  
conversor Delta-Sigma, 275  
CPU, 17  
CREN, 172  
cristal, 189  
CSRC, 170

## D

DEBOUNCE, 284  
Decodificador, 31, 32, 277, 297  
decodificador BCD, 32  
DECW, 289  
demultiplexador, 31  
display, 277, 297, 309  
interface, 277, 278, 281  
divisão, 252, 253, 255  
divisão no PIC, 252  
DSC, 35  
DSP, 35  
DsPIC, 35

## E

EEPROM, 183, 187, 200  
entrada paralela, 33  
entrada serial, 33  
equações, 258  
erros, 222

## F

fan-out, 23  
FERR, 172  
flags, 195, 240, 244  
C, 105  
DC, 107  
Z, 108  
Flip-flop, 26, 28, 29  
Flip-flop "tipo D", 27  
Flip-flop SR, 26  
Flip-flop SR sincronizado, 27

frequêncímetro, 309

Full-Duplex, 167

## G

GIE, 47

GOTO, 42

## H

Half-duplex, 167

## I

ICD, 209

ICE, 208

IC-Prog, 215

ICSP, 197

INCW, 289

instruções, 57

ADDLW, 70, 240

ADDFWF, 72, 240

afetando o STATUS, 58

ANDLW, 78

ANDWF, 78

BCF, 66

BSF, 68

BTFS, 91

BTFS, 92

CALL, 89

CLRF, 62

CLRW, 63

CLRWDT, 96, 192, 194

DECf, 65

DECFSZ, 93

GOTO, 88

INCF, 64

INCFSZ, 95

IORLW, 79

IORWF, 80

MOVf, 60

MOVLW, 59, 225

MOVWF, 60, 225

NOP, 96

OPTION, 97

RETFIE, 91

RETLW, 90

RETURN, 89

RLF, 84

RMW, 230

RRF, 86

SLEEP, 96, 194

SUBLW, 74, 243

SUBWF, 75, 227, 243

SWAPF, 69

TRIS, 97

XORLW, 81

XORWF, 82, 227

interrupção, 46, 111

ADCIF, 54

ADIF, 54

CCP1IF, 52, 150, 152

CCP2IF, 53

CMIE, 163

CMIF, 53

EEIF, 51, 185

GIE, 47

GPIF, 55

INTCON, 113

INTF, 51, 124

latência, 55

LCDIF, 55

modo SLEEP, 56

OVFIF, 54

PEIE, 48

PIC16F62x, 112

PIE, 114

PIR1, 116

PSPIF, 55

RBIF, 51, 124

RCIF, 53, 176, 177, 181

SSPIF, 54

T0IF, 52, 133

TMR1IF, 52, 137

TMR2IF, 52, 144

TXIF, 53, 175, 179

vetor, 46

interrupções, 111, 235

Desativando o GIE, 235

sub-rotinas, 239

IRP, 109

## J

JPE, 291

JPG, 293

JPGS, 294

JPL, 292

JPLS, 293

JPNE, 292

JPWNZ, 295

JPWZ, 294

## L

latch, 30  
LIFO, 44  
linguagem BASIC, 219  
linguagem C, 218  
linguagem Pascal, 219  
LVP, 187

## M

macro, 239  
macroinstruções, 289  
macros, 289  
MCLRE, 188  
MCU, 18  
memória de programa, 37  
endereçamento, 38  
EPROM, 37  
FLASH, 37  
OTP, 37  
página, 38  
paginação, 38  
ROM, 37  
memória RAM, 39  
bancos de memória, 39  
endereços, 39  
seleção do banco de memória, 39  
MIPS, 36  
mnemônico, 57  
modo SLEEP, 194  
Wake-up, 48  
MOVLW, 59  
MPASM, 212  
MPLAB, 203  
MPU, 18  
multiplexação, 281, 301  
multiplicação, 248, 249, 251  
multiplicação no PIC, 247

## N

NAND, 21  
NO, 23  
NOR, 22

## O

OERR, 172  
OR, 22  
oscilador RC interno, 190

## P

palavra de configuração, 186  
paridade, 288  
pilha, 44  
POP, 45  
PUSH, 44  
sobrecarga, 46  
pinagem, 100  
pinos de I/O  
observações sobre a porta A, 122  
porta B, 123  
RA0, 119  
RA1, 119  
RA2, 119  
RA3, 120  
RA4, 120  
RA5, 121  
RA6, 121  
RA7, 122  
RB0, 125  
RB1, 126  
RB2, 126  
RB3, 127  
RB4, 128  
RB5, 127  
RB6, 129  
RB7, 130  
pipeline, 36  
portas de I/O, 117, 228  
observações sobre a porta B, 130  
porta A, 117  
postscaler, 143  
prescaler, 109, 132, 136, 143, 145, 193  
programação, 197  
projeto, 207, 220  
Proteção de código, 186  
PSA, 193  
pull-up, 110, 124  
pulso, 272  
PWM, 147, 154  
PWRT, 191

## R

raiz quadrada, 256  
recepção assíncrona, 175  
registrator de deslocamento, 33  
registrator PCL, 266  
registradores  
bancos de memória, 39  
CCP1CON, 147

CCPR1H, 148  
CCPR1L, 148  
CMCON, 157  
EEADR, 183  
EECON1, 184  
EEDATA, 183  
FSR, 103  
GPIO, 37  
GPR, 39  
INDF, 103  
INTCON, 113  
mapa de memória, 102  
OPTION\_REG, 109  
PC, 41  
PCL, 41  
PCLATH, 39, 41  
PCON, 110  
PIE1, 114  
PIR1, 116  
PORTA, 118  
PORTB, 124  
PR2, 144  
RCSTA, 171  
SFR, 39  
SPBRG, 173  
STATUS, 105, 241  
T1CON, 138  
TMR0, 132  
TMR1H, 139  
TMR1L, 139  
TMR2CON, 145  
TRISA, 118  
TRISB, 124  
TXSTA, 170  
W, 41  
registradores GPR, 226  
relógio, 301  
reset, 192, 194  
    GPR, 197  
    PC, 195  
    PCON, 195  
registradores SFR, 195  
    STATUS, 195  
ressonador, 189  
RETFIE, 48  
RSR, 170  
RTI, 237  
RX9, 171  
RX9D, 172, 177

**S**  
saída paralela, 33  
saída serial, 33  
SCI, 167  
serial, 167, 197  
simulador, 208  
SPEN, 171  
SREN, 172  
STATUS, 58  
sub-rotinas, 232  
subtração de 16 bits, 245  
subtração no PIC, 243  
SYNC, 170

**T**  
T1CKI, 139  
T1OSCEN, 138, 140  
tabelas, 265  
teclas, 284  
tempo de execução, 211  
timer 0, 131, 269  
timer 1, 136, 270, 273, 309  
timer 2, 143, 271  
TMR1CS, 138, 139  
TMR1IF, 147  
TMR1ON, 138  
TMR2ON, 145  
transmissão assíncrona, 174  
tratamento de interrupções, 237  
TRMT, 171  
TSR, 169  
TX9, 170  
TX9D, 171  
TXEN, 170

**U**  
USART, 167  
endereçamento, 177  
modo de 9 bits, 177  
transmissão síncrona modo mestre, 179

**W**  
wake-up, 194  
Watch, 211  
watchdog, 109, 133, 192  
WDTE, 188

**X**  
XOR, 22

## Referências Bibliográficas

- CIARCIA, S. **Construa o Seu Próprio Microcomputador Z80.** São Paulo: McGraw-Hill, 1984.
- MICROCHIP. **PICmicro: Mid-Range MCU Family Reference Manual.** USA: MICROCHIP, 1997.
- \_\_\_\_\_ **Embedded Control Handbook: Volume 1.** USA: MICROCHIP, 1997.
- \_\_\_\_\_ **Embedded Control Handbook: Volume 2: Math Library.** USA: MICROCHIP, 1997.
- \_\_\_\_\_ **Embedded Control Handbook: Update 2000.** USA: MICROCHIP, 1999.
- \_\_\_\_\_ **PIC16F62X: Datasheet DS40300B.** USA: MICROCHIP, 1999.
- \_\_\_\_\_ **AN700: Make a Delta-Sigma Converter Using a Microcontroller's Analog Comparator Module.** USA: MICROCHIP, 1999.
- NICOLOSI, D. E. C. **Microcontrolador 8051: Detalhado.** São Paulo: Érica, 2001.
- PALMER, M. **AN576: Techniques do Disable Global Interrupts.** USA: MICROCHIP, 1997.
- REVISTA NOVA ELETRÔNICA. Números 47 e 48. São Paulo: Editele, 1981
- SILVA JR., V. P. **Aplicações Práticas do Microcontrolador 8051.** São Paulo: Érica, 1998.
- SOUZA, D. J. **Desbravando o PIC.** São Paulo: Érica, 2000.

### Sites:

- <http://www.piclist.org>  
<http://ee.cleversoul.com/pic.html>  
<http://www.microchip.com>  
<http://www.gnupic.org>  
<http://www.circuitcellar.com/>  
<http://www.myke.com/>  
<http://www.dattalo.com/>  
<http://techref.massmind.org/techref/microchip/piclinks.htm>  
<http://www.geocities.com/SiliconValley/Cable/7772/>  
<http://www.dontronics.com/>

## Marcas Registradas

TRISTATE é uma marca registrada da National Semiconductors.

PIC, PICmicro, MPLAB, MPASM e ICSP são marcas registradas da Microchip Technology.

Todos os demais nomes registrados, marcas registradas, ou direitos de uso citados neste livro pertencem aos seus respectivos proprietários.

MICROCONTROLADORES

# PIC

TÉCNICAS AVANÇADAS

**Este livro é dedicado aos autodidatas e profissionais da área eletrônica que desejam expandir seus conhecimentos na área de programação e projeto com microcontroladores PIC. Abrange desde os conceitos básicos e conjunto de instruções até os periféricos internos, e finalmente um avançado estudo de técnicas de programação e hardware. É um material indispensável na biblioteca de todos os que pretendem aprofundar seus conhecimentos em relação aos microcontroladores PIC.**

**Publicações Érica, Clareza e Objetividade.**

**Invista em você.**  
  
**Visite uma livraria.**



**EDITORIA ÉRICA LTDA.**  
Rua São Gil, 159 • Tatuapé  
CEP: 03401-030 • São Paulo • SP  
Fone: (11) 295-3066 • Fax: (11) 217-4060  
[www.editoraerica.com.br](http://www.editoraerica.com.br)

