

BRINDE

Código-fonte dos sete exemplos
completos dados no capítulo 11
e uma versão de solução para o
projeto proposto do "Dimmer"
disponíveis na INTERNET.



Desbravando o PIC

Ampliado e Atualizado para PIC16F628A

ABPDEA
ABPDEA
ABPDEA
ABPDEA
ABPDEA
ABPDEA
ABPDEA

Associação Brasileira para
a Proteção dos Direitos
Editoriais e Autorais
RESPEITE O AUTOR
NAO FAÇA CÓPIA
www.abpdea.org.br

Seja Nosso 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, você passará a receber 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-sé 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
2. Fax: (11) 6197.4060
3. Carta: Rua São Gil, 159 - Tatuapé - CEP 03401-030 - São Paulo - SP



David José de Souza

Desbravando o PIC Ampliado e Atualizado para PIC16F628A

Editora Érica Ltda.

2005 - 8^a Edição

Conselho Editorial:

Diretor Editorial:	Antonio Marco Vicari Cipelli
Diretor Comercial:	Paulo Roberto Alves
Diretor de Publicidade:	Waldir João Sandrini
Capa:	Maurício S. de França
Editoração:	Ana Luisa Nobrega Cury
Revisão Gramatical:	Marlene Teresa Santin Alves
Revisão Gramatical da 6 ^a ed.:	Vera Lúcia Quintanilha
Revisão Interna:	Érica Regina Pagano
Coordenação e Revisão:	Rosana Ap. Alves dos Santos
Revisão Técnica:	Eduardo César Alves Cruz

Copyright © 2003 da Editora Érica Ltda.

Dados Internacionais de Catalogação na Publicação (CIP)

(Câmara Brasileira do Livro, SP, Brasil)

Souza, David José de, 1971 –

Desbravando o PIC: ampliado e atualizado para PIC 16F628A / David José de Souza. – 6.ed. -- São Paulo: Érica, 2003.

Bibliografia

ISBN 85-7194-867-4

1. PIC (Microcontroladores) I. Título

03-3831

CDD-004.16

Índices para catálogo sistemático

1. PIC: Microcontroladores: Processamento de Dados 004.16

Todos os direitos reservados. Proibida a reprodução total ou parcial, por qualquer meio ou processo, especialmente por sistemas gráficos, microfílmicos, fotográficos, reprográficos, fonográficos, videográficos, 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 indemnizações 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 nenhum com o livro, não garantindo a sua existência nem divulgação. Eventuais erratas estarão disponíveis no site da Editora Érica para download.

Editora Érica Ltda.

Rua São Gil, 159 - Tatuapé

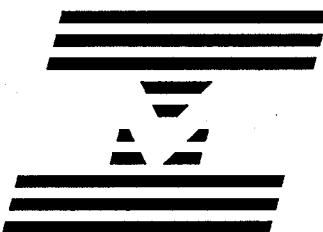
CEP: 03401-030 - São Paulo - SP

Fone: (11) 295-3066 - Fax: (11) 6197-4060

www.editoraerica.com.br

A MOSAICO

A Mosaico nasceu dos ideais e da amizade de engenheiros eletrônicos em meados de 1995. Em busca do aperfeiçoamento de seus conhecimentos e da utilização de suas teorias, resolveram fundar uma empresa experimental. Como sempre, escolher o nome para batizá-la não foi uma tarefa nada fácil. A idéia veio de um antigo projeto, que reunia amigos para escrever um livro de poesia. Como existiriam vários autores, pensou-se em nomear o livro como "Mosaico". Consultando o dicionário Aurélio, veio a confirmação: "Mosaico - qualquer trabalho intelectual ou manual composto de várias partes distintas ou separadas". Foi criada então, oficialmente, a Mosaico Engenharia Eletrônica S/C Ltda.



Com o crescente aumento da demanda na especialização de microcontroladores, criada pela introdução no Brasil de componentes mais flexíveis e baratos, principalmente por parte da Microchip, abriu-se uma nova frente de trabalho. A Mosaico começou então a sua especialização em microcontroladores na segunda metade de 1996. Em 1997, a empresa começou a crescer rapidamente, junto com o mercado desses componentes. EM 1998, entrou na área de treinamentos. Em 2000, aventurou-se no setor de materiais didáticos. Um pouco mais tarde, em 2002, obteve um justo reconhecimento por parte da Microchip, recebendo o título de primeira empresa nacional denominada Consultora Oficial da Microchip no Brasil. É o reflexo dos sonhos e da capacidade desses profissionais no decorrer dos últimos anos.

Somos hoje uma empresa extremamente especializada no projeto de produtos ou sistemas baseados na utilização de microcontroladores, e contamos com excelente Know-how nos produtos da linha PIC. Desenvolvemos projetos eletrônicos tanto de hardware (circuitos e componentes) como de software (programa para o microcontrolador) e nossos serviços já originaram produtos para clientes das mais variadas áreas. São projetos de equipamentos médicos, segurança, alarme, hobby, instrumentação, automação, diversão, coletores de dados e muito mais. Foi toda essa experiência que tornou possível a realização desta obra, repassando um pouco do conhecimento adquirido a outras pessoas.

A Mosaico também tem colaborado bastante para a divulgação e popularização do PIC no Brasil, realizando cursos dedicados e palestras informativas para universitários. Outra realização de destaque é o grande apoio que ela tem dado aos trabalhos de formatura de universidades da região, ajudando muitos jovens a aprofundar seus conhecimentos e enfrentar seus primeiros problemas para o início de uma carreira profissional.

Grupo Mosaico Eletrônica.

Projetos eletrônicos, consultoria, treinamento e materiais didáticos.

Rua Galeão Carvalhal, 125 - Bairro Bela Vista, Santo André - SP

CEP: 09041-400

PABX: (11) 4992.8775

E-mail: mosaico@mosaico-eng.com.br

Site: www.mosaico-eng.com.br

DISTRIBUIDORES

Grupo Mosaico

Consultor Oficial da Microchip no Brasil
E-mail: mosaico@mosaico-eng.com.br
Home page: www.mosaico-eng.com.br

Microchip Technology Inc.

Fabricante do PIC.
2355, W. Chandler Blvd. , Chandler, Arizona - USA
Tel: (01xx480) 768-7200
Fax: (01xx480) 899-9210
Home page: www.microchip.com

Aplicações Eletrônicas Artimar Ltda.

Representante exclusivo Microchip no Brasil.
Rua Marquês de Itu, 70, 10º andar, São Paulo - SP
Tel: (11) 231-0277
Fax: (11) 255-0511
E-mail: artimar@ibm.net
Home page: www.artimar.com.br

Aut-Comp

Revendedor autorizado.
Rua Belgrado, 330, São Paulo - SP
Tel: (11) 6915-7443
Fax: (11) 6915-7579
E-mail: autcomp@uol.com.br

Hitech

Revendedor autorizado.
Rua Branco de Moraes, 489, São Paulo - SP
Tel: (11) 5188-4130
Fax: (11) 5188-4191
E-mail: microchip@hitech.com.br
Home page: www.hitech.com.br

Future Electronics

Revendedor autorizado.
Rua Lusitana, 740, 10º andar, Campinas - SP
Tel: (19) 3737-4100
Fax: (19) 3236-9834
E-mail: future@zaz.com.br

DEDICATÓRIA

Dedico este livro à minha querida esposa.

*Quem retribui os favores será recordado um dia;
e, no momento da queda, encontrará apoio.*

Eclo 3,31

AGRADECIMENTOS

Foi muito gratificante a experiência de escrever este livro. Neste momento penso em agradecer a muitas pessoas, mas fico receoso de esquecer de alguém. Por isso, começo agradecendo a todas as pessoas que eu conheço e que já me aturam há um bom tempo. Entretanto, é lógico que eu não poderia deixar de enfatizar alguns agradecimentos especiais.

Começo com minha família, agradecendo a meus pais José Carlos e Neusa, sem os quais nada do que eu faço seria possível. À minha esposa Eliete, que soube compreender as muitas horas perdidas na frente do computador, os momentos de ausência e de irritação, e mesmo assim continuou me amando. Eu também a amo muito. Ao meu irmão e sócio José Carlos de Souza Júnior, que me deu a oportunidade de conhecer o mundo da eletrônica, e, é claro, à Mosaico Engenharia e toda a sua equipe, que me capacitaram, gerando o conhecimento necessário para uma obra técnica deste nível.

Destaco ainda meus demais sócios: Vanderlei, Nicolás e Gil, que me apoiaram e me suportaram durante todos estes anos. Gil, para você que além de sócio é meu melhor amigo e compadre, uma lembrança especial por tudo que já passamos nesta vida e pelo que ainda passaremos, sempre juntos, ainda mais agora que o seu filho (meu querido afilhadinho: Gilzinho) precisará de tantos bons exemplos em sua jornada. Luís Otávio, também não esqueci seus esforços para a publicação desta obra. Muito carinho também ao pessoal da Artimar, que tem nos ajudado a trilhar esta dura jornada.

Agora, o principal: você, leitor, que está apenas começando a conhecer o resultado de tanto trabalho.

SOBRE O AUTOR

David José de Souza é formado em Engenharia Mecânica, e trabalhou durante quatro anos no setor. Após isso, trabalhou durante três anos no segmento de informática pela Canal 1 Informática. Atualmente, é sócio e diretor administrativo do Grupo Mosaico, ao qual tem se dedicado nos últimos anos. Especializou-se nos microcontroladores da Microchip devido à sua grande utilização nos laboratórios da Mosaico Engenharia. É responsável também pelas especificações de projetos, com as quais adquiriu experiência necessária para poder ministrar palestras e cursos sobre o assunto. Pretende ainda se dedicar a outras obras sobre microcontroladores e desenvolvimento de projetos.

SOBRE O MATERIAL DISPONÍVEL NA INTERNET

O material disponível na Internet contém o código-fonte dos sete exemplos completos dados no capítulo 11, e uma versão de solução para o projeto proposto do "Dimmer".

Para que você possa utilizá-los, basta possuir o MPLab instalado em seu computador. Caso você ainda não tenha o MPLab, uma cópia pode ser conseguida gratuitamente no site da Microchip (www.microchip.com) ou da Mosaico (www.mosaico-eng.com.br). O PIC16F628A, assim como os sistemas disponíveis para gravação e execução, podem ser adquirido no site da Mosaico ou através dos distribuidores da Microchip no Brasil (veja a página de Distribuidores).

desbravando.exe 19 KB

Procedimento para Download:

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

- **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.
- **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.

Exemplo: "C:\Desbravando o 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

Não pairam dúvidas sobre a época em que vivemos: a "ERA DO CONHECIMENTO". As empresas tomam ciência que seu maior patrimônio não está nos equipamentos ou instalações, mas na capacidade de raciocínio e conhecimento de seus profissionais.

Há sete anos a ARTIMAR, representante exclusivo da MICROCHIP no Brasil, começou a desbravar o mercado, buscando introduzir uma nova tecnologia em microcontroladores: a família PIC. Hoje, anos mais tarde, fomos presenteados com esta obra, a qual aconselhamos a qualquer pessoa que já programou ou pretende programar um microcontrolador, pois seu conteúdo envolve conceitos teóricos e exemplos práticos, além de apêndices altamente informativos.

Atualmente, é imprescindível que obras como esta sejam acessíveis aos profissionais da área, a qual foi idealizada e apoiada por uma empresa altamente experiente no assunto, a MOSAICO ENGENHARIA, que poderia assumir a cômoda posição de se calar e manter somente para si o conhecimento e a experiência repassados aos leitores. Felizmente, o ideal dessa empresa vai contra este conceito, pois nesta era do conhecimento, a distribuição de riqueza inicia-se com a disseminação de informação.

Escrita em linguagem simples e objetiva, sem com isso deixar de lado os aspectos técnicos, é uma consulta obrigatória para engenheiros, técnicos ou hobbistas que pretendam estabelecer o primeiro contato com os microcontroladores da linha PIC. Aqueles que já possuem conhecimento de outro tipo de microcontrolador, terão acesso ao material necessário para uma tranquila transição para essa família. Para os que já programam, o livro pode servir como uma excelente fonte de referência e consulta.

Que este seja apenas o início de um projeto maior, que visa a criação de uma série de obras que difunda, com seriedade e responsabilidade, o bem maior de nossa era: o CONHECIMENTO.

André Rabner - Gerente de Vendas
ARTIMAR LTDA.



SUMÁRIO

CAPÍTULO 1 - INTRODUÇÃO	19
O OBJETIVO DESTE LIVRO.....	19
A MICROCHIP	19
ESTRUTURA DO LIVRO E PÚBLICO ALVO	20
CAPÍTULO 2 - MICROCONTROLADORES.....	21
A MICROCHIP NO BRASIL	21
O QUE SÃO E PARA QUE SERVEM.....	21
A ARQUITETURA HAVARD E A FILOSOFIA RISC.....	22
A ESTRUTURAÇÃO INTERNA	23
OS CICLOS DE MÁQUINA	24
CAPÍTULO 3 - INTRODUÇÃO ÀS MEMÓRIAS.....	27
A MEMÓRIA DE PROGRAMA.....	27
VETOR DE RESET	27
VETOR DE INTERRUPÇÃO.....	28
PILHA (STACK)	28
A MEMÓRIA DE DADOS.....	28
REGISTRADORES ESPECIAIS	28
REGISTRADORES DE USO GERAL.....	29
EEPROM	29
CAPÍTULO 4 - INTRODUÇÃO ÀS INTERRUPÇÕES	31
O QUE SÃO E COMO FUNCIONAM	31
AS INTERRUPÇÕES EXISTENTES NO PIC.....	31
INTERRUPÇÕES DE TIMERS (3).....	32
INTERRUPÇÃO EXTERNA	32
INTERRUPÇÃO POR MUDANÇA DE ESTADO	32
INTERRUPÇÃO DE FIM NA ESCRITA NA EEPROM	32
INTERRUPÇÃO DE COMPARADOR	33
INTERRUPÇÕES DE USART (2)	33
INTERRUPÇÃO DE CCP	33
COMO TRATÁ-LAS ?	33

CAPÍTULO 5 - O PIC 16F628A	35
INTRODUÇÃO	35
A PINAGEM	36
AS NOMENCLATURAS UTILIZADAS	36
CARACTERÍSTICAS ELÉTRICAS E OUTRAS	39
MAPAS DAS MEMÓRIAS	39
MEMÓRIA DE PROGRAMA	39
MEMÓRIA DE DADOS	40
 CAPÍTULO 6 - OS REGISTRADORES ESPECIAIS QUE CONTROLAM TUDO	41
INTRODUÇÃO	41
GERAIS	41
STATUS E PCON	41
OPTION	42
INTCON, PIR1 E PIE1	42
CONHECENDO O PCL E PCLATH	42
PORTAS	42
TRIS	42
PORTS	43
CONTADORES	43
TIMER 0	43
TIMER 1	43
TIMER 2	43
EEPROM	44
EEADR E EEDATA	44
EECON1 E EECON2	44
MÓDULO CCP	44
CCP1CON, CCPR1H E CCPR1L	44
MÓDULO COMPARADOR	44
CMCON	44
MÓDULO VOLTAGEM DE REFERÊNCIA	45
VRCON	45
MÓDULO USART	45
TXSTA E RCSTA	45
SPBRG	45
TXREG E RCREG	45
ENDEREÇAMENTO INDIRETO	45
FSR E O INDF	45

CAPÍTULO 7 - CONHECENDO UM POUCO O SET DE INSTRUÇÕES	47
OS TERMOS UTILIZADOS	47
A CONSTRUÇÃO DOS NOMES DAS INSTRUÇÕES.....	48
OS GRUPOS DE INSTRUÇÕES	48
O RESUMO DAS INSTRUÇÕES	49
CAPÍTULO 8 - MPLAB	51
INTRODUÇÃO À FERRAMENTA.....	51
O AMBIENTE DE TRABALHO.....	51
ABRINDO UMA ÁREA DE TRABALHO E UM PROJETO	52
ASSOCIANDO E ABRINDO UM ARQUIVO DE CÓDIGO-FONTE.....	54
CONFIGURANDO O WORKSPACE E O PROJETO	55
COMPILEANDO O PROJETO.....	57
ERROS, WARNINGS E MENSAGENS.....	57
CAPÍTULO 9 - CONSIDERAÇÕES INICIAIS SOBRE O HARDWARE.....	59
INTRODUÇÃO.....	59
CONFIGURANDO AS OPÇÕES DO PIC	59
TIPO DE OSCILADOR.....	60
WATCHDOG TIMER	60
POWER UP TIMER	60
BROWN OUT DETECT	60
MASTER CLEAR ENABLE	61
LOW VOLTAGE PROGRAM	61
DATA EE READ PROTECT	61
CÓDIGO DE PROTEÇÃO.....	61
DEFININDO AS CONFIGURAÇÕES NO PRÓPRIO PROGRAMA.....	62
GRAVAÇÃO DE IDS	63
CHECKSUM	63
OPÇÕES DE GRAVAÇÃO.....	63
EFETUANDO A GRAVAÇÃO	64
GRAVAÇÃO IN-CIRCUIT	65
GRAVADORES E OUTROS	66

CAPÍTULO 10 - GRAVANDO O PIC	67
ALIMENTAÇÃO	67
OSCILADORES	67
RC	68
RESSOADOR.....	68
CRISTAL.....	69
HÍBRIDO OU CIRCUITOS DE OSCILAÇÃO.....	69
POWER-ON RESET (POR) BÁSICO.....	70
 CAPÍTULO 11 - PROGRAMAÇÃO	 71
CRIANDO UM PROGRAMA.....	71
ESTRUTURANDO O CÓDIGO-FONTE.....	71
A IMPORTÂNCIA DOS COMENTÁRIOS	72
ARQUIVOS DE DEFINIÇÃO: INCLUDES	72
CONSTANTES E DEFINIÇÕES: EQU E DEFINES	78
EXEMPLO 0 - ESTRUTURAÇÃO	79
TRABALHANDO COM A MEMÓRIA.....	82
O REGISTRADOR WORK (W OU ACUMULADOR).....	82
CONHECENDO OS BANCOS DE MEMÓRIA.....	82
LIDANDO COM DADOS (MOVLW, MOVWF, MOVF, CLRF E CLRW).....	85
INICIALIZANDO O SISTEMA.....	87
DEFININDO LOCAIS PARA AS VARIÁVEIS.....	87
RESERVANDO ESPAÇO PARA FLAGS.....	87
CRIANDO CONSTANTES.....	88
DEFININDO AS ENTRADAS E SAÍDAS	89
O VETOR DE RESET	90
INICIALIZANDO AS VARIÁVEIS	91
TRABALHANDO COM ROTINAS	91
ROTINAS DE DESVIO	92
ROTINAS DE CHAMADA.....	93
TOMANDO DECISÕES E FAZENDO DESVIOS.....	94
TESTANDO BITS E FLAGS (BTFSC E BTFSS)	94
MUDANDO BITS E FLAGS (BSF E BCF).....	94
TRABALHANDO COM AS PORTAS	95
LENDÔ UMA PORTA.....	95
ESCREVENDO EM UMA PORTA	96
EXEMPLO 1 - BOTÃO E LED	97

FAZENDO OPERAÇÕES ARITMÉTICAS BÁSICAS	101
SOMANDO (INCF, INCFSZ, ADDWF E ADDLW)	101
SUBTRAINDO (DECf, DECFSZ, SUBWF E SUBLW)	103
AS COMPARAÇÕES MAIOR QUE, MENOR QUE E IGUAL	105
MULTIPLICANDO (RLF).....	105
DIVIDINDO (RRF)	107
EXEMPLO 2 - CONTADOR SIMPLIFICADO	108
TRABALHANDO DIRETAMENTE COM BYTES	113
AND (ANDWF E ANDLW).....	113
OR (IORWF E IORLW).....	114
XOR (XORWF E XORLW).....	115
COMPLEMENTO (COMF).....	117
INVERSÃO (SWAPF)	117
CONTANDO TEMPO E CRIANDO DELAYS	118
UTILIZANDO REGISTROS TEMPORÁRIOS PARA CRIAR DELAYS.....	118
EXEMPLO 3 - PISCA-PISCA	119
OPERANDO DIRETAMENTE COM O PROGRAM COUNTER	125
USANDO O PCL PARA ESCOLHER ENTRE VÁRIAS ROTINAS	126
USANDO O PCL PARA MONTAR UMA TABELA DE VALORES	126
EXEMPLO 4 - CONTADOR MELHORADO	127
EXPLORANDO AS INTERRUPÇÕES	132
LIGANDO AS CHAVES CORRETAS.....	133
A ESTRUTURA BÁSICA DA ROTINA DE INTERRUPÇÃO (RETFIE)	133
CHECANDO QUAL FOI A INTERRUPÇÃO OCORRIDA	135
CONHECENDO MELHOR O TIMER 0 E O PRESCALER.....	136
CONHECENDO O PRESCALER	136
UTILIZANDO O TIMER PARA MARCAR TEMPO	137
TRATANDO A INTERRUPÇÃO DE TIMER.....	138
TRATANDO A INTERRUPÇÃO EXTERNA	139
TRATANDO A INTERRUPÇÃO DE MUDANÇA DE ESTADO	140
OUTRAS INTERRUPÇÕES	140
EXEMPLO 5 - TIMER SIMPLIFICADO	140
UTILIZANDO A EEPROM	148
ESCREVENDO NA EEPROM	149
TRATANDO A INTERRUPÇÃO DE FINAL DE ESCRITA DA EEPROM	150
LENDÔ DA EEPROM	151
EXEMPLO 6 - CONTADOR FINAL.....	151
ACESSO INDIRETO À MEMÓRIA	159

TRABALHANDO COM FSR E INDF	159
TRABALHANDO COM WATCHDOG (WDT) (CRLWDT).....	160
UTILIZANDO O WDT PARA EVITAR TRAVAMENTOS	160
UTILIZANDO O WDT PARA MELHORAR A INICIALIZAÇÃO	161
UTILIZANDO O WDT PARA AJUDAR NA SOLUÇÃO DE PROBLEMAS	162
MODO SLEEP.....	163
ENTRANDO NO MODO SLEEP (SLEEP)	163
SAINDO DO MODO SLEEP	163
CAPÍTULO 12 - RECURSOS AVANÇADOS	165
TIMER 1	165
TIMER 2	166
CCP - CAPTURE, COMPARE E PWM.....	167
CAPTURE	167
COMPARE	168
PWM	168
COMPARADORES.....	169
TENSÃO DE REFERÊNCIA AJUSTÁVEL	170
USART	171
MODO ASSÍNCRONO	171
CAPÍTULO 13 - NOVAS CONSIDERAÇÕES SOBRE O HARDWARE	175
EVITANDO PROBLEMAS COM O PIC E SUAS PORTAS	175
POWER-ON RESET (POR) MELHORADO	176
BROWN-OUT	176
CAPÍTULO 14 - SIMULANDO E “DEBUGANDO” O SISTEMA	179
INTRODUÇÃO.....	179
UMA LISTAGEM COMPLETA (ABSOLUT LIST).....	179
ACERTANDO AS CONDIÇÕES DO HARDWARE	180
EXECUTANDO O PROGRAMA.....	181
RODANDO DIRETO (RUN)	182
RODANDO EM MODO DE ANIMAÇÃO (ANIMATE)	182
PARALISANDO A EXECUÇÃO (HALT).....	182
RODANDO PASSO A PASSO (STEPS)	182
RESETANDO O PROGRAMA	183

PARANDO NOS PONTOS CERTOS (BRAKEPOINTS E RUN TO CURSOR)	183
CONTROLANDO AS PASSAGENS (TRACE).....	183
OUTROS RECURSOS	184
VISUALIZANDO A MEMÓRIA.....	184
MAPA DA MEMÓRIA	185
REGISTROS ESPECIAIS (SFRS)	185
PILHA (STACK)	186
EEPROM	186
MEMÓRIA DE PROGRAMA.....	186
REGISTROS DIVERSOS (WATCHS)	186
CONTROLANDO AS ENTRADAS	187
ESTÍMULOS DIRETOS (ASSÍNCRONOS).....	187
ESTÍMULOS PERIÓDICOS (SÍNCRONOS).....	189
ARQUIVO DE ESTÍMULOS.....	189
CONTANDO O TEMPO CORRETAMENTE.....	190
 CAPÍTULO 15 - PROJETOS PROPOSTOS	 191
TIMER	191
DIMMER.....	192
 APÊNDICE A - REGISTRADORES ESPECIAIS (SFR).....	 193
APÊNDICE B - SET DE INSTRUÇÕES COMPLETO	209
APÊNDICE C - DIRETRIZES DA LINGUAGEM MPASM	229
APÊNDICE D - INSTRUÇÕES ESPECIAIS	255
APÊNDICE E - OPERADORES DO MPASM	257
APÊNDICE F - TABELAS	259
APÊNDICE G - HARDWARE PROPOSTO	261

CAPÍTULO

1

INTRODUÇÃO

O OBJETIVO DESTE LIVRO

A principal idéia que nos levou à elaboração desta obra foi a real necessidade de uma literatura atualizada de suporte aos interessados no aprendizado da programação dos microcontroladores da Microchip.

Atualmente, a eletrônica tem evoluído rapidamente em muitas áreas, e a eletrônica digital ficou cada vez mais acessível a todos os técnicos e engenheiros, e até mesmo a curiosos. Os microcontroladores surgiram exatamente para aumentar ainda mais essa evolução e facilitar o desenvolvimento de novos produtos e tecnologias. Por isso, a cada dia, mais e mais pessoas precisam atualizar-se e aprender outras formas de projetar novos sistemas.

Infelizmente, no Brasil, ainda não temos as mesmas facilidades de obter informações como em outras partes do mundo, mas com a globalização, Internet, abertura de fronteiras e outras ferramentas que aproximam os povos e popularizam a tecnologia, hoje já é possível ter acesso direto a componentes de última geração, como, por exemplo, os microcontroladores. Com este livro estamos facilitando também o acesso ao aprendizado desta tecnologia, tentando expressar em palavras e exemplos a experiência adquirida em anos de prática com o desenvolvimento de projetos, testes em laboratórios e muitos acertos e erros pelos quais não desejamos que você passe. O importante é que este material foi todo preparado e testado com o que há de mais moderno (até a data de fechamento da edição) em relação ao ambiente de desenvolvimento e modelo de microcontrolador.

Nesta edição foram feitas uma revisão e a atualização, visando três metas principais:

1. Revisar a estruturação do livro, alterando a ordem e a organização de alguns assuntos para facilitar ainda mais a didática e a transferência de informações.

2. Atualizar tecnicamente o material em relação à versão do software MPLab apresentado. Agora tratamos da versão 6.22, atual no momento, com a alteração dos conceitos e das telas relacionadas a esta versão.
3. Atualização técnica em relação ao microcontralador utilizado como base de estudo, no caso o modelo PIC16F628A e seus recursos adicionais.

A MICROCHIP

O nome PIC, assim como outros termos existentes nesta publicação, tais como: MPASM e MPLab, pertencem à Microchip. Destacamos ainda que muitas figuras, termos e até formatações foram retirados dos manuais originais da Microchip com o objetivo de facilitar ao leitor a comparação entre esses materiais e a obra em questão.

ESTRUTURA DO LIVRO E PÚBLICO-ALVO

A estruturação deste livro foi cuidadosamente estudada de forma que cada assunto introduzisse ou colaborasse com o próximo. Desta forma, recomendamos aos iniciantes em microcontroladores que a ordem dos capítulos seja naturalmente seguida. É claro que para os mais familiarizados com o assunto, os primeiros capítulos podem trazer poucas novidades e poderiam ser deixados de lado, mas acreditamos que sua leitura não irá representar nenhuma perda de tempo; muito pelo contrário, poderá reforçar certas bases fundamentais.

Tentamos criar aqui uma ferramenta que sirva para todos os tipos de usuários, desde aqueles que possuem somente os princípios básicos da eletrônica, até aqueles que já a dominam muito bem. Desta forma, o livro pode ser o responsável por um grande aprendizado da eletrônica digital ou servir como uma fonte de referência para consultas rápidas sobre microcontroladores da Microchip, possuindo ótimos apêndices para isso.

CAPÍTULO

2

MICROCONTROLADORES

A MICROCHIP NO BRASIL

A Microchip iniciou seus negócios no Brasil em 1990, por meio da seleção de um parceiro à altura para o mercado brasileiro, permanecendo o mesmo até hoje. A Aplicações Eletrônicas Artimar Ltda., ou só Artimar, como é conhecida no mercado, foi fundada em 1962, pelo sr. Artur Rabner. Desde o princípio, a Artimar vem trabalhando junto ao mercado como representante exclusiva e distribuidora de várias empresas americanas no setor de eletroeletrônica. Portanto, são mais de 35 anos de tradição, parcerias e muitas novas amizades. Por causa deste perfil, a Microchip viu na Artimar uma empresa capaz, técnica e comercialmente para introduzir seus produtos em nosso mercado. O resultado foi um grande sucesso. Nove anos mais tarde, a Microchip, junto com a Artimar, constitui um sólido canal de distribuição, visando atender da melhor maneira possível o cliente, com o intuito de facilitar seu acesso a esta nova tecnologia.

Hoje, a Artimar passou a ser somente a representante exclusiva da Microchip no Brasil, e a distribuição e comercialização foi passada a outras empresas com o intuito de expandir ainda mais a divulgação e o acesso aos microcontroladores da família PIC. Para mais informações sobre as empresas ligadas à Microchip no Brasil, consulte a página de distribuidores.

O QUE SÃO E PARA QUE SERVEM

Em poucas palavras, poderíamos definir o microcontrolador como um "pequeno" componente eletrônico, dotado de uma "inteligência" programável, utilizado no controle de processos lógicos. Para entendermos melhor esta definição, vamos analisá-la por partes:

O controle de processos deve ser entendido como o controle de periféricos, tais como: LEDs, botões, displays de segmentos, displays de cristal líquido (LCD), resistências, relês, sensores diversos (pressão, temperatura, etc.) e muitos outros. São chamados de controles lógicos, pois a operação do sistema baseia-se nas ações

lógicas que devem ser executadas, dependendo do estado dos periféricos de entrada e/ou saída.

O microcontrolador é programável, pois toda a lógica de operação de que acabamos de falar é estruturada na forma de um programa e gravada dentro do componente. Depois disso, toda vez que o microcontrolador for alimentado, o programa interno será executado. Quanto à "inteligência" do componente, podemos associá-la à Unidade Lógica Aritmética (ULA), pois é nessa unidade que todas as operações matemáticas e lógicas são executadas. Quanto mais poderosa a ULA do componente, maior sua capacidade de processar informações.

Na nossa definição, o microcontrolador ganhou ainda o adjetivo "pequeno", pois em uma única pastilha de silício encapsulada (popularmente chamada de CI ou CHIP), temos todos os componentes necessários ao controle de um processo, ou seja, o microcontrolador está provido internamente de memória de programa, memória de dados, portas de entrada e/ou saída paralela, timers, contadores, comunicação serial, PWMs, conversores analógico-digitais, etc. Esta é uma das características fundamentais que diferencia os microcontroladores dos microprocessadores, pois os últimos, apesar de possuírem uma ULA muito mais poderosa, não possuem todos esses recursos em uma única pastilha.

Atualmente, muitos equipamentos de nosso uso diário, tais como: eletrodomésticos, videocassetes, alarmes, celulares e brinquedos, entre outros, utilizam microcontroladores para execução de suas funções básicas. Portanto, pode ser que você nem sabia, mas esses componentes já fazem parte da sua vida há um bom tempo.

A ARQUITETURA HAVARD E A FILOSOFIA RISC

Os microcontroladores PIC apresentam uma estrutura de máquina interna do tipo Havard, enquanto grande parte dos microcontroladores tradicionais apresenta uma arquitetura tipo Von-Neumann. A diferença está na forma como os dados e o programa são processados pelo microcontrolador. Na arquitetura tradicional, tipo Von-Neumann, existe apenas um barramento (bus) interno (geralmente de 8 bits), por onde passam as instruções e os dados. Já na arquitetura tipo Havard existem dois barramentos internos, sendo um de dados e outro de instruções. No caso dos microcontroladores PIC, o barramento de dados é sempre de 8 bits e o de instruções pode ser de 12, 14 ou 16 bits, dependendo do microcontrolador. Esse tipo de arquitetura permite que, enquanto uma instrução é executada, outra seja "buscada" da memória, o que torna o processamento mais rápido. Além disso, como o barramento de instruções é maior do que 8 bits, o OPCODE da instrução já inclui o dado e o local onde ela vai operar (quando necessário), o que significa que apenas uma posição de memória é utilizada por instrução, economizando assim muita memória de programa.

Desta forma, podemos observar que dentro da palavra do OPCODE, que pode ser de 12, 14 ou 16 bits, não sobra muito espaço para o código da instrução propriamente dito. Por isso, os PICs utilizam uma tecnologia chamada RISC, que significa Reduced

Instruction Set Computer (Computador com set de instruções reduzido). Desta forma, os PICs possuem cerca de 35 instruções (o número correto varia de acordo com o microcontrolador), muito menos que os microcontroladores convencionais (CISC) que chegam a possuir mais de cem instruções. Isso torna o aprendizado muito mais fácil e dinâmico, mas, por outro lado, implica no fato de que muitas funções devem ser "construídas", pois não possuem uma instrução direta, exigindo maior habilidade do programador.

A ESTRUTURAÇÃO INTERNA

Caso você ainda seja um leigo no que diz respeito a microcontroladores ou eletrônica digital, talvez o diagrama seguinte seja muito complicado à primeira vista. Mas não se preocupe, ele foi colocado aqui para que os leitores mais experientes possam conhecer um pouco mais o componente por dentro. No decorrer do livro, explicaremos cada uma das partes apresentadas neste diagrama, e voltar a consultá-lo mais tarde pode ser muito interessante. O diagrama mostrado é do PIC16F628A (figura 2.1).

No diagrama de blocos (retirado do datasheet original da Microchip) podem ser visualizadas as diversas partes que compõem o microcontrolador PIC16F628A (modelo que será estudado no decorrer do livro). Observe a ULA (em inglês: ALU), diretamente ligada ao registrador W (work reg - explicado posteriormente). No canto superior esquerdo temos a memória de programa, e saindo deste bloco temos um barramento de 14 bits (Program Bus 14). Mais ao centro está a memória de dados (RAM). Ela já possui um barramento de 8 bits (Data Bus 8), conforme explicado na definição da arquitetura Havard. Do lado direito podemos visualizar as portas com todos os seus pinos de I/O. Na parte inferior, os periféricos, tais como a EEPROM (memória de dados não volátil), os timers (Timer0, Timer1 e Timer2), o comparador interno, o módulo CCP (Capture, Compare e PWM) e a porta serial (USART). Um pouco mais ao centro, temos o registrador de status (STATUS reg). Algumas informações importantes sobre as operações aritméticas da ULA ficam armazenadas nesse registrador. Na parte superior temos ainda o contador de linha de programa (Program Counter) e a pilha de 8 níveis (Stack). Entre todos os periféricos, a comunicação é feita por meio de um barramento de oito vias. Temos ainda os circuitos internos de reset, osciladores, Watchdog Timer (WDT), Power-up e Brown-out internos.

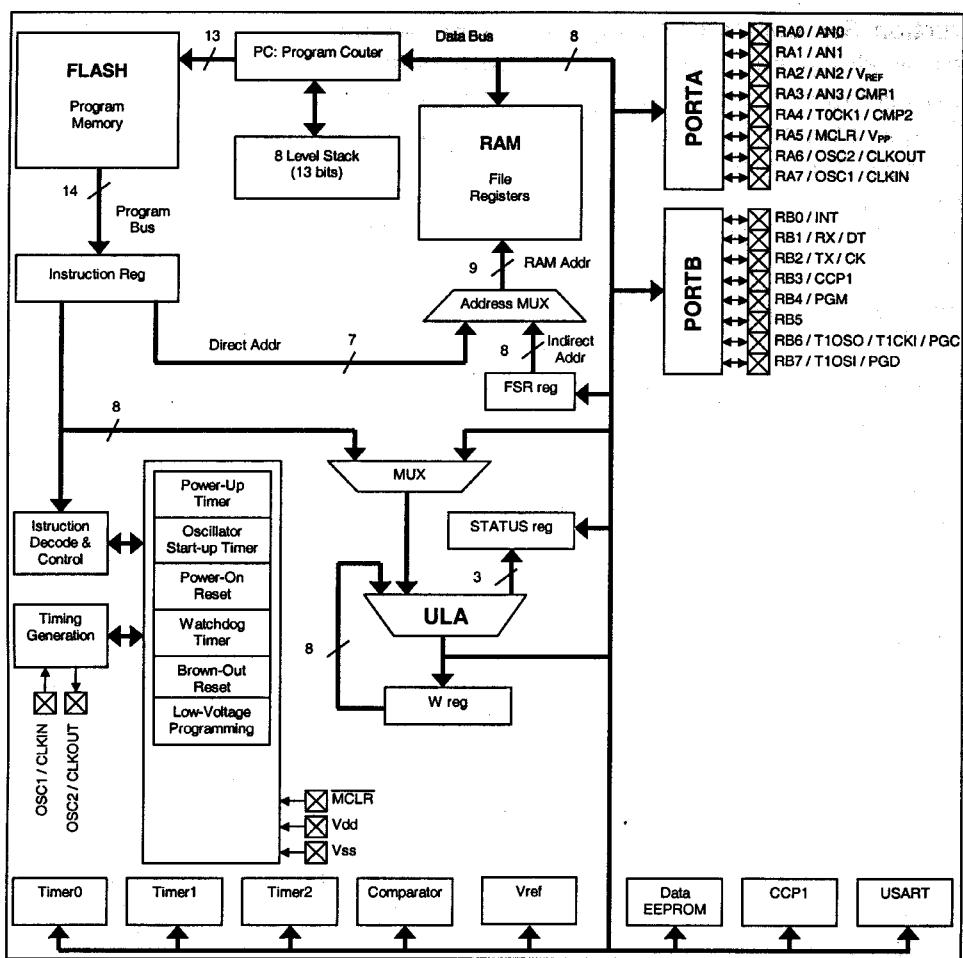


Figura 2.1 - Diagrama Interno do PIC16F628A.

OS CICLOS DE MÁQUINA

Nos microcontroladores PIC, o sinal do clock é internamente dividido por quatro. Portanto, para um clock externo de 4 MHz, temos um clock interno de 1 MHz e, consequentemente, cada ciclo de máquina dura 1μs.

A divisão do clock por quatro forma as fases Q1, Q2, Q3 e Q4. O program counter é incrementado automaticamente na fase Q1 do ciclo de máquina e a instrução seguinte é buscada da memória de programa e armazenada no registrador de instruções no ciclo Q4. Ela é decodificada e executada no próximo ciclo, no intervalo de Q1 até Q4. Essa característica de buscar a informação num ciclo de máquina e executá-la no próximo é conhecida como PIPELINE. Ela permite que quase todas as instruções sejam executadas em apenas um ciclo, gastando assim 1μs (para um clock de 4 MHz) e tornando o sistema muito mais rápido. As únicas

exceções referem-se às instruções que geram "saltos" no program counter, como chamadas de rotinas e retornos. Ao executar essas instruções, o PIPELINE deve ser primeiramente limpo para depois poder ser carregado novamente com o endereço correto, consumindo para isso dois ciclos de máquina. Esse PIPELINE é facilmente implementado devido à arquitetura Harvard.

O diagrama seguinte foi retirado do manual original da Microchip e demonstram claramente as divisões do ciclo nas quatro fases (Q1 a Q4) e o conceito de PIPELINE.

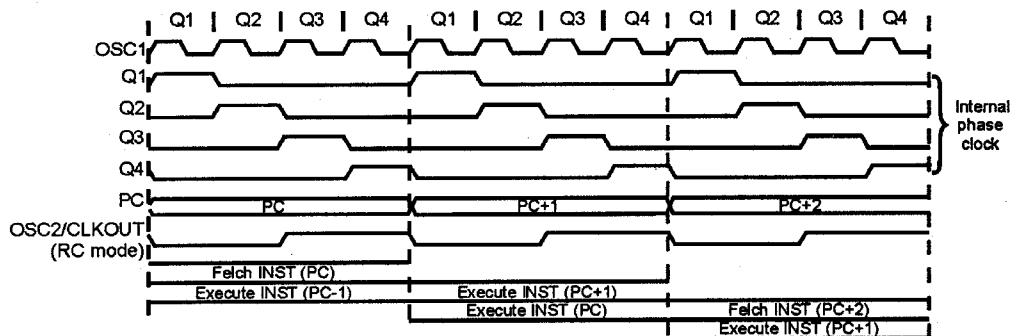


Figura 2.2 - Esquema dos Ciclos de Máquina.

CAPÍTULO

3

INTRODUÇÃO ÀS MEMÓRIAS

Este capítulo deve explicar de uma forma geral como funcionam e como são organizadas as memórias do PIC. Muitos detalhes específicos, principalmente as referências a comandos, serão tratados nos capítulos posteriores.

Como já foi dito durante a explicação da arquitetura Havard, o PIC possui barramentos diferenciados para as memórias de programa e de dados. Desta forma, podemos concluir também que essas memórias são totalmente separadas. No caso do PIC 16F628A, que será o modelo estudado por esta obra, existe ainda uma terceira memória: a memória não-volátil - EEPROM.

A MEMÓRIA DE PROGRAMA

Devido à estruturação Havard, a memória de programa do PIC pode ser de 12, 14 ou 16 bits. O tamanho dessa memória também varia muito de modelo para modelo. Na maioria dos modelos, essa memória é do tipo EPROM, que só pode ser gravada uma vez para PICs normais, ou gravada várias vezes no caso de PICs janelados (que podem ser apagados por meio de luz ultravioleta). Existem ainda modelos que possuem a memória de programa do tipo FLASH, que pode ser gravada várias vezes sem a necessidade de apagar a gravação anterior. Esses PICs são muito mais fáceis de trabalhar para o desenvolvimento do sistema, mas, por outro lado, são muito mais caros para uma fabricação em série.

VETOR DE RESET

Trata-se do primeiro endereço da memória de programa que será executado quando o PIC começar a rodar (após a alimentação ou um reset). Na maioria dos modelos, o reset aponta para o endereço 0x00, mas em alguns modelos mais antigos ele pode apontar para o último endereço disponível.

VETOR DE INTERRUPÇÃO

As rotinas de interrupções serão armazenadas na área de programação, juntamente com todo o resto do programa. No entanto, existe um endereço que é reservado para o início do tratamento de todas as interrupções, nos modelos de PIC que possuem esse recurso. Esse endereço é denominado vetor de interrupção e encontra-se sempre na posição 0x04.

PILHA (STACK)

A pilha é um local, totalmente separado da memória de programação, em que serão armazenados os endereços de retorno quando utilizarmos instruções de chamadas de rotinas. Quando o programa é desviado para o começo de uma rotina por meio da instrução correta, o endereço seguinte ao ponto que estava sendo rodado é armazenado na pilha para que, ao fim da rotina, o programa possa retornar. O tamanho da pilha também varia de acordo com o modelo de PIC, e esse tamanho determina a quantidade de rotinas que podem ser chamadas ao mesmo tempo. Caso se tente chamar um número de rotinas maior que o tamanho da pilha, o endereço de retorno mais antigo será perdido.

A MEMÓRIA DE DADOS

A memória de dados do sistema é a RAM, que é utilizada para guardar todas as variáveis e registradores utilizados pelo programa. Essa memória armazena dados de 8 bits e é volátil, ou seja, quando o PIC é desligado, ela é automaticamente perdida. Podemos dividi-la em dois grupos que serão estudados em seguida: Registradores especiais e Registradores de uso geral.

A memória de dados muitas vezes também é dividida em mais de um banco, para possibilitar o acesso aos endereços, com o auxílio de chaves que controlam o banco que está sendo utilizado no momento.

REGISTRADORES ESPECIAIS

Nessa região da memória encontram-se todos os registradores especiais, denominados SFRs, utilizados pelo microcontrolador para a execução do programa e processamentos da ULA. Esses registradores serão minuciosamente estudados mais adiante. O que realmente importa neste momento é entendermos que esses registradores ocupam espaço na RAM e podem ser acessados da mesma maneira que as variáveis do sistema, com mudança somente do endereço de acesso. Esses registradores podem ser escritos/lidos tanto pelo usuário quanto pelo hardware. A quantidade de SFRs depende do modelo de PIC, mas eles sempre são armazenados na parte baixa da memória (início dos endereços) e às vezes podem estar espalhadas em mais de um banco de memória.

REGISTRADORES DE USO GERAL

Trata-se de uma área destinada ao armazenamento de variáveis definidas pelo usuário para serem escritas e lidas pelo programa. O tamanho dessa memória varia de acordo com o modelo de PIC e também pode ocupar mais de um banco.

EEPROM

Alguns modelos de PIC possuem ainda uma terceira memória que também pode ser utilizada pelo usuário para guardar dados. Entretanto, ao contrário da memória de dados vista anteriormente, esta é uma EEPROM, isto é, uma memória não volátil, que consegue manter as informações mesmo sem alimentação. Os modelos que não possuem esta memória internamente podem utilizar esse recurso por intermédio de uma memória EEPROM externa, interligada ao microcontrolador por I/Os e com rotinas implementadas para possibilitar a escrita e leitura de dados.

INTRODUÇÃO ÀS INTERRUPÇÕES

O QUE SÃO E COMO FUNCIONAM?

Você já deve ter ouvido falar muitas vezes em interrupções, seja na área dos microcontroladores ou mesmo sobre os microcomputadores. Mas será que você sabe o que realmente elas são? Daremos agora uma explicação global sobre as interrupções, e para que servem, sem entrarmos em muitos detalhes técnicos, pois existe um capítulo específico para isso.

Como o próprio nome diz, uma interrupção serve para interromper o programa imediatamente. Desta maneira, podemos tomar atitudes instantâneas. As interrupções são ações tratadas diretamente pelo hardware, o que as torna muito rápidas e disponíveis em qualquer ponto do sistema. Assim sendo, quando uma interrupção acontece, o programa é paralisado, uma função específica (definida pelo programador) é executada e depois o programa continua a ser executado no mesmo ponto em que estava. Fantástico, não é? Mas para que serve isso, afinal? Para a solução de muitos problemas complexos. Conhecendo agora os tipos de interrupções disponíveis no PIC, podemos ver alguns exemplos de suas aplicações.

AS INTERRUPÇÕES EXISTENTES NO PIC

Existem modelos que nem possuem interrupções, como é o caso, por exemplo, dos PICs 12C50X, 16C54, 16C55 e outros de gerações mais antigas, mas em todos os modelos mais novos as interrupções estão presentes. Existe também uma grande quantidade de interrupções com finalidades diferentes, dependendo do PIC analisado. Veremos aqui somente as interrupções existentes no PIC16F628A, apesar do fato que nem todas serão estudadas neste livro. Desta forma, temos quatro grupos de interrupções:

INTERRUPÇÕES DE TIMERS (3)

Essas interrupções acontecem sempre que um dos contadores de tempo interno, denominados TMRO (Timer 0) e TMR1 (Timer 1), estouram. No caso do TMRO, como ele é um contador de 8 bits, sempre que ele passar de 0xFF para 0x00. Já no caso do TMR1 (16 bits), sempre que ele passar de 0xFFFF para 0x0000. Existe também a interrupção para o TMR2 (Timer 2). Neste caso, entretanto, além de este contador ser de 8 bits, a interrupção não acontece necessariamente quando ele estoura o limite de 0xFF, e sim quando ele atinge um valor qualquer especificado em outro registrador especial (PR2). Este tipo de interrupção é utilizado normalmente para a contagem de tempo. Como pode acontecer a qualquer momento, a contagem de tempo fica precisa, não dependendo de análises constantes durante o programa para garantir que o tempo seja contado. Como veremos adiante, TMRO e TMR1 podem tanto ser incrementados internamente pelo clock da máquina como também por um sinal externo. Neste caso, eles passam a ser contadores de pulsos, podendo ser utilizados para outras finalidades.

INTERRUPÇÃO EXTERNA

Essa interrupção é gerada por um sinal externo ligado a uma porta específica do PIC, que no caso é a porta RB0 (as portas e suas nomenclaturas serão melhor conhecidas no próximo capítulo), caso ela esteja configurada como entrada. Desta maneira, podemos identificar e processar imediatamente um sinal externo. Ela é utilizada para diversas finalidades, como, por exemplo, a comunicação entre micros, garantindo o sincronismo, o reconhecimento de um botão ou outro sinal do sistema que necessite de uma ação imediata.

INTERRUPÇÃO POR MUDANÇA DE ESTADO

A interrupção externa, vista acima, funciona somente na borda de subida ou na borda de descida (quando o sinal lógico sobe ou desce), dependendo de como ela foi configurada. Já a interrupção por mudança de estado acontece em ambos os casos. Essa interrupção, por sua vez, está ligada às portas RB4, RB5, RB6 e RB7 simultaneamente. Por isso, se essas portas forem configuradas como entradas, a mudança de estado em qualquer uma delas irá gerar a interrupção. Esse tipo de interrupção pode ser utilizado, por exemplo, para criar um sincronismo com a rede de 60Hz, para o controle de um triac ou outro sistema semelhante.

INTERRUPÇÃO DE FIM NA ESCRITA NA EEPROM

Como já foi visto anteriormente, alguns PICs possuem uma memória EEPROM interna. Essa interrupção serve para detectarmos o final de uma rotina de escrita nessa memória. A utilização da interrupção não é obrigatória para que a escrita funcione, mas, como a EEPROM é lenta na hora de escrever, em alguns sistemas sua utilização pode ser necessária para evitar uma parada durante a escrita na EEPROM.

INTERRUPÇÃO DE COMPARADOR

Uma vez que o PIC16F628A possui dois comparadores internos, estes poderão ser utilizados de forma totalmente independente do programa (com acesso aos pinos, como se fossem comparadores externos), ou de forma mista. No último caso, o resultado da comparação poderá ser analisado pelo programa para a tomada de decisões. Para facilitar ainda mais esta função, a interrupção do comparador pode ser utilizada para avisar o sistema sempre que houver uma mudança de estado na resposta da comparação.

INTERRUPÇÕES DE USART (2)

O PIC16F628A possui um sistema completo (via hardware) para comunicação serial do tipo USART. Este sistema, além de facilitar todo o processamento para entrada e saída de dados seriais, possui duas interrupções para informar o programa quando um dado foi recebido e quando a transmissão de outro dado já foi terminada.

INTERRUPÇÃO DE CCP

O módulo CCP (Capture, Compare e PWM) também possui uma interrupção associada a ele para informar ao programa uma das duas situações possíveis: Fim da captura (Capture) ou Fim da comparação (Compare). O modo Capture pode ser utilizado para contar o tempo (TMR1) entre duas mudanças de estado de uma entrada específica (T1CKI). Com isso podemos implementar, por exemplo, um periodímetro. Já como modo Compare, podemos comparar o valor de TMR1 com o especificado em outro registrador especial (CCPR1). Desta forma podemos criar timers específicos ou monitorar a quantidade de pulsos na entrada relacionada ao TMR1 (T1CKI).

COMO TRATÁ-LAS?

Sempre que uma interrupção acontece (qualquer uma delas), o programa guarda o endereço da próxima linha a ser executada na pilha e desvia a execução do programa para um endereço fixo da memória de programação. Basta então, nesse endereço (0x04), escrever a rotina que irá reconhecer e tratar a interrupção acontecida. Quando a rotina de interrupção for terminada, o programa automaticamente voltará para o ponto em que estava antes da interrupção acontecer. O reconhecimento e o tratamento da interrupção serão analisados em um capítulo posterior.

CAPÍTULO

5

O PIC16F628A

INTRODUÇÃO

Infelizmente, não seria possível criarmos uma obra que englobasse todos os modelos da família PIC. Por isso tivemos de escolher um modelo que servisse de base para o estudo da grande maioria das funções existentes. Precisávamos então de um microcontrolador versátil, compacto e poderoso. O PIC16F628A foi o grande escolhido devido às suas características:

- Microcontrolador de 18 pinos, o que facilita a montagem de hardwares experimentais;
- Até 16 portas configuráveis como entrada ou saída e 2 osciladores internos (4 MHz e 37 kHz);
- 10 interrupções disponíveis (Timers, Externa, Mudança de Estado, EEPROM, USART, CCP e Comparador);
- Memória de programação FLASH com 2.048 words, que permite a gravação do programa diversas vezes no mesmo chip, sem a necessidade de apagá-lo por meio de luz ultravioleta, como acontece nos microcontroladores de janela;
- Memória EEPROM (não-volátil) interna com 128 bytes;
- Recursos adicionais avançados: módulo CCP, Comparador interno e USART;
- Programação com 14 bits e 35 instruções.

A grande vantagem da família PIC é que todos os modelos possuem um set de instruções bem parecido, assim como mantêm muitas semelhanças entre suas características básicas. Desta forma, ao conhecermos e estudarmos o PIC16F628A, estaremos nos familiarizando com todos os microcontroladores da Microchip (principalmente os de 12 e 14 bits), o que tornará a migração para outros modelos muito mais simples.

A PINAGEM

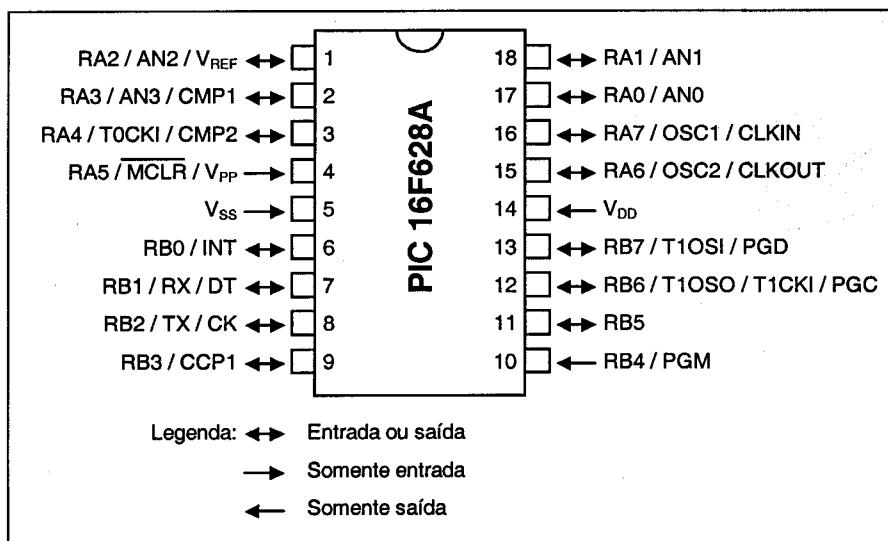


Figura 5.1 - O PIC16F628A.

AS NOMENCLATURAS UTILIZADAS

O PIC16F628A possui um total de 16 I/Os separados em dois grupos denominados PORTAS. Desta forma, temos a Porta A e a Porta B. Para facilitarmos o entendimento e a comparação com os datasheets originais, usaremos os termos provenientes do inglês: PORTA (port A) e PORTB (port B).

O PORTA possui oito pinos que podem ser configurados como entrada ou saída, e seus nomes são definidos como RA0, RA1, RA2, RA3, RA4, RA5, RA6 e RA7. Para termos a disponibilidade do pino RA5, perderemos o MCLR externo. Da mesma forma, para disponibilizarmos RA6 e RA7 não poderemos utilizar esse pinos para ligação de um oscilador externo. Por esse motivo, poderemos utilizar um dos dois osciladores internos existentes. O pino RA4 também pode ser utilizado para incremento externo do TMRO. Alguns outros pinos do PORTA ainda possuem funções sobre carregadas em relação aos dois compradores existentes. O pino RA2 pode ainda ser utilizado como uma saída de tensão programável (V_{REF}) com 16 níveis diferentes.

O PORTB também possui oito pinos configuráveis como entrada ou saída, sendo seus nomes RB0, RB1, RB2, RB3, RB4, RB5, RB6 e RB7. O RB0 pode ser utilizado também para gerar a interrupção externa, assim como os pinos de RB4 a RB7 podem gerar a interrupção por mudança de estado. Os pinos RB1 e RB2 também são utilizados para a comunicação serial (USART). Já o pino RB3 é utilizado no módulo

de CCP, para a saída do PWM. O pino RB6 pode ainda ser utilizado para incremento do TMR1 e, juntamente com o RB7, para a programação do microcontrolador.

Para que o microcontrolador possa funcionar, é necessária também a sua alimentação: são os pinos V_{SS} (GND) e V_{DD} ($+5V_{CC}$). A tensão de alimentação nominal dos PICs é de $5V_{CC}$, mas o ranger de variação desta tensão depende do modelo estudado. No caso do PIC16F628A, ela vai de 2.0 a $5.5V_{CC}$.

O oscilador externo deve ser ligado aos pinos OSC1 e OSC2. Os tipos de osciladores e suas ligações serão discutidos no Capítulo 9, assim como a utilização dos osciladores internos.

Temos ainda o pino denominado MCLR (barrado), que se refere ao Master Clear externo. Sempre que esse pino for colocado em nível lógico baixo (GND), o programa será resetado e o processamento paralisado. Ao ser colocado em nível alto ($+5V$), a execução do programa será retomada do ponto inicial.

Para entender melhor o significado das nomenclaturas utilizadas na identificação dos pinos, a tabela seguinte descreve os detalhes de cada uma delas.

Número do Pino	Função	Tipo Entrada	Tipo Saída	Descrição
17	RA0	ST	CMOS	I/O digital bidirecional.
	AN0	AN	-	Entrada analógica para os comparadores.
18	RA1	ST	CMOS	I/O digital bidirecional.
	AN1	AN	-	Entrada analógica para os comparadores.
1	RA2	ST	CMOS	I/O digital bidirecional.
	AN2	AN	-	Entrada analógica para os comparadores.
	V_{REF}	-	AN	Saída da tensão de referência programável.
2	RA3	ST	CMOS	I/O digital bidirecional.
	AN3	AN	-	Entrada analógica para os comparadores.
	CMP1	-	CMOS	Saída do comparador 1.
3	RA4	ST	OD	I/O digital bidirecional.
	TOCKI	ST	-	Entrada externa do contador TMRO.
	CMP2	-	OD	Saída do comparador 2.
4	RA5	ST	-	Entrada digital.
	MCLR	ST	-	Master Clear (reset) externo. O PIC só funciona quando este pino encontra-se em nível alto.
	V_{PP}		-	Entrada para tensão de programação (13V).
15	RA6	ST	CMOS	I/O digital bidirecional.
	OSC2	-	XTAL	Saída para cristal externo.
	CLKOUT	-	CMOS	Saída com onda quadrada em $\frac{1}{4}$ da freqüência imposta em OSC1 quando em modo RC. Essa freqüência equivale aos ciclos de máquina internos.

Número do Pino	Função	Tipo Entrada	Tipo Saída	Descrição
16	RA7	ST	CMOS	I/O digital bidirecional.
	OSC1	XTAL	-	Entrada para cristal externo.
	CLKIN	ST	-	Entrada para osciladores externos (híbridos ou RC).
6	RBO	TTL	CMOS	I/O digital bidirecional com pull-up interno.
	INT	ST	-	Entrada para interrupção externa.
7	RB1	TTL	CMOS	I/O digital bidirecional com pull-up interno.
	RX	ST	-	Recepção para comunicação USART assíncrona.
	DT	ST	CMOS	Via de dados para comunicação USART síncrona.
8	RB2	TTL	CMOS	I/O digital bidirecional com pull-up interno.
	TX	-	CMOS	Transmissão para comunicação USART assíncrona.
	CK	ST	CMOS	Via de clock para comunicação USART síncrona.
9	RB3	TTL	CMOS	I/O digital bidirecional com pull-up interno.
	CCP1	ST	CMOS	I/O para o Capture, Compare e PWM.
10	RB4	TTL	CMOS	I/O digital bidirecional com pull-up interno.
	PGM	ST	-	Entrada para programação em baixa tensão (5V).
	RB5	TTL	CMOS	I/O digital bidirecional com pull-up interno.
11	RB6	TTL	CMOS	I/O digital bidirecional com pull-up interno.
	T1OSO	-	XTAL	Interrupção por mudança de estado.
	T1CKI	ST	-	Entrada externa do contador TMR1.
	PGC	ST	-	Clock da programação serial (ICSP).
12	RB7	TTL	CMOS	I/O digital bidirecional com pull-up interno.
	T1OSI	XTAL	-	Interrupção por mudança de estado.
	PGD	ST	CMOS	Entrada para cristal externo para TMR1.
5	V _{SS}	P	-	GND.
14	V _{DD}	*	P	Alimentação positiva.

Legenda:

- P = Power (alimentação)
- = Não-utilizado
- TTL = Entrada tipo TTL
- ST = Entrada tipo Schmitt Trigger
- CMOS = Saída do tipo CMOS
- OD = Saída tipo Dreno Aberto (Open Drain)
- NA = Entrada/Saída analógica

CARACTERÍSTICAS ELÉTRICAS E OUTRAS

Temperatura de trabalho	-40°C até +125°C
Temperatura de armazenamento	-65°C até +150°C
Tensão de trabalho	3.0V a +5.5V
Voltagem máxima no pino V_{DD} (em relação ao V_{SS})	-0.3V até +6.5V
Voltagem máxima no pino MCLR (em relação ao V_{SS})	-0.3V até +14V
Voltagem máxima nos demais pinos (em relação ao V_{SS}).....	-0.3V até (V_{DD} + 0.3V)
Dissipação máxima de energia	800 mW
Corrente máxima de saída no pino V_{SS}	300 mA
Corrente máxima de entrada no pino V_{DD}	250 mA
Corrente máxima de entrada de um pino (quando em V_{SS}).....	25 mA
Corrente máxima de saída de um pino (quando em V_{DD})	25 mA
Corrente máxima de entrada em PORTA + PORTB	200 mA
Corrente máxima de saída em PORTA + PORTB	200 mA

MAPAS DAS MEMÓRIAS

Veja agora como estão organizadas as memórias de programação e de dados do PIC16F628A.

MEMÓRIA DE PROGRAMA

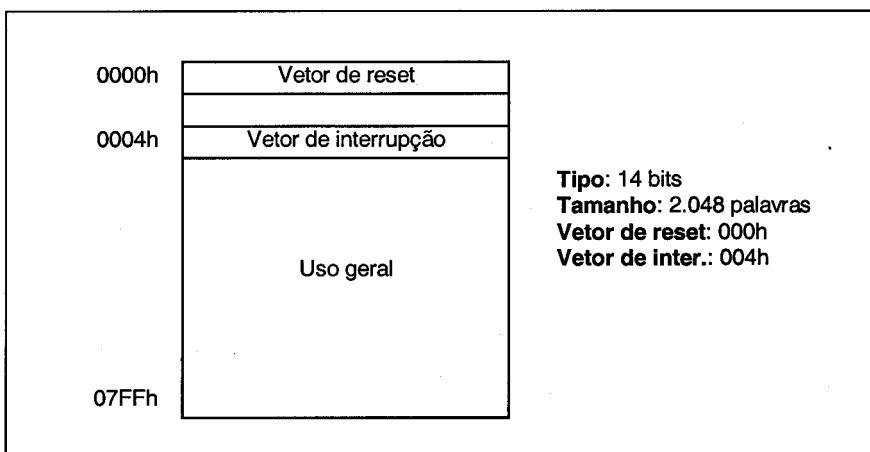


Figura 5.2 - Memória de Programa.

MEMÓRIA DE DADOS

Banco 0	Banco 1	Banco 2	Banco 3
000h INDF	080h INDF	100h INDF	180h INDF
001h TMR0	081h OPTION_REG	101h TMR0	181h OPTION_REG
002h PCL	082h PCL	102h PCL	182h PCL
003h STATUS	083h STATUS	103h STATUS	183h STATUS
004h FSR	084h FSR	104h FSR	184h FSR
005h PORTA	085h TRISA	105h	185h
006h PORTB	086h TRISB	106h PORTB	186h TRISB
007h	087h	107h	187h
008h	088h	108h	188h
009h	089h	109h	189h
00Ah PCLATH	08Ah PCLATH	10Ah PCLATH	18Ah PCLATH
00Bh INTCON	08Bh INTCON	10Bh INTCON	18Bh INTCON
00Ch PIR1	08Ch PIE1	10Ch	18Ch
00Dh	08Dh	10Dh	18Dh
00Eh TMR1L	08Eh PCON	10Eh	18Eh
00Fh TMR1H	08Fh	10Fh	18Fh
010h T1CON	090h	110h	190h
011h TMR2	091h		
012h T2CON	092h PR2		
013h	093h		
014h	094h		
015h CCPR1L	095h		
016h CCPR1H	096h		
017h CCP1CON	097h		
018h RCSTA	098h TXSTA		
019h TXREG	099h SPBRG		
01Ah RCREG	09Ah EEDATA		
01Bh	09Bh EEADR		
01Ch	09Ch EECON1		
01Dh	09Dh EECON2		
01Eh	09Eh		
01Fh CMCON	09Fh VRCQN	11Fh	
020h	0A0h Uso Geral 80 bytes	120h Uso Geral 48 bytes	
07Fh Uso Geral 96 bytes	0EFh	14Fh	
	0F0h Espelho do Banco 0	150h	
	0FFh	16Fh	
		17Fh Espelho do Banco 0	
			1EFh
			1F0h
			1FFh Espelho do Banco 0

Não implementado

Figura 5.3 - Memória de Dados.

OS REGISTRADORES ESPECIAIS QUE CONTROLAM TUDO

INTRODUÇÃO

Até aqui você já foi apresentado ao mundo dos microcontroladores e ao PIC16F628A especificamente, e em muitas vezes falamos sobre configurações que podem ser feitas para definir as portas como entradas ou saídas, ativar as interrupções, ativar a contagem do TMRO por sinal externo e muitas outras. Mas, afinal, onde ficam essas configurações? Como foi visto no capítulo de introdução à memória, o PIC possui uma série de registradores especiais que são denominados SFR (Special Function Registers) que servem exatamente para guardar a configuração e o estado de funcionamento atual da máquina. Veremos agora um apanhado geral sobre esse registradores. Mais detalhes serão comentados adiante, quando se fizer necessária a utilização destes registradores. Além disso, o Apêndice A traz um descriptivo detalhado de cada um deles.

GERAIS

STATUS E PCON

Esses registradores servem para mostrar o estado da ULA, a forma do último reset, configurar o oscilador interno e também para configurar a página de programação atual, quando necessário.

Provavelmente as funções mais utilizadas do registrador STATUS estão relacionadas às operações matemáticas, pois ele indica os estouros de registradores (C-Carry e DC-Digit Carry) e resultados iguais a zero (Z). É bom observar que, no caso da subtração, o Carry trabalha com a lógica invertida.

Quanto ao registrador PCON, a função mais utilizada diz respeito à escolha da freqüência do oscilador interno (OSCF): 37 KHz ou 4 MHz.

OPTION

Esse registrador serve para configurar uma série de opções para a operação do microcontrolador, tais como: habilitação dos pull-ups do PORTB, configurações do prescaler, configurações do TMR0 e seleção da borda para a interrupção externa.

Apesar deste registrador receber o nome Option, ele será referenciado em nossos programas pelo nome OPTION_REG, pois os PICs mais antigos possuíam uma instrução chamada Option.

INTCON, PIR1 E PIE1

Esses registradores servem para configurar e identificar todas as interrupções existentes no PIC16F628A.

O registrador INTCON opera com as interrupções principais, que são: Timer 0 (TMR0), Interrupção externa e Interrupção por mudança de estado. Além disso, ele possui a chave geral de todas as interrupções (GIE) e a chave geral para as interrupções de periféricos.

Os registradores PIR1 e PIE1 são os responsáveis pelo gerenciamento das interrupções de periféricos: EEPROM, Comparadores, USART (recepção e transmissão), CCP, Timer 1 (TMR1) e Timer 2 (TMR2).

CONHECENDO O PCL E PCLATH

O PCL é um registrador que armazena os 8 bits menos significativos do PC (Program Counter), o qual indica a próxima linha do programa que será executada no momento. A cada ciclo de máquina, o PC é automaticamente alterado, para que o programa possa ser executado normalmente. Esse registrador também pode ser alterado pelo programa, mas isso deve ser feito com extremo cuidado, para que o sistema não se perca e/ou trave.

PORTAS

TRIS

Esses registradores servem para configurar os pinos das portas como entrada ou saída. Quando é colocado "1" em um bit do TRIS, o pino relacionado a ele é configurado como entrada. Para configurar o pino como saída, você deve escrever "0" no bit relacionado. Uma maneira prática para memorizar essa regra é associar o "1" ao "I" de Input (entrada), e o "0" ao "O" de Output (saída). Para configurar o PORTA, deve ser utilizado o TRISA, e para configurar o PORTB, deve ser utilizado o TRISB.

PORT

Como já foi visto, o PIC16F628A possui duas portas: PORTA e PORTB. O estado dessas portas é acessado diretamente em duas posições distintas da memória. Quando um pino dessas portas é configurado como entrada, ao leremos o seu bit relacionado, encontraremos diretamente o nível lógico aplicado a esse pino. Da mesma maneira, ao configurarmos um pino como saída, podemos alterar o seu estado, escrevendo diretamente no bit relacionado. Fácil, não é?

CONTADORES

TIMER 0

O TMRO é um contador de 8 bits que pode ser acessado diretamente na memória, tanto para a leitura quanto para a escrita. A diferença entre ele e os demais registradores é que seu incremento é automático e pode ser feito pelo clock da máquina ou por um sinal externo. Vale lembrar que o estouro desse contador pode gerar uma interrupção.

TIMER 1

O TMR1, por sua vez, é um contador de 16 bits que também pode ser acessado diretamente na memória, tanto para a leitura quanto para a escrita. No entanto, devido ao seu tamanho, esse registrador é armazenado em dois endereços: TMR1H (parte alta) e TMR1L (parte baixa). Além disso, o registrador T1CON é o responsável pelas diversas configurações relacionadas ao Timer1, tais como: habilitação, prescaler, oscilador externo próprio, origem do incremento (interno ou externo) e sincronismo de incremento. O estouro desse contador pode gerar uma interrupção. Esse contador será utilizado também no módulo CCP.

TIMER 2

O TMR2 é outro contador de 8 bits que também pode ser acessado diretamente na memória, tanto para a leitura quanto para a escrita. Além disso, o registrador T2CON é o responsável pelas diversas configurações relacionadas ao Timer 2, tais como: habilitação, prescaler e protscaler. Para gerar uma interrupção, o valor desse contador é comparado ao valor especificado em outro registrador, o PR2, em vez de esperar o estouro do mesmo.

EEPROM

EEADR E EEDATA

O primeiro (EEADR) é o registrador onde será especificado o endereço para escrita ou leitura da EEPROM interna do PIC16F628A. O outro (EEDATA) possui duas funções distintas: nas operações de escrita da EEPROM, ele deve ser preenchido com o dado a ser armazenado, já nas operações de leitura, ele armazena o dado lido.

EECON1 E EECON2

Existem dois registradores de operação da EEPROM: EECON1 e EECON2. O EECON1 é o responsável pelas operações de escrita e leitura da EEPROM e detecção de erro.

Em relação ao EECON2, devemos comentar que não se trata de um registrador verdadeiramente implementado na memória. Ele só é utilizado durante a inicialização do ciclo de escrita na EEPROM por uma questão de segurança, evitando assim que a memória seja alterada accidentalmente. A função do EECON2 será melhor explicada no capítulo relativo à utilização da EEPROM.

MÓDULO CCP

CCP1CON, CCPR1H E CCPR1L

Através do módulo CCP é possível acessar três modos diferentes de operação: Capture, Compare e PWM. O registrador CCP1COM é o responsável pela configuração desse modo. O módulo CCP utiliza ainda o TMR1 como base de tempo e os registradores complementares CCPR1H (parte alta) e CCPR1L (parte baixa).

MÓDULO COMPARADOR

CMCON

O registrador CMCON é utilizado para a configuração dos dois comparadores internos existentes no PIC16F628A, possibilitando a utilização de uma entre as oito possibilidades de ligações elétricas disponíveis. Nesse registrador temos ainda a configuração de inversão das duas saídas e a leitura dos estados dessas saídas.

MÓDULO VOLTAGEM DE REFERÊNCIA

VRCON

O registrador VRCON é utilizado para configurar o módulo de voltagem de referência, que nada mais é do que uma saída analógica com 16 valores configuráveis por software. Por intermédio desse registrador é possível habilitar/desabilitar este sistema e selecionar o valor da saída (malha R2R interna).

MÓDULO USART

TXSTA E RCSTA

Esses registradores configuram e monitoram todas as possibilidades para a comunicação via módulo USART, um para a Transmissão (TXSTA) e outro para recepção (RCSTA).

SPBRG

Esse é o registrador responsável pela configuração do Baund Rate.

TXREG E RCREG

São os buffers (acumuladores) para os dados recebidos (RCREG) ou a serem enviados (TXREG).

ENDEREÇAMENTO INDIRETO

FSR E O INDF

O endereçamento indireto da memória será realmente estudado no capítulo referente à programação, mas já que estamos apresentando todos os registradores especiais, não poderíamos deixar esses de fora. O FSR é um registrador em que pode ser escrito um outro endereço de memória que será acessado indiretamente, como se ele fosse apenas um ponteiro. Já o INDF não é um registrador realmente verdadeiro; trata-se somente de um espelho do endereço apontado pelo FSR. Complicado? Não se preocupe, na hora certa, isso será devidamente explicado e exemplificado.

CAPÍTULO

7

CONHECENDO UM POUCO O SET DE INSTRUÇÕES

OS TERMOS UTILIZADOS

Para facilitar o aprendizado do set de instruções do PIC, é conveniente que você entenda corretamente os termos utilizados na construção dos nomes das instruções e seus argumentos. Vamos então conhecê-los:

- **Work:** Trata-se de um registrador temporário para as operações da ULA. No Assembler do PIC, ele é conhecido como W. Também é comum chamá-lo de acumulador.
- **File:** Referência a um registrador (posição de memória) propriamente dito. Utilizaremos a letra F para sua representação nos nomes de instruções e f nos seus argumentos.
- **Literal:** Um número qualquer que pode ser escrito na forma decimal, hexadecimal ou binária. Utilizaremos a letra L para sua representação nos nomes de instruções e k nos seus argumentos.
- **Destino:** O local onde deve ser armazenado o resultado da operação. Existem somente dois destinos possíveis: F, que guardará o resultado no próprio registrador passado como argumento; ou W, que colocará o resultado em Work. Na verdade, na sintaxe das instruções, o destino deve ser expresso pelos números 0 (W) e 1 (F). No entanto, como veremos mais adiante, as letras F e W são definidas no "include" para facilitar a programação.
- **Bit:** Refere-se a um bit específico dentro de um byte. Utilizaremos a letra B para sua representação nos nomes das instruções e b nos seus argumentos.
- **Teste:** Quando queremos testar o estado de um bit, para descobrirmos se ele é zero ou um. Utilizaremos a letra T para representá-lo nos nomes das instruções.
- **Skip:** Significa "pulo", e é utilizado para criar desvios, pulando a próxima linha. Utilizaremos a letra S para representá-lo nos nomes das instruções.
- **Set:** Refere-se ao ato de setar um bit, isto é, torná-lo equivalente a UM. Utilizaremos a letra S para representá-lo nos nomes das instruções.

- **Clear:** Refere-se ao "clear" de um bit, isto é, torná-lo equivalente a ZERO. Utilizaremos a letra C para representá-lo nos nomes das instruções.
- **Zero:** Algumas instruções podem gerar desvios se o resultado da operação efetuada for zero. Neste caso, utilizaremos a letra Z para indicar tal condição.

Todos os demais termos utilizados são específicos das ações realizadas pelas instruções e são praticamente auto-explicativos. Abaixo, eles aparecem conforme são utilizados:

ADD: Soma.

AND: Lógica "E".

CLR: Limpar, zerar (Clear).

COM: Complemento.

DEC: Decremento de uma unidade.

INC: Incremento de uma unidade.

IOR: Lógica "OU".

MOV: Mover, transferir para algum lugar.

RL: Rotacionar 1 bit para a esquerda (rotation left).

RR: Rotacionar 1 bit para a direita (rotation right).

SUB: Subtração.

SWAP: Inversão entre as partes alta e baixa de um registrador.

XOR: Lógica "OU exclusivo".

A CONSTRUÇÃO DOS NOMES DAS INSTRUÇÕES

Com base nos termos que você acabou de aprender, será muito mais fácil entender o significado de uma instrução por meio do seu nome, pois ele é composto pela junção desses termos. Por exemplo, digamos que você deseja decrementar o valor de um determinado registrador. A instrução que fará isso é composta pelos termos referentes à ação que você quer fazer:

- Decrementar (DEC) um registrador (F) = DECF

Agora vamos fazer a análise ao contrário, isto é, partindo do nome de uma instrução, vamos descobrir para que ela serve:

- **DECFSZ** = Decrementa (DEC) o registrador (F) e pula (S) se o resultado for zero (Z).

OS GRUPOS DE INSTRUÇÕES

Ficou muito fácil entender a lógica dos nomes das instruções do PIC, não é mesmo? Para facilitar ainda mais, organizaremos todas as 35 instruções do 16F628A em quatro grupos, conforme suas aplicações:

- *Operações com registradores;*
- *Operações com literais;*
- *Operações com bits; e*
- *Controles.*

O RESUMO DAS INSTRUÇÕES

Bem, agora você já está apto a conhecer todo o set de instruções do PIC16F628A. Reveja os termos apresentados para que você comprehenda corretamente a utilização de cada uma, facilitando também sua memorização.

Operações com registradores		
Instrução	Argumentos	Descrição
ADDWF	f,d	Soma W e f, guardando o resultado em d.
ANDWF	f,d	Lógica "E" entre W e f, guardando o resultado em d.
CLRF	f	Limpa f.
COMF	f,d	Calcula o complemento de f, guardando o resultado em d.
DECFSZ	f,d	Decrementa f, guardando o resultado em d e pula a próxima linha se o resultado for zero.
INCF	f,d	Incrementa f, guardando o resultado em d.
INCFSZ	f,d	Incrementa f, guardando o resultado em d, e pula a próxima linha se o resultado for zero.
IORWF	f,d	Lógica "OU" entre W e f, guardando o resultado em d.
MOVF	f,d	Move f para d (cópia).
MOVWF	f	Move W para f (cópia).
RLF	f,d	Rotaciona f 1 bit para a esquerda.
RRF	f,d	Rotaciona f 1 bit para a direita.
SUBWF	f,d	Subtrai W de f (f - W), guardando o resultado em d.
SWAPF	f,d	Executa uma inversão entre as partes alta e baixa de f, guardando o resultado em d.
XORWF	f,d	Lógica "OU exclusivo" entre W e f, guardando o resultado em d.

Operações com literais		
Instrução	Argumentos	Descrição
ADDLW	k	Soma k com W, guardando o resultado em W.
ANDLW	k	Lógica "E" entre k e W, guardando o resultado em W.
IORLW	k	Lógica "OU" entre k e W, guardando o resultado em W.
MOVlw	k	Move k para W.
SUBLW	k	Subtrai W de k (k - W), guardando o resultado em W.
XORLW	k	Lógica "OU exclusivo" entre k e W, guardando o resultado em W.

Operações com bits		
Instrução	Argumentos	Descrição
BCF	f,b	Impõe 0 (zero) ao bit b do registrador f.
BSF	f,b	Impõe 1 (um) ao bit b do registrador f.
BTFS	f,b	Testa o bit b do registrador f, e pula a próxima linha se ele for 0 (zero).
BTFS	f,b	Testa o bit b do registrador f, e pula a próxima linha se ele for 1 (um).
Controles		
Instrução	Argumentos	Descrição
CLRW	-	Limpa W.
NOP	-	Gasta um ciclo de máquina sem fazer absolutamente nada.
CALL	R	Executa a rotina R.
CLRWD	-	Limpa o registrador WDT para não acontecer o reset.
GOTO	R	Desvia para o ponto R, mudando o PC.
RETFIE	-	Retorna de uma interrupção.
RETLW	k	Retorna de uma rotina, com k em W.
RETURN	-	Retorna de uma rotina, sem afetar W.
SLEEP	-	Coloca o PIC em modo sleep (dormindo) para economia de energia.

CAPÍTULO

8

MPLab

INTRODUÇÃO À FERRAMENTA

O MPLab é um programa para PC, que roda sobre a plataforma Windows, e serve como ambiente de desenvolvimento de programas para PICs. Ele é uma ferramenta muito poderosa e um dos principais responsáveis pela popularização do PIC, pois junta, no mesmo ambiente, o gerenciamento de projetos, a compilação, a simulação, a emulação e a gravação do chip. Na maioria dos sistemas utilizados por outros microcontroladores, essas funções são executadas por programas separados, tornando o trabalho muito mais cansativo e demorado.

Nosso objetivo aqui não é a elaboração de um manual de operação do MPLab, pois isso fugiria ao nosso escopo principal. Por outro lado, não podemos deixar de registrar certos comentários sobre sua operação, pois ele é essencial para a programação de qualquer sistema apresentado neste livro.

A partir deste ponto, consideraremos então que você já possui o MPLab corretamente instalado em seu computador. As telas e os comentários apresentados referem-se à versão 6.22.

O AMBIENTE DE TRABALHO

Ao iniciarmos o MPLab, teremos acesso ao ambiente de trabalho global. Trata-se de uma área para abertura das janelas de trabalho, um menu superior e uma barra de ferramentas com diversos ícones relativos a funções específicas no momento.

No decorrer deste livro adotaremos o padrão de especificarmos o caminho, através dos menus, para encontrarmos os comandos desejados. Este caminho aparecerá sempre em itálico/negrito e separações pelo símbolo > (sinal de maior).

Por exemplo: *File > Open Workspace...*

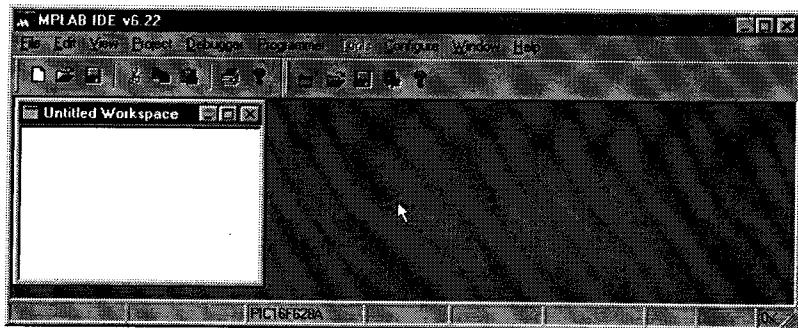


Figura 8.1 - Ambiente de Trabalho.

ABRINDO UMA ÁREA DE TRABALHO E UM PROJETO

Para que possamos trabalhar dentro deste ambiente, não basta o arquivo de código-fonte, é necessário ter muitas outras informações para que o sistema possa ser compilado e executado. Até a versão 6, o MPLab utilizava o conceito de projeto para armazenar todas essas informações necessárias à compilação de um sistema. A partir da versão 6, este conceito foi modificado. Passou a existir, então, além do Projeto, também a área de trabalho, conhecida como Workspace.

Um projeto é, então, um arquivo que guarda as informações básicas necessárias à compilação do sistema, tais como a relação de arquivos-fonte, opções de compilação e algumas ferramentas de compilação. O conceito de Workspace é ainda mais abrangente que o de Projeto, pois ele armazena todas as demais informações necessárias ao desenvolvimento de um sistema. No arquivo do Workspace serão armazenadas informações relacionadas a um ou mais projetos associados, qual está ativo no momento, quais as janelas abertas e suas posições na tela, configurações do ambiente de trabalho, etc.

O importante é saber que o MPLab não funciona adequadamente se um Workspace/Projeto não for aberto.

Caso você seja um iniciante no MPLab, provavelmente não terá nenhum Workspace já gravado. Neste caso, teremos de trabalhar com um novo. Observe que no seu ambiente de trabalho existe uma janela branca definida como "Untitled Workspace" no lado esquerdo. Esta será a janela do nosso Workspace. Em primeiro lugar, vamos definir um nome para ele e salvá-lo em um arquivo no nosso HD. Para isso, utilize o menu *File > Save Workspace....*

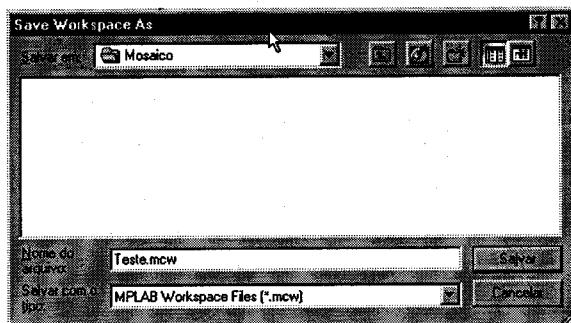


Figura 8.2 - Definição de Nome para o Workspace.

Uma vez denominados o diretório e o nome do workspace, será criado um arquivo com a extensão MCW onde serão armazenadas todas as suas informações de trabalho dentro do MPLab. Recomendamos que, antes de mais nada, você organize seu HD em diretórios coerentes onde devem ser guardados os arquivos de workspace, projetos e códigos-fonte. Em nosso exemplo, estamos utilizando um diretório chamado Mosaico.

Agora que você já possui um workspace, será necessário também criarmos um projeto para utilizá-lo dentro desse workspace. Para isso, accese o comando *Project > New...*. Como no caso anterior, especifique o diretório e o nome do arquivo desejado.

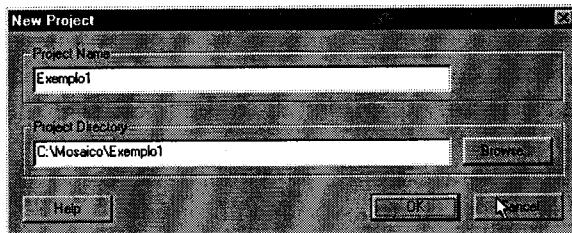


Figura 8.3 - Criação do Projeto.

Em nosso exemplo, escolhemos um subdiretório chamado Exemplo1 localizado dentro de Mosaico. Esta escolha serve para exemplificar que um workspace pode servir para trabalhar com projetos diferentes, organizados em diretórios específicos. É uma boa idéia a criação de uma estrutura de diretórios com uma pasta para cada projeto, organizando seus trabalhos e clientes. Para localizar a pasta correta dentro da sua estrutura de diretórios, utilize o botão *Browse...*. Será criado um arquivo com a extensão MCP.

Na sua área de trabalho será possível visualizar, agora, o workspace e o projeto aberto no momento, como pode ser observado na próxima figura.

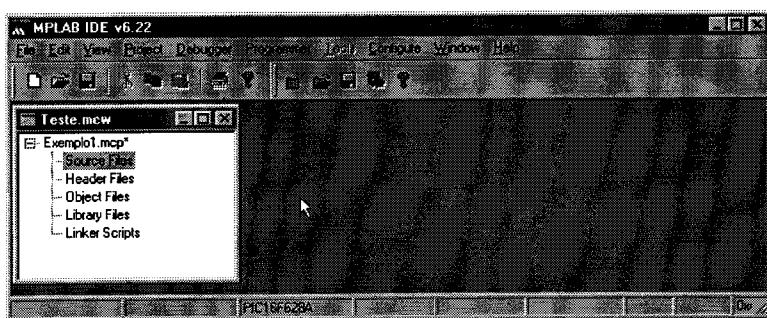


Figura 8.4 - Tela do Workspace com Projeto Aberto.

Caso você tente criar um projeto sem definir o workspace, este será definido automaticamente com o mesmo nome do projeto.

Para salvar todas as configurações atuais, você deve salvar o workspace através do comando *File > Save Workspace*. Através do comando *File > Save Workspace As...* é possível salvá-lo com outro nome e/ou em outro diretório e com o comando *File > Close Workspace* é possível fechá-lo, voltando à condição inicial de abertura do MPLab.

As mesmas opções são válidas para o arquivo de projeto, através dos comandos: *Project > Close*, *Project > Save Project* e *Project > Save Project As...*

ASSOCIANDO E ABRINDO UM ARQUIVO DE CÓDIGO-FONTE

O próximo passo será então criar um arquivo de código-fonte para o projeto aberto; ou, então, associar um já existente. Até o momento, não foi criado nenhum arquivo de código fonte, mas será mostrado como associá-lo ao projeto, aproveitando a ordem natural das coisas. Mais para a frente, será mostrado como criar um arquivo de código-fonte, e você já saberá como associá-lo ao projeto.

Através do comando *Project > Add Files to Project...* será possível associar arquivos ao projeto aberto no momento. Basta especificar o diretório e o nome do arquivo corretamente.

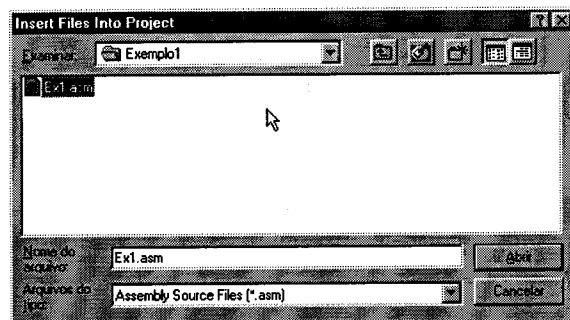


Figura 8.5 - Associando Arquivos ao Projeto.

Os arquivos de código-fonte que utilizaremos no decorrer deste livro serão sempre escritos para a linguagem assembly e, por esta razão, a extensão dos mesmos será ASM.

Depois disso, o nome do arquivo irá aparecer na tela do workspace, dentro do grupo denominado Source Files. Para abrir este arquivo dentro da área de trabalho, basta um duplo clique sobre seu nome.

O importante é saber que, para compilar um arquivo, não basta ele estar aberto na tela, é preciso que o nome do mesmo esteja associado ao projeto.



Figura 8.6 - Abrindo o Arquivo de Código-fonte.

Caso você queira associar outro arquivo ao projeto, é necessário antes eliminar a associação anterior. Para isso, utilize o comando *Project > Remove File From Project*. Esta tarefa de acrescentar e remover arquivos do projeto também pode ser feita facilmente através de um clique com o botão direito do mouse sobre o grupo *Source Files* ou sobre o nome do arquivo, na janela do workspace.

Para criar um arquivo de código-fonte, basta utilizar o comando *File > New*. Outra opção é a abertura de um arquivo existente através do comando *File > Open...* e a posterior gravação deste arquivo (como um modelo) com outro nome, ou em outro diretório, através do comando *File > Save As...*. Os comandos *File > Save* e *File > Save All* servem para gravar os arquivos abertos (ativo ou todos, respectivamente), apesar de que normalmente esses arquivos são salvos também sempre que o workspace ou o projeto são salvos, ou ainda toda vez que o sistema for compilado. Para fechar a janela de um arquivo, basta clicar no botão de fechamento (lado superior esquerdo) ou utilizar o comando *File > Close*.

CONFIGURANDO O WORKSPACE E O PROJETO

Agora que já preparamos todos os arquivos que deveriam ser preparados, incluindo o código-fonte, o projeto e o workspace, vejamos quais as configurações básicas necessárias para podermos compilar o sistema.

O primeiro passo e o mais importante de todos é a definição do microcontrolador que será utilizado. Essa escolha é primordial, pois o trabalho do compilador depen-

de dela. Isso significa que a compilação se baseará nos dados do modelo de PIC escolhido.

O PIC válido no momento pode ser verificado através da barra de status, na parte inferior da janela do MPLab. Para trabalhar com um modelo diferente, acesse o comando *Configure > Select Device...*

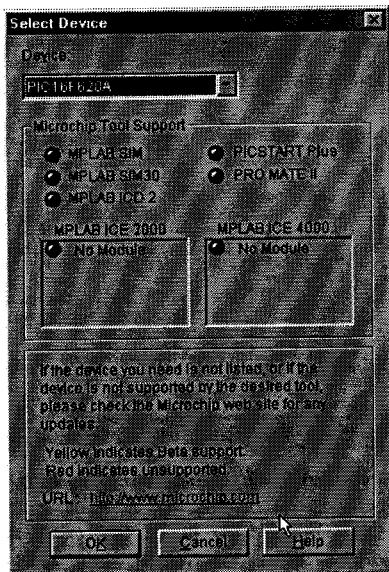


Figura 8.7 - Selecionando o PIC Desejado.

Na parte superior da tela (Device), você terá acesso a todos os modelos de PIC disponíveis. Cada vez que um novo modelo é lançado pela Microchip, uma nova versão do MPLab também é disponibilizada para que esta lista seja atualizada.

Baste selecionar o modelo com o qual trabalharemos no decorrer de todos os nossos estudos: PIC16F628A.

Na parte central desta tela, você poderá visualizar quais as ferramentas da Microchip estão disponíveis (LED verde) para o modelo selecionado. Estas ferramentas são o simulador (MPLab SIM), os Gravadores (lado direito) e os Emuladores. Obviamente, somente o simulador não é dependente de hardwares adicionais.

Depois de pressionado o botão OK, o MPLab demorará alguns segundos para reconfigurar todas as variáveis e parâmetros internos. O PIC selecionado aparecerá na parte inferior da tela e o sistema estará pronto para a compilação.

Uma maneira de efetuar todos os passos apresentados até aqui de maneira mais simples e seqüencial é através do comando *Project > Project Wizard...*. Basta seguir as telas e informar os dados solicitados, como o nome do projeto, o diretório e o arquivo-fonte associado.

COMPILANDO O PROJETO

Muito bem, neste momento seu projeto está pronto para ser compilado. Como você já deve saber, compilar um projeto significa processar o código-fonte para criar um arquivo que realmente vai ser entendido pelo microcontrolador. No MPLab, quando compilamos um arquivo ASM, ele é transformado em um HEX. Esse novo arquivo é o que realmente será gravado no PIC.

Para compilar um projeto, você deve utilizar o comando *Project > Make*, ou a tecla F10 para facilitar o trabalho. Caso nenhuma alteração tenha sido feita desde a última compilação, aparecerá uma mensagem informando que a compilação não é necessária. Se mesmo assim você quiser garantir o processo, a compilação deve ser feita através do comando *Project > Build All* (Ctrl+F10).

Uma janela irá mostrar a compilação sendo executada. Ao final desta, o sistema voltará ao ambiente de trabalho, só que com uma nova janela aberta: *Output*. Dentro da janela *Output*, a pasta *Build* irá apresentar um relatório sobre a compilação. O importante é que a compilação seja bem-sucedida, aparecendo a mensagem "BUILD SUCCEEDED" no final do relatório.

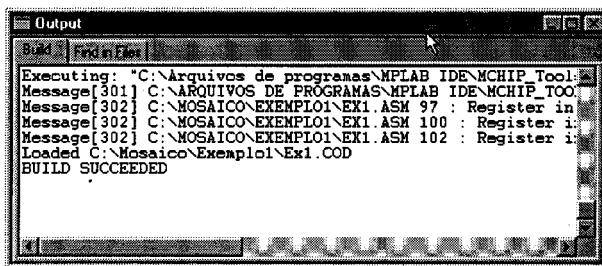


Figura 8.8 - Relatório de Compilação com Sucesso.

O problema é que nem sempre isso acontece. Neste caso, houve algum erro durante a compilação que deve ser resolvido. Mas não se preocupe, o MPLab irá ajudá-lo nesta tarefa também.

ERROS, WARNINGS E MENSAGENS

Existem três grupos de avisos que são emitidos nos relatórios de compilação:

- **Warning:** É uma espécie de alerta do compilador; algo está estranho e deve ser checado. Normalmente um Warning é seguido de um erro, mas ele sozinho não causa erro na compilação.
- **Error:** Quando aparece este aviso é porque algum erro realmente aconteceu, impedindo que a compilação tenha sucesso. Pode ser um erro de sintaxe, de argumentos, etc.

- **Message:** Trata-se apenas de um comunicado do compilador para o programador. Ele está avisando que alguma ação está sendo feita e que isso pode gerar um problema, se não for corretamente implementada. O aparecimento de mensagens é normal em alguns programas, mas elas não devem ser desprezadas.

The screenshot shows a Windows-style window titled "Output". Inside, there is a command-line interface output from the MPLAB IDE's assembler. The text reads:

```
Executing: "C:\Arquivos de programas\MPLAB IDE\MCU1P_Tools\mpasmwin.exe"
Message[301] C:\ARQUIVOS DE PROGRAMAS\MPLAB IDE\MCU1P_TOOLS\P16F84.INC 3
Warning[207] C:\MOSAICO\EXEMPLO1\EX1.ASM 78 : Found label after column 1
Error[122] C:\MOSAICO\EXEMPLO1\EX1.ASM 78 : Illegal opcode (INICIO)
Message[302] C:\MOSAICO\EXEMPLO1\EX1.ASM 97 : Register in operand not in
Message[302] C:\MOSAICO\EXEMPLO1\EX1.ASM 100 : Register in operand not in
Message[302] C:\MOSAICO\EXEMPLO1\EX1.ASM 102 : Register in operand not in
Halting build on first failure as requested.
BUILD FAILED
```

Figura 8.9 - Relatório de Compilação com Erro.

Para cada aviso do relatório será mostrado o seu tipo, seguido do código específico para a situação, o nome do arquivo, a linha na qual ele ocorreu e a sua descrição. Para facilitar ainda mais o seu trabalho, basta clicar com o mouse duas vezes sobre o aviso, que o MPLab já altera para a janela do código-fonte, na linha correta. Então, é só resolver o problema e compilar novamente.

CAPÍTULO

9

GRAVANDO O PIC

INTRODUÇÃO

De nada adianta elaborarmos e implementarmos um programa se ele não for gravado na memória do componente para que funcione corretamente. Para tal, existem diversas maneiras e ferramentas capazes de efetuar essa gravação, mas seria impossível nos referirmos a todas elas. Por isso, este capítulo será baseado no Picstart Plus, que é o gravador oficial de menor custo da Microchip, sendo totalmente compatível com o MPLab. Entretanto, este é um gravador relativamente caro, e por isso não é muito popular entre os programadores aqui no Brasil. Para resolver este problema, foi desenvolvido uma linha de gravadores de baixo custo totalmente nacionais. O modelo mais econômico e popular desta linha é o McFlash, um gravador muito barato que trabalha com todos os modelos FLASH da Microchip. O mais importante deste gravador é que ele opera exatamente igual ao Picstart Plus, isto é, dentro do MPLab. Desta forma, não é necessário instalar e aprender qualquer outro software, e nem mesmo mudar de ambiente. Toda a programação será feita no próprio MPLab, conforme veremos a seguir.

Consideraremos então que você possua um McFlash ou um Picstart Plus, e que ele esteja corretamente instalado em seu computador (conectado à porta serial).

CONFIGURANDO AS OPÇÕES DO PIC

O primeiro passo será você configurar corretamente as opções de gravação para o microcontrolador. Estas opções são conhecidas como Configuration Bits, e a janela para alterá-las pode ser acessada em *Configure > Configuration Bits...*

Address	Value	Category	Setting
5007	0FFF	Oscillator	PC C 4MHz
		Watchdog Timer	On
		Power Up Timer	Off
		Brown Out Detect	Enabled
		Master Clear Enable	Enabled
		Low Voltage Program	Enabled
		Data EE Read Protect	Disabled
		Code Protect	Off

Figura 9.1 - Tela dos Configurations Bits.

TIPO DE OSCILADOR

Existem dois grupos básicos de osciladores para uso com o PIC16F628A: internos e externos. Este modelo possui dois osciladores internos (37 KHz e 4 MHz - seleção por software) e capacidade para operar com vários tipos de osciladores externos. A escolha deve ser feita levando-se em conta o hardware do projeto. Os tipos de osciladores serão demonstrados no próximo capítulo. Veja agora as opções disponíveis:

- **RC_CLKOUT:** Para oscilador externo tipo RC com o pino 15 operando como CLKOUT, isto é, com uma onda quadrada de $\frac{1}{4}$ da freqüência.
- **RC_I/O:** Para oscilador externo tipo RC com o pino 15 operando como I/O (RA6).
- **INTOSC_CLKOUT:** Para oscilador interno com o pino 15 operando como CLKOUT, isto é, com uma onda quadrada de $\frac{1}{4}$ da freqüência.
- **INTOSC_I/O:** Para oscilador interno com o pino 15 operando como I/O (RA6).
- **EC_I/O:** Para clock externo (circuito auto-oscilante) com o pino 15 operando como I/O (RA6).
- **XT:** Para osciladores externos tipo cristal ou ressoadores.
- **HS:** Para cristais ou ressoadores externos com freqüências elevadas (acima de 4 MHz).
- **LP:** Para cristais ou ressoadores externos com baixas freqüências (abaixo de 200 KHz). Utilizado para minimizar o consumo.

WATCHDOG TIMER

O WDT também pode ser ativado ou não na hora da gravação, e esta configuração não poderá ser alterada posteriormente pelo programa. Por isso, lembre-se de que, para ativar essa opção, seu programa deve estar preparado para limpar o contador do WDT periodicamente. Caso contrário, seu programa será resetado toda vez que esse contador estourar.

POWER UP TIMER

O PIC16F628A possui o Power Up Timer (uma espécie de POR melhorado - veja no próximo capítulo) interno que pode ser habilitado ou não na hora da gravação. Esta opção irá fazer com que o PIC só comece a operar cerca de 72 ms após o pino MCLR ser colocado em nível alto. Se você estiver utilizando um circuito de POR melhorado (externo), então essa opção deve estar desativada.

BROWN OUT DETECT

No próximo, onde falamos especificamente do hardware, falaremos também de um circuito externo para Brown Out (BOR). No entanto, o modelo de PIC em estudo já possui esse circuito interno. Trata-se de um sistema de detecção automática de

baixa tensão capaz de resetar o PIC. Isso significa que, se a tensão de alimentação (V_{DD}) for menor que 4V (típico) por mais de $100\mu s$, o sistema será reiniciado. Esses dados foram retirados do data sheet, da seção de especificações elétricas são valores fixos. Para trabalhar com valores diferentes; será necessário desabilitar este recurso e montar um circuito externo de BOR ligado ao pino MCLR.

MASTER CLEAR ENABLE

Esta é opção que define o uso do pino 4, que pode ser I/O ou Master Clear externo (MCLR). Ao habilitar esta opção, o pino 4 funciona como MCLR.

LOW VOLTAGE PROGRAM

Este também é um recurso relativamente novo para muitos modelos de PIC. Trata-se do sistema de programação do PIC (gravação da memória de programa) em baixa tensão: 5V. Normalmente essa programação é habilitada por uma alta tensão (13V) no pino MCLR. Acontece que hoje é possível criarmos sistemas onde um PIC possa gravar o programa de outro PIC, ou então efetuarmos um upgrade remoto. Para facilitar esta implementação, a Microchip elaborou um sistema onde não é necessário os 13V, raramente disponíveis na maioria dos projetos. Assim, todo o processo utiliza somente o nível TTL, customizando o hardware. Entretanto, nem tudo são flores. Para que esse sistema funcione de forma robusta e eficaz, um pino deve ser dedicado à função de entrar no modo de programação. Quando habilitada esta opção, o pino 10 deixa de ser o RB4 e passa a operar como PGM.

DATA EE READ PROTECT

Com esta opção ativada, não será possível ler a memória de dados (EEPROM Interna) através do gravador do PIC. Durante o desenvolvimento, ou até em alguns tipos de projeto, sua leitura pode ser uma forma interessante de achar erros e solucionar problemas.

CÓDIGO DE PROTEÇÃO

Para a gravação em série é muito importante que essa opção esteja ativada, pois isso impedirá que qualquer pessoa consiga ler o programa gravado dentro do PIC. Esta é a única proteção que você terá para que ninguém possa "copiar" o seu sistema. No caso do PIC16F628A, que é regravável eletronicamente, não há problemas em deixar essa opção sempre ativa, mesmo durante a fase de desenvolvimento, pois esse código impedirá que você leia a memória (inclusive para o comando "Verify"), mas não impedirá que você grave outro programa por cima (desde que grave toda a memória de programa, e não só parte dela para agilizar o processo). Entretanto, tome muito cuidado se você estiver trabalhando com PICs janelados (apagáveis por luz ultravioleta), porque a gravação de um componente com essa opção ligada pode significar sua perda, pois pode ser que você nunca mais consiga regravá-lo.

DEFININDO AS CONFIGURAÇÕES NO PRÓPRIO PROGRAMA

Para evitarmos a chata tarefa de termos de configurar essas opções todas as vezes que vamos gravar um PIC, e também para evitarmos dúvidas sobre o correto estado de cada uma delas, é possível especificarmos esta escolha no próprio código do programa, por meio de uma diretriz de compilação. A diretriz _CONFIG (com 2 underlines) configura diretamente as opções de gravação. Para facilitar o trabalho com ela, os arquivos de include já definem nomes para as diversas opções. No caso do PIC16F628A, teremos as seguintes opções:

- _BOREN_ON: Para BOR ligado.
- _BOREN_OFF: Para BOR desligado.
- _CP_ON: Para code protection ligado.
- _CP_OFF: Para code protection desligado.
- _DATA_CP_ON: Para acesso externo à EEPROM habilitado.
- _DATA_CP_OFF: Para acesso externo à EEPROM desabilitado.
- _PWRTE_ON: Para Power Up ligado.
- _PWRTE_OFF: Para Power Up desligado.
- _WDT_ON: Para WatchDog ligado.
- _WDT_OFF: Para WatchDog desligado.
- _LVP_ON: Para sistema de programação em baixa tensão ativado.
- _LVP_OFF: Para sistema de programação em baixa tensão desativado.
- _MCLRE_ON: Para Master Clear externo ativado.
- _MCLRE_OFF: Para Master Clear externo desativado.
- _RC_OSC_CKOUT: Para RC externo com saída CKOUT.
- _RC_OSC_NOCKOUT: Para RC externo sem saída CKOUT (com I/O).
- _INTOSC_OSC_CKOUT: Para oscilador interno com saída CKOUT.
- _INTOSC_OSC_NOCKOUT: Para oscilador interno sem saída CKOUT (com I/O).
- _EXTCLK_OSC: Para clock externo sem saída CKOUT (com I/O).
- _LP_OSC : Para oscilador tipo LP.
- _XT_OSC : Para oscilador tipo XT.
- _HS_OSC : Para oscilador tipo HS.

A combinação destas opções deve ser feita por intermédio do operador & ("E").

Desta forma, a sintaxe da diretriz _CONFIG é a seguinte:

_CONFIG _CP_ON & _PWRTE_ON & _WDT_OFF & _INTOSC_OSC_NOCKOUT

GRAVAÇÃO DE IDs

O PIC possui ainda 4 bytes (posições 2000h a 2003h) que não são acessíveis ao programa, mas que podem ser gravados para uma identificação de versão, por exemplo. A grande vantagem desses bytes é que podem ser lidos mesmo com o code protection ativado, facilitando o rastreamento do software em casos de problemas futuros. Entretanto, esses 4 bytes são mais limitados do que parecem, pois só podem ser utilizados os 7 bits menos significativos de cada byte. Desta forma, apesar dos 4 bytes, só temos disponíveis 28 bits de informação. O valor a ser gravado nos IDs pode ser escolhido diretamente através do comando *Configure > ID Memory...*

CHECKSUM

O checksum é o resultado de uma somatória feita entre os dados da área de programação do PIC, e é utilizado para verificações e checagem da integridade desses dados. Você pode, por exemplo, ler o checksum de um PIC já gravado (mesmo com código de proteção ligado) e comparar ao checksum do seu programa compilado, para saber se ambos estão utilizando a mesma versão de código-fonte.

OPÇÕES DE GRAVAÇÃO

Algumas outras opções podem ser escolhidas e configuradas para uma correta gravação do PIC. Essas opções podem ser acessadas através do comando *Configure > Settings...* Antes de mais nada, especifique corretamente a porta serial na qual está conectado o seu gravador. No caso de não saber qual a porta correta, verifique a documentação do seu micro ou efetue diversas tentativas. O mais comum é que ele esteja ligado à porta COM1. Caso você possua um mouse serial, é capaz que o gravador esteja ligado à COM2, COM3 e COM4 raramente estão disponíveis nas máquinas atuais.

Você poderá ainda especificar quais as partes do PIC a serem trabalhadas, incluindo o endereço inicial e final da área de programa que você deseja. Com isso é possível acessar (gravação e leitura) somente o trecho da memória de programa que está sendo usado (para agilizar o processo) ou somente os fusíveis configuração ou somente os IDs ou ainda somente os dados na EEPROM. É possível também trabalhar com a combinação dessas opções.

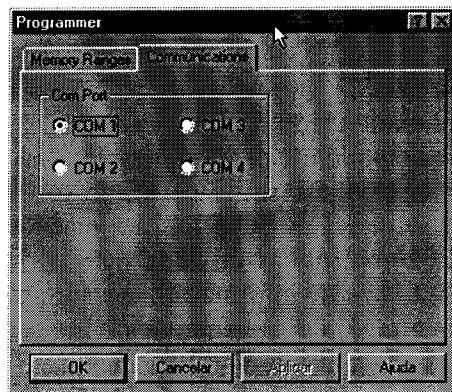


Figura 9.2 - Seleção da Porta Serial.

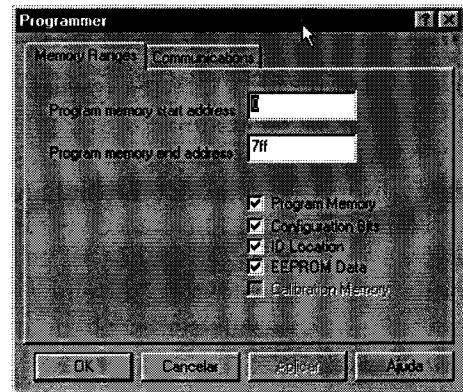


Figura 9.3 - Opções de Memória.

EFETUANDO A GRAVAÇÃO

Basta então você selecionar o gravador a partir do comando *Programmer > Select Programmer > Picstar Plus*.

Será aberta a janela *Output* com uma seção específica para o Picstart Plus.

Será necessário então habilitar o gravador através do comando *Programmer > Enable Programmer*. Este comando só deve ser acessado quando o gravador estiver devidamente conectado à porta serial e com a fonte de alimentação ligada. Caso contrário, uma mensagem de erro irá aparecer na tela *Output*.

Uma vez habilitado o gravador, serão habilitadas também todas as opções válidas dentro do menu *Programmer*, com as quais você poderá efetuar a gravação, leitura, verificação, etc. Na parte superior da tela aparecerá também alguns ícones relacionados a essas opções do menu, para facilitar o acesso. Ao lado, também na parte superior da tela, uma tabela de checagem das gravações estará disponível, informando o número de tentativas, o número de falhas e o total.

Bem, mas vamos logo ao que interessa. Para gravar o software que está na memória do PC (após uma compilação, por exemplo) para dentro do PIC que está no gravador, basta acessar o comando *Programmer > Program*. Esta opção irá não só gravar o PIC mas também verificar automaticamente cada dado gravado. Isso é necessário, pois, no caso de você estar utilizando o Code Protection, a verificação não será possível após o término da gravação.

Após a gravação, a janela *Output > Picstart* irá apresentar os resultados. No caso de falha, cheque a ligação do gravador, a alimentação, as conexões de cabos, a posição do PIC e tente novamente. Caso o erro continue, tente outro PIC ou consulte a documentação do seu gravador.

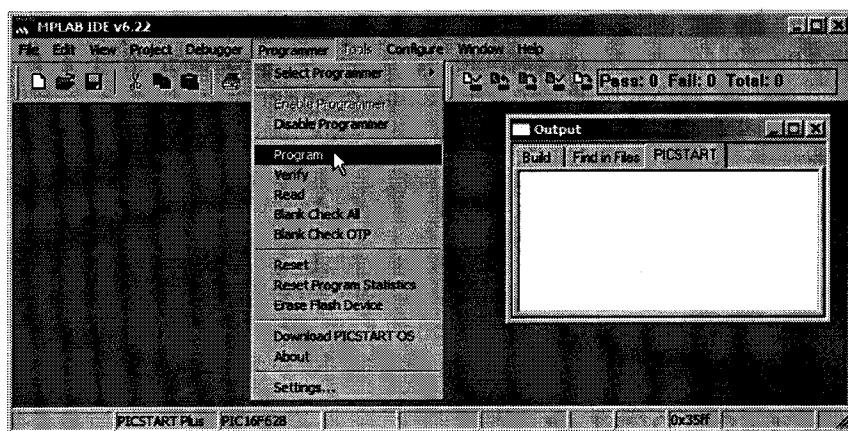


Figura 9.4 - Menu do Gravador.

Caso você não esteja utilizando o Code Protection (durante o desenvolvimento, por exemplo) e deseja verificar se o código gravado no PIC é exatamente o mesmo que está na memória do PC, poderá ser utilizado o comando *Programmer > Verify*. Este comando irá checar o valor existente em cada posição da memória de programa, apresentando um relatório das divergências encontradas.

Você também pode ler o código gravado no PIC. Para isso, faça uso do comando *Programmer > Read*. O conteúdo existente na memória de programa, nos IDs, nos fusíveis e até na EEPROM (se permitido) será carregado para dentro do MPLab. Através do menu *View*, as janelas que mostram esses dados poderão ser acessadas.

As opções de *Blank* são utilizadas para checar se um PIC está nas condições iniciais de fabricação, isto é, se ele está completamente apagado. A opção *Erase Flash Device* serve para apagar PICs da família FLASH. Esta opção deve ser utilizada antes de efetuarmos a gravação parcial de um PIC que havia sido gravado com Code Protection.

GRAVAÇÃO IN-CIRCUIT

Uma grande parte dos modelos da família PIC possibilita a chamada gravação "in-circuit". Isso serve para que o PIC possa ser gravado diretamente na placa montada. Como a gravação é feita de forma serial, utilizando somente cinco pinos, é possível a instalação de um soquete na placa para introduzir os sinais necessários à gravação. Isso exige uma série de cuidados no projeto do hardware, a fim de evitar que as tensões de gravação prejudiquem os demais periféricos. Está fora do nosso objetivo explicarmos exatamente como uma gravação "in-circuit" deve ser feita, mas, para aprofundar um pouco mais os seus conhecimentos, veja quais são os pinos necessários à gravação:

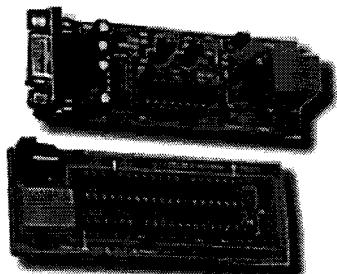
- V_{DD} : Alimentação de 5Vcc.
- V_{SS} : GND.

- **MCRL:** Tensão de programação. Para que o PIC entre em modo de programação, a tensão nesse pino deve ser de aproximadamente 13Vcc. Por isso, o resto do circuito ligado a esse pino deve ser protegido.
- **RB6:** Clock da comunicação serial imposto pelo gravador.
- **RB7:** Dados na comunicação serial, que podem ser impostos pelo gravador (escrita) ou pelo próprio PIC (leitura).

Uma vez que esses pinos estejam acessíveis, basta possuir um gravador que os gerencie corretamente. A gravação realizada pelo McFlash é totalmente compatível com a gravação "in-circuit", já que este já possui um cabo que pode ser conectado ao seu sistema. A pinagem deste cabo é fornecida na documentação do gravador.

GRAVADORES E OUTROS

Atualmente no Brasil o gravador mais interessante do mercado é, sem dúvida, o McFlash. Como já foi dito no decorrer deste capítulo, trata-se de um gravador nacional, de baixo custo e que opera diretamente junto com o MPLab. Este gravador, que pode ser visto na foto ao lado, é composto basicamente de duas placas: o gravador propriamente dito (placa superior) e o soquete de gravação (placa inferior). Neste soquete de gravação é colocado o PIC a ser gravado. O McFlash opera com todos os modelos da família FLASH, de 8 a 40 pinos (DIP). Junto com o gravador você receberá também a fonte de alimentação, o cabo de ligação entre as placas e o cabo de comunicação com o PC (serial).



O McFlash foi desenvolvido com o conceito de duas placas para facilitar a gravação de sistemas "in-circuit". Desta forma, todos os demais produtos da Mosaico possuem um soquete para a ligação direta do cabo que sai do gravador, dispensando o uso da placa de soquete. Você também poderá utilizar este mesmo conceito em seus próprios projetos.

Caso você necessite trabalhar com outros PICs que não são da família FLASH, opte pelo McPlus, um gravador que opera com quase todos os modelos de PIC.

CAPÍTULO

10

CONSIDERAÇÕES INICIAIS SOBRE O HARDWARE

ALIMENTAÇÃO

Agora que você já conhece um pouquinho do MPLab e como o PIC deve ser gravado, deve estar ansioso para fazer seus primeiros ensaios práticos, mas antes gostaríamos que você soubesse um pouco mais sobre o hardware básico para que um PIC funcione. Afinal, não adianta nada fazer um programa e não poder testá-lo na prática, não é mesmo?

O primeiro passo é em relação à alimentação. Quando foi vista a pinagem do PIC16F628A, falamos sobre os pinos V_{SS} e V_{DD} . Portanto, basta termos uma fonte segura de $5V_{CC}$, ligando o GND ao pino 5 e o $+5V$ ao pino 14. É importante porém, que essa fonte não tenha grandes variações de tensão (ripple) e ruídos. O PIC16F628A, apesar de ter sua tensão nominal de alimentação igual a $5V_{CC}$, pode ser alimentado de 3.0 a 5.50V, mas essa tensão não deve ficar variando durante sua utilização. Para garantir um melhor funcionamento, recomendamos que seja montado um capacitor de desacoplamento entre os pinos 5 e 14. Pela experiência adquirida nos projetos desenvolvidos pela Mosaico Engenharia, recomendamos que esse capacitor seja cerâmico, com valor entre 100pF e 100nF, e que seja posicionado o mais próximo possível dos pinos em questão.

OSCILADORES

O PIC16F628A possui dois osciladores internos e na grande maioria das vezes eles podem ser utilizados. O oscilador interno de 4 MHz (típico) é do tipo RC e possui uma precisão entre ± 1 e ± 5 , dependendo das condições de tensão e temperatura do sistema.

Entretanto, existirão casos em que será necessário a montagem de um oscilador externo, seja por motivo de precisão ou para poder trabalhar com uma frequência diferenciada. Só não se esqueça que, para a utilização de osciladores externos, 1 ou até 2 I/Os serão perdidos.

Existem basicamente quatro tipos de osciladores externos que podem ser utilizados com o PIC. A escolha de um deles dependerá da precisão e custos envolvidos.

RC

Esse é o tipo de oscilador mais simples que existe e também o mais barato, mas, por outro lado, é o menos preciso, variando muito com a tolerância dos componentes, tensão e temperatura. Para o PIC16F628A, esta utilização é pouco provável pois apresenta mais perdas que ganhos em relação ao RC interno. Um RC externo para o PIC deve ser montado, conforme a figura 10.1.

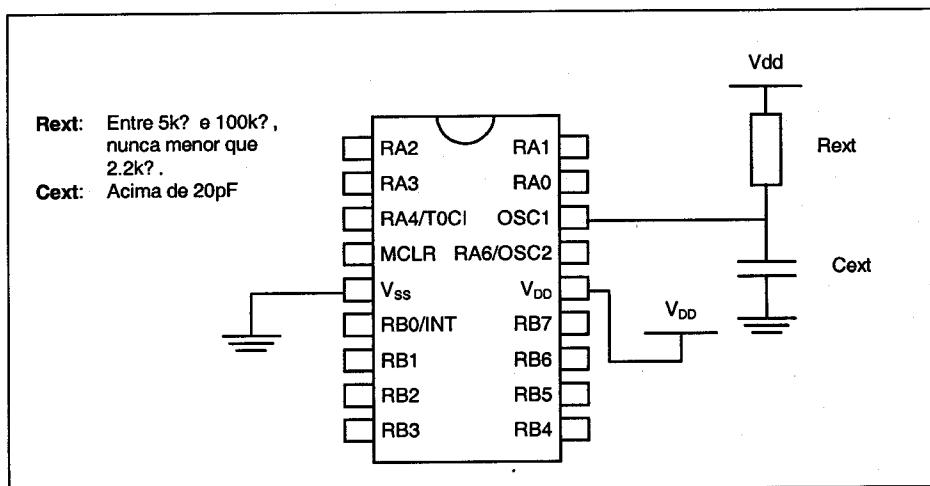


Figura 10.1 - Montagem de RC.

Neste caso, como é utilizado somente o pino OSC1, poderemos configurar o pino OSC2 para CLKOUT ou RA6. Como CLKOUT, este pino possuirá uma onda com freqüência quatro vezes menor que a freqüência do RC (Fosc), devido ao próprio hardware do PIC.

RESSOADOR

O ressoador cerâmico (também conhecido como ressonador) é uma segunda opção. Não é tão barato quanto um RC, mas é bem mais preciso e estável. Por outro lado, perde-se obrigatoriamente 2 I/Os para o circuito do oscilador.

Existem dois tipos de ressoadores, com três e dois pinos, que devem ser montados conforme os esquemas:

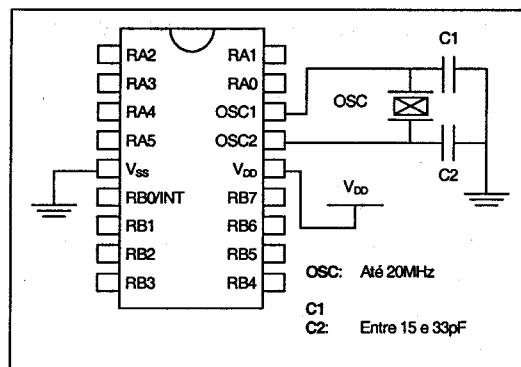


Figura 10.2 - Ressonador de dois Pinos.

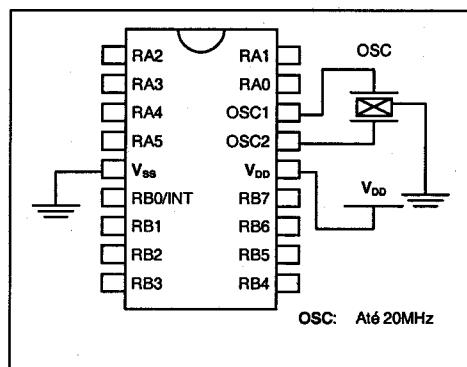


Figura 10.3 - Ressonador de três Pinos.

CRISTAL

Sem dúvida, os cristais são os osciladores mais precisos que podemos utilizar, mas também são os mais caros. Por isso, sua utilização só é realmente necessária quando prezamos muito pela precisão do sistema. O PIC já possui internamente o sistema de oscilação do cristal, por isso ele pode ser ligado diretamente aos pinos OSC1 e OSC2, com capacitores para melhorar a estabilidade, conforme o esquema:

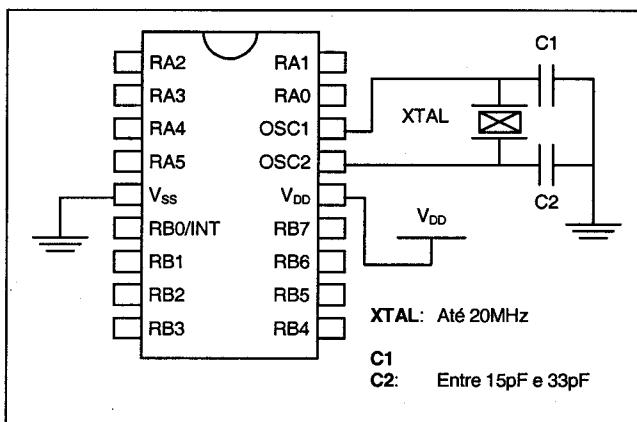


Figura 10.4 - Ligação de Cristal.

Como no caso dos ressoadores, para os cristais também são perdidos 2 I/Os.

HÍBRIDO OU CIRCUITOS DE OSCILAÇÃO

Podem ser utilizados ainda cristais híbridos ou circuitos próprios para oscilação, que devem ser ligados diretamente ao pino OSC1, como no esquema mostrado para o RC. Essa aplicação pode ser viável quando utilizamos mais de um microcontrolador no projeto e queremos garantir o sincronismo de oscilação para todos eles. Neste caso, esse sistema pode ser até mais barato que um cristal para cada PIC. Neste caso

também, como o circuito é ligado somente ao pino OSC1, não perde-se o segundo I/O, podendo ser utilizado como RA6.

POWER-ON RESET (POR) BÁSICO

Além da alimentação e do oscilador, para que o PIC possa funcionar, é necessário também criar um POR. Para o PIC16F628A existe um POR básico interno, quando o Master Clear Externo não está habilitado. Caso o MCLR esteja habilitado, será necessário criar um POR através de hardware complementar. O sistema mais básico para isso é a ligação do pino 4 (MCLR) diretamente ao V_{DD}. Mais adiante, demonstraremos outros circuitos de POR mais elaborados.

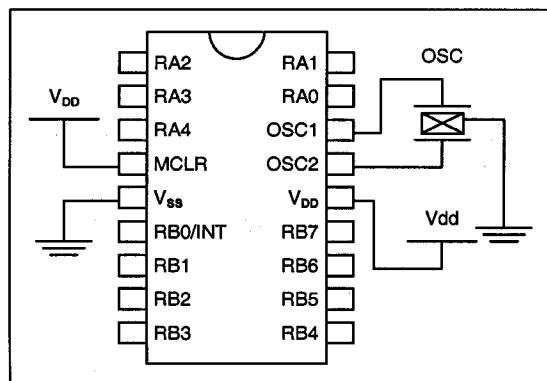


Figura 10.5 - Circuito POR Básico.

CAPÍTULO

11

PROGRAMAÇÃO

criando um programa

Finalmente, chegou o momento de falarmos da programação do PIC. Até aqui já foram vistas praticamente todas as teorias e conceitos necessários para que você tenha um bom rendimento durante o aprendizado da programação. Nossa intenção é que este capítulo lhe passe as informações necessárias para que você crie seus próprios programas e projetos. Vamos começar, então, com algumas dicas sobre a montagem do arquivo de código-fonte.

ESTRUTURANDO O CÓDIGO-FONTE

Para que um programa seja escrito e funcione corretamente, basta que as instruções certas sejam colocadas na ordem correta. Esta expressão é totalmente verdadeira, mas se só tivermos isso no código do programa, apesar de ele ser funcional, não será eficiente. Isso porque ele não estará devidamente estruturado e padronizado, dificultando futuras alterações e/ou o entendimento por outros programadores. Pode ser que em um futuro próximo, você também não consiga mais entender seu próprio programa. Por isso, recomendamos que, desde o primeiro código-fonte, você tenha sempre uma preocupação com a estruturação e organização desse arquivo.

Os anos de trabalho nos laboratórios da Mosaico Engenharia, dedicados à programação do PIC, nos possibilitaram a elaboração de uma estrutura eficiente, o que não significa que ela seja a ideal. Isso depende muito do próprio estilo do programador. Com base na nossa experiência, mostraremos então um exemplo dessa estruturação, que poderá ser adaptada ou modificada de acordo com as suas necessidades. Reforçamos somente a idéia de que a subdivisão e identificação de todas as tarefas (definição de variáveis, entradas, saídas, rotinas, sub-rotinas, programa principal, etc.) é a maneira mais utilizada pela grande maioria dos programadores.

A IMPORTÂNCIA DOS COMENTÁRIOS

Outro recurso tão importante quanto a estruturação do arquivo são os comentários inseridos pelo programador. Esses comentários nada mais são do que textos explicativos que serão desconsiderados pelo compilador na criação do arquivo executável. Isso faz, então, com que os comentários ocupem espaço somente no código-fonte, não afetando o tamanho do arquivo final. Por isso, quanto mais comentado estiver um programa, mais fácil será para você (ou qualquer outra pessoa) entendê-lo posteriormente.

No compilador do PIC, para escrever um comentário, basta iniciá-lo com um ponto-e-vírgula (;) em qualquer local do seu programa. Recomendamos que esses comentários estejam escritos sempre na mesma indentação, ao lado direito das instruções. O correto é que praticamente todas as linhas sejam comentadas e recomendamos ainda que isso seja feito ao mesmo tempo que a programação. Escrever um programa inteiro e só depois comentá-lo pode não ser uma técnica muito eficiente, embora seja possível. Com o tempo você descobrirá qual a melhor maneira de escrever comentários eficientes. Todos os exemplos dados neste livro estão devidamente comentados. Esses comentários dão uma boa base para que você crie os seus próprios.

É importante informá-lo também que o compilador do MPLab pode diferenciar as letras maiúsculas das minúsculas. Para evitar problemas com isso, recomendamos que todo o código seja escrito com somente um tipo de letra. Na Mosaico nos acostumamos a escrever nossos códigos somente em letras maiúsculas.

Para a correta indentação dos comandos e comentários, recomendamos a utilização de tabulação no lugar de espaços, apesar de o compilador ignorar ambos.

ARQUIVOS DE DEFINIÇÃO: INCLUDES

A fim de padronizar e agilizar ainda mais a programação, existe a possibilidade de criarmos e utilizarmos arquivos de definições, que foram chamados pela Microchip de arquivos "Includes". Esses arquivos nada mais são do que arquivos de texto, ou mesmo código-fonte, que serão inclusos no seu programa. Desta forma, a própria Microchip já criou um arquivo include para cada tipo de microcontrolador, em que estão definidos os nomes e endereços de todos os SFRs e uma série de outras definições necessárias para a utilização dos microcontroladores. Com esses arquivos evita-se a redigitação de todas essas informações na hora de começar um novo programa.

Os arquivos de includes devem ser gravados com a extensão INC. Futuramente, você também poderá criar seus próprios arquivos de definições. Inicialmente, recomendamos que sejam utilizados os arquivos padrão da Microchip, pois eles são disponibilizados durante a instalação do MPLab (arquivos includes personalizados devem ser sempre copiados com os arquivos-fontes, no caso de transferir o projeto para outro computador).

Para utilização de um arquivo de definições, a seguinte sintaxe deve ser utilizada:

```
#INCLUDE      <nome_do_arquivo.inc>
```

Do modo como foi demonstrado acima, o arquivo deve estar localizado no mesmo diretório de instalação do MPLab. Esta é a melhor forma de referenciar-se aos arquivos fornecidos pela Microchip.

Para os arquivos personalizados, deve ser escrita, além do nome do arquivo, sua localização completa. Neste caso, os símbolos < e > são substituídos por aspas ("").

```
#INCLUDE      "Drive:diretório\name_do_arquivo.inc"
```

Veja agora o arquivo de include da Microchip para o PIC16F628A (P16F626A.INC).

```
LIST
; P16F628A.INC Standard Header File, Version 1.10 Microchip Technology, Inc.
; NOLIST

; This header file defines configurations, registers, and other useful bits
; of
; information for the PIC16F628A microcontroller. These names are taken to
; match
; the data sheets as closely as possible.

; Note that the processor must be selected before this file is
; included. The processor may be selected the following ways:

;     1. Command line switch:
;         C:\ MPASM MYFILE.ASM /PIC16F628A
;     2. LIST directive in the source file
;         LIST P=PIC16F628A
;     3. Processor Type entry in the MPASM full-screen interface
;=====

; Revision History
;=====

;Rev: Date: Reason:
;1.01 14 Nov 2002 Updated to reflect BOD terminology changed to BOR
;1.00 22 Aug 2002 Initial Release
;=====

; Verify Processor
;=====

IFNDEF __16F628A
    MESSG "Processor-header file mismatch. Verify selected
processor."
    ENDIF
;=====

; Register Definitions
```

```
;-----  
W EQU H'0000'  
F EQU H'0001'
```

----- Register Files -----

INDF	EQU	H'0000'
TMR0	EQU	H'0001'
PCL	EQU	H'0002'
STATUS	EQU	H'0003'
FSR	EQU	H'0004'
PORTA	EQU	H'0005'
PORTB	EQU	H'0006'
PCLATH	EQU	H'000A'
INTCON	EQU	H'000B'
PIR1	EQU	H'000C'
TMR1L	EQU	H'000E'
TMR1H	EQU	H'000F'
T1CON	EQU	H'0010'
TMR2	EQU	H'0011'
T2CON	EQU	H'0012'
CCPR1L	EQU	H'0015'
CCPR1H	EQU	H'0016'
CCP1CON	EQU	H'0017'
RCSTA	EQU	H'0018'
TXREG	EQU	H'0019'
RCREG	EQU	H'001A'
CMCON	EQU	H'001F'
OPTION_REG	EQU	H'0081'
TRISA	EQU	H'0085'
TRISB	EQU	H'0086'
PIE1	EQU	H'008C'
PCON	EQU	H'008E'
PR2	EQU	H'0092'
TXSTA	EQU	H'0098'
SPBRG	EQU	H'0099'
EEDATA	EQU	H'009A'
EEADR	EQU	H'009B'
EECON1	EQU	H'009C'
EECON2	EQU	H'009D'
VRCON	EQU	H'009F'

----- STATUS Bits -----

IRP	EQU	H'0007'
RP1	EQU	H'0006'
RPO	EQU	H'0005'
NOT_TO	EQU	H'0004'
NOT_PD	EQU	H'0003'
Z	EQU	H'0002'
DC	EQU	H'0001'
C	EQU	H'0000'

----- INTCON Bits -----

GIE	EQU	H'0007'
PEIE	EQU	H'0006'
TOIE	EQU	H'0005'
INTE	EQU	H'0004'
RBIE	EQU	H'0003'
TOIF	EQU	H'0002'
INTF	EQU	H'0001'
RBIF	EQU	H'0000'

;----- PIR1 Bits -----

EEIF	EQU	H'0007'
CMIF	EQU	H'0006'
RCIF	EQU	H'0005'
TXIF	EQU	H'0004'
CCP1IF	EQU	H'0002'
TMR2IF	EQU	H'0001'
TMR1IF	EQU	H'0000'

;----- T1CON Bits -----

T1CKPS1	EQU	H'0005'
T1CKPS0	EQU	H'0004'
T1OSCEN	EQU	H'0003'
NOT_T1SYNC	EQU	H'0002'
TMR1CS	EQU	H'0001'
TMR1ON	EQU	H'0000'

;----- T2CON Bits -----

TOUTPS3	EQU	H'0006'
TOUTPS2	EQU	H'0005'
TOUTPS1	EQU	H'0004'
TOUTPS0	EQU	H'0003'
TMR2ON	EQU	H'0002'
T2CKPS1	EQU	H'0001'
T2CKPS0	EQU	H'0000'

;----- CCP1CON Bits -----

CCP1X	EQU	H'0005'
CCP1Y	EQU	H'0004'
CCP1M3	EQU	H'0003'
CCP1M2	EQU	H'0002'
CCP1M1	EQU	H'0001'
CCP1M0	EQU	H'0000'

;----- RCSTA Bits -----

SPEN	EQU	H'0007'
RX9	EQU	H'0006'
SREN	EQU	H'0005'
CREN	EQU	H'0004'
ADEN	EQU	H'0003'
FERR	EQU	H'0002'
OERR	EQU	H'0001'
RX9D	EQU	H'0000'

;----- CMCON Bits -----

C2OUT	EQU	H'0007'
-------	-----	---------

C1OUT	EQU	H'0006'
C2INV	EQU	H'0005'
C1INV	EQU	H'0004'
CIS	EQU	H'0003'
CM2	EQU	H'0002'
CM1	EQU	H'0001'
CM0	EQU	H'0000'

----- OPTION Bits -----

NOT_RBPU	EQU	H'0007'
INTEDG	EQU	H'0006'
T0CS	EQU	H'0005'
T0SE	EQU	H'0004'
PSA	EQU	H'0003'
PS2	EQU	H'0002'
PS1	EQU	H'0001'
PS0	EQU	H'0000'

----- PIE1 Bits -----

EEIE	EQU	H'0007'
CMIE	EQU	H'0006'
RCIE	EQU	H'0005'
TXIE	EQU	H'0004'
CCP1IE	EQU	H'0002'
TMR2IE	EQU	H'0001'
TMR1IE	EQU	H'0000'

----- PCON Bits -----

OSCF	EQU	H'0003'
NOT_POR	EQU	H'0001'
NOT_BO	EQU	H'0000'
NOT_BOR	EQU	H'0000'
NOT_BOD	EQU	H'0000'; Backwards compatibility to 16F62X

----- TXSTA Bits -----

CSRC	EQU	H'0007'
TX9	EQU	H'0006'
TXEN	EQU	H'0005'
SYNC	EQU	H'0004'
BRGH	EQU	H'0002'
TRMT	EQU	H'0001'
TX9D	EQU	H'0000'

----- EECON1 Bits -----

WRERR	EQU	H'0003'
WREN	EQU	H'0002'
WR	EQU	H'0001'
RD	EQU	H'0000'

----- VRCON Bits -----

VREN	EQU	H'0007'
VROE	EQU	H'0006'
VRR	EQU	H'0005'

```

VR3          EQU      H'0003'
VR2          EQU      H'0002'
VR1          EQU      H'0001'
VR0          EQU      H'0000'

;=====
;      RAM Definition
;=====

__MAXRAM H'01FF'
__BADRAM H'07'-H'09', H'0D', H'13'-H'14', H'1B'-H'1E'
__BADRAM H'87'-H'89', H'8D', H'8F'-H'91', H'93'-H'97', H'9E'
__BADRAM H'105', H'107'-H'109', H'10C'-H'11F', H'150'-H'16F'
__BADRAM H'185', H'187'-H'189', H'18C'-H'1EF'

;=====
;      Configuration Bits
;=====

__BODEN_ON      EQU      H'3FFF';Backwards compatibility to 16F62X
__BODEN_OFF     EQU      H'3FBF';Backwards compatibility to 16F62X
__BOREN_ON      EQU      H'3FFF'

__BOREN_OFF     EQU      H'3FBF'
__CP_ON          EQU      H'1FFF'
__CP_OFF         EQU      H'3FFF'
__DATA_CP_ON    EQU      H'3EFF'
__DATA_CP_OFF   EQU      H'3FFF'
__PWRTE_OFF     EQU      H'3FFF'
__PWRTE_ON      EQU      H'3FF7'
__WDT_ON          EQU      H'3FFF'
__WDT_OFF         EQU      H'3FFB'
__LVP_ON          EQU      H'3FFF'
__LVP_OFF         EQU      H'3F7F'
__MCLRE_ON       EQU      H'3FFF'
__MCLRE_OFF      EQU      H'3FDF'
__RC_OSC_CLKOUT EQU      H'3FFF'
__RC_OSC_NOCLKOUT EQU      H'3FFE'
__ER_OSC_CLKOUT EQU      H'3FFF';Backwards compatibility to 16F62X
__ER_OSC_NOCLKOUT EQU      H'3FFE';Backwards compatibility to 16F62X
__INTOSC_OSC_CLKOUT EQU      H'3FFD'
__INTOSC_OSC_NOCLKOUT EQU      H'3FFC'
__INTRC_OSC_CLKOUT EQU      H'3FFD';Backwards compatibility 16F62X
__INTRC_OSC_NOCLKOUT EQU      H'3FFC';Backwards compatibility 16F62X
__EXTCLK_OSC    EQU      H'3FEF'
__HS_OSC          EQU      H'3FEE'
__XT_OSC          EQU      H'3FED'
__LP_OSC          EQU      H'3FEC'

```

LIST

CONSTANTES E DEFINIÇÕES: EQU E DEFINES

Ao observarmos o arquivo mostrado anteriormente, encontramos, além dos comentários, a palavra EQU. Na verdade, EQU não é um comando do PIC, mas sim uma diretriz para o compilador. Essa diretriz associa um nome a um número. Desta maneira, fica muito mais fácil nos referencermos a uma variável pelo seu nome, no lugar do seu endereço na memória. Isso também é muito utilizado para definirmos constantes que serão utilizadas no decorrer da programação. Quando for necessário alterar o valor dessa constante, basta modificarmos o número relacionado ao seu nome. Simples, não é mesmo?

A sintaxe para a utilização do EQU é a seguinte:

nome_da_variável	EQU	Endereço_da_memória
------------------	-----	---------------------

ou

nome_da_constante	EQU	Valor_da_constante
-------------------	-----	--------------------

Aproveitamos o momento também para informá-lo de que um número qualquer pode ser representado de várias formas dentro do assember do PIC:

Decimal: D'???' ou.??

Hexadecimal: H'???' ou 0X??

Binário: B'??????????'

ASCII: A'?'

Vejamos, então, um exemplo utilizado no arquivo P16F628A.INC:

STATUS EQU	H'0003'
FSR	EQU H'0004'
PORTE	EQU H'0005'
PORTB	EQU H'0006'

Na hora de compilar o código-fonte, toda vez que aparecer a palavra PORTA, ela será automaticamente substituída pelo número 05 (em hexa), que é o endereço de memória para a porta A. Desta maneira, é muito mais fácil programar utilizando o nome PORTA no lugar do número 05, não é mesmo? Isso torna as coisas mais fáceis ainda quando nos referimos às variáveis de usuário ou constantes, que devem ser posteriormente alteradas. Neste caso, é muito mais simples alterar o valor em uma definição do que no programa inteiro.

Para esse mesmo tipo de aplicação existe também a diretriz #DEFINE. Esta, no entanto, é um pouco mais poderosa que a EQU, pelo fato de que ela não substitui nomes somente por números, mas sim por expressões inteiras (substituição de texto).

Essa diretriz é comumente utilizada para definirmos os pinos de entrada e saída do sistema. Para nos referenciarmos a um pino, devemos especificar o bit correto dentro do registrador da porta em questão. Podemos, então, dar um nome ao conjunto registrador/bit para facilitar o entendimento do programa. Por exemplo:

```
#DEFINE LED PORTB,1
```

LED é o nome da definição, e PORTB,1 é o que será considerado toda vez que o nome for utilizado durante o programa. Desta forma, após a utilização do define, podemos nos referir ao pino RB1 com a palavra LED. Isso também facilita muito as alterações futuras no hardware. Caso esse LED seja alterado para o pino RB2, basta alterarmos o #DEFINE, sem precisarmos modificar o código inteiro. Futuramente, veremos que o #DEFINE pode ser utilizado também para a criação de pequenos comandos.

EXEMPLO 0 - ESTRUTURAÇÃO

Vejamos agora um exemplo da estruturação completa. Este modelo será utilizado, no decorrer deste livro, como ponto de partida para todos os programas.

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                               NOME DO PROJETO
;*                               CLIENTE
;*                               DESENVOLVIDO PELA MOSAICO ENGENHARIA E CONSULTORIA
;* VERSÃO: 1.0                      DATA: 17/06/03
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                               DESCRIÇÃO DO ARQUIVO
;* -----
;*      MODELO PARA PIC 16F628A
;*
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                               ARQUIVOS DE DEFINIÇÕES
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
#INCLUDE <P16F628A.INC> ;ARQUIVO PADRÃO MICROCHIP PARA 16F628A

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

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                               PAGINAÇÃO DE MEMÓRIA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;DEFINIÇÃO DE COMANDOS DE USUÁRIO PARA ALTERAÇÃO DA PÁGINA DE MEMÓRIA
#define BANK0 BCF STATUS,R0 ;SETA BANK 0 DE MEMÓRIA
#define BANK1 BSF STATUS,R0 ;SETA BANK 1 DE MAMÓRIA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                               VARIÁVEIS
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS
```

```
; PELO SISTEMA
CBLOCK 0x20           ;ENDEREÇO INICIAL DA MEMÓRIA DE
                      ;USUÁRIO
W_TEMP               ;REGISTRADORES TEMPORÁRIOS PARA USO
STATUS_TEMP          ;JUNTO ÀS INTERRUPÇÕES

;NOVAS VARIÁVEIS

ENDC                 ;FIM DO BLOCO DE MEMÓRIA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS FLAGS UTILIZADOS PELO SISTEMA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODAS AS CONSTANTES UTILIZADAS PELO SISTEMA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO ENTRADA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO SAÍDA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; VETOR DE RESET
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

ORG     0x00          ;ENDEREÇO INICIAL DE PROCESSAMENTO
GOTO    INICIO

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; INÍCIO DA INTERRUPÇÃO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; ENDEREÇO DE DESVIO DAS INTERRUPÇÕES. A PRIMEIRA TAREFA É SALVAR OS
; VALORES DE "W" E "STATUS" PARA RECUPERAÇÃO FUTURA

ORG     0x04          ;ENDEREÇO INICIAL DA INTERRUPÇÃO
MOVWF  W_TEMP         ;COPIA W PARA W_TEMP
SWAPF   STATUS,W
MOVWF  STATUS_TEMP    ;COPIA STATUS PARA STATUS_TEMP

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; ROTINA DE INTERRUPÇÃO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; AQUI SERÁ ESCRITA AS ROTINAS DE RECONHECIMENTO E TRATAMENTO DAS
; INTERRUPÇÕES
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; ROTINA DE SAÍDA DA INTERRUPÇÃO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

; OS VALORES DE "W" E "STATUS" DEVEM SER RECUPERADOS ANTES DE
; RETORNAR DA INTERRUPÇÃO

SAI_INT

```
SWAPF STATUS_TEMP,W
MOVWF STATUS           ;MOVE STATUS_TEMP PARA STATUS
SWAPF W_TEMP,F
SWAPF W_TEMP,W         ;MOVE W_TEMP PARA W
RETFIE
```

*
* ROTINAS E SUBROTINAS *
*
; CADA ROTINA OU SUBROTINA DEVE POSSUIR A DESCRIÇÃO DE FUNCIONAMENTO
; E UM NOME COERENTE ÀS SUAS FUNÇÕES.

SUBROTINA1

```
;CORPO DA ROTINA
RETURN
```

*
* INÍCIO DO PROGRAMA *
* *

INICIO

```
BANK1          ;ALTERA PARA O BANCO 1
MOVLW B'00000000'
MOVWF TRISA      ;DEFINE ENTRADAS E SAÍDAS DO PORTA
MOVLW B'00000000'
MOVWF TRISB      ;DEFINE ENTRADAS E SAÍDAS DO PORTB
MOVLW B'10000100'
MOVWF OPTION_REG ;DEFINE OPÇÕES DE OPERAÇÃO
MOVLW B'00000000'
MOVWF INTCON     ;DEFINE OPÇÕES DE INTERRUPÇÕES
BANK0          ;RETORNA PARA O BANCO
MOVLW B'00000111'
MOVWF CMCON      ;DEFINE O MODO DE OPERAÇÃO DO COMP. ANALOG.
```

*
* INICIALIZAÇÃO DAS VARIÁVEIS *
* *

*
* ROTINA PRINCIPAL *
* *

MAIN

```
;CORPO DA ROTINA PRINCIPAL
GOTO MAIN
```

*
* FIM DO PROGRAMA *
* *

END

Algumas novas diretrizes de compilação apareceram no exemplo e devem ser explicadas:

- **ORG:** Trata-se de um direcionamento para a posição de memória de programação. Só devemos nos preocupar com esse endereçamento no início do programa (vetor de reset), no início das interrupções (vetor de interrupção) e em alguns casos específicos de paginação da área de programa, que não é muito importante no caso do PIC16F628A. Observe que, como o vetor de reset do PIC16F628A é o endereço 0x00, antes da escrita da primeira instrução é dada a diretriz ORG 0x00, para que o programa comece neste ponto. Logo em seguida, um pulo é dado para uma rotina localizada bem mais à frente (GOTO INICIO). Isso é necessário porque no endereço 0x04 deve ser iniciada a rotina de interrupção. Para tal, uma nova diretriz é especificada: ORG 0x04.
- **END:** Essa diretriz deve ser sempre colocada ao final do programa, pois, quando o compilador encontrá-la, a compilação será terminada. Para uma melhor estruturação, sempre deixamos a rotina principal no final do arquivo, seguida somente pelo END.
- **CBLOCK e ENDC:** É uma maneira simplificada de definirmos vários EQUs com endereços seqüenciais. Observe que utilizamos este recurso na definição das variáveis do sistema, mas precisamos informar somente o endereço da primeira variável (CBLOCK 0x20). As demais são definidas na seqüência. Isso possibilita mudarmos facilmente o local de todo o bloco de variáveis. Este recurso é utilizado, por exemplo, quando estamos convertendo o programa de um modelo de PIC para outro, cuja RAM de usuário inicia-se em um endereço diferente.

Os demais itens apresentados neste exemplo e que ainda não são conhecidos referem-se a comandos que serão vistos nas próximas seções. O Apêndice B explica todas as diretrizes existentes.

TRABALHANDO COM A MEMÓRIA

O objetivo desta seção é a apresentação da operação e dos comandos relacionados à memória do PIC. Como guardar e recuperar valores e também movê-los de um lugar para outro.

O REGISTRADOR WORK (W OU ACUMULADOR)

Para reforçar os conceitos já vistos, vale lembrá-lo de que o PIC possui um registrador temporário utilizado nas operações da ULA e não faz parte direta da memória RAM do sistema. Esse registrador é o Work (W) e será extremamente utilizado de agora em diante, já que não podemos ler ou escrever diretamente na memória sem o uso dele.

CONHECENDO OS BANCOS DE MEMÓRIA

Como também já foi visto no capítulo de memórias, o PIC16F628A possui quatro bancos de memória para os registradores SFRs e para a memória de variáveis do sistema. Por isso, quando queremos acessar algum registrador SFR que está no banco 1, por exemplo, devemos primeiro informar ao sistema que queremos trabalhar com esse banco.

Para tal, deve-se alterar o valor dos bits RP0 e RP1 no registrador STATUS. A combinação desses dois bits (chaves) possibilita selecionar um dos quatro bancos como o ativo no momento. A tabela seguinte relaciona o valor desses bits com o banco selecionado:

Banco	RP1	RP0
0	0	0
1	0	1
2	1	0
3	1	1

Sempre que o PIC é ligado, tanto RP1 quanto RP0 são iniciados com o valor 0 (zero). Desta forma, o banco 0 sempre é o primeiro selecionado. Caso desejemos trabalhar somente com os bancos 0 e 1, como em todos os exemplos apresentados neste livro, podemos então nos preocuparmos somente com o bit RP0.

Partindo desta premissa e com o objetivo de tornar essa tarefa muito mais fácil, criamos dois comandos virtuais chamados BANK0 e BANK1, que são definidos no arquivo-modelo da seguinte maneira:

```
#DEFINE      BANK0  BCF STATUS,RP0 ;SETA BANK 0 DE MEMÓRIA
#DEFINE      BANK1  BSF STATUS,RP0 ;SETA BANK 1 DE MEMÓRIA
```

Assim sendo, quando escrevermos BANK0, estaremos na verdade mandando limpar o bit RP0 do registrador STATUS; e quando escrevermos BANK1, estaremos mandando setar esse mesmo bit. Recomendamos, no entanto, que, após efetuadas as alterações necessárias no banco 1, o sistema retorne sempre para o banco 0.

Por exemplo, observe como foram ajustados alguns registradores do banco 1 no arquivo-modelo:

```
BANK1           ;ALTERA PARA O BANCO 1
MOVLW          B'00000000'
MOVWF          TRISB      ;DEFINE ENTRADAS E SAÍDAS DO PORTB
MOVLW          B'00000000'
MOVWF          TRISA      ;DEFINE ENTRADAS E SAÍDAS DO PORTA
MOVLW          B'10000100'
MOVWF          OPTION_REG ;DEFINE OPÇÕES DE OPERAÇÃO
MOVLW          B'00000000'
```

MOVWF BANK0	INTCON	;DEFINE OPÇÕES DE INTERRUPÇÕES ;RETORNA PARA O BANCO 0
----------------	--------	---

Caso os comandos BANK0 e BANK1 não existissem, o sistema ficaria assim:

BSF	STATUS, RP0	;ALTERA PARA O BANCO 1
MOVLW	B'00000000'	
MOVWF TRISB	B'00000000'	;DEFINE ENTRADAS E SAÍDAS DO PORTB
MOVLW	B'00000000'	
MOVWF TRISA	B'10000100'	;DEFINE ENTRADAS E SAÍDAS DO PORTA
MOVLW	B'00000000'	
MOVWF OPTION_REG	B'00000000'	;DEFINE OPÇÕES DE OPERAÇÃO
BCF	STATUS, RP0	;DEFINE OPÇÕES DE INTERRUPÇÕES ;RETORNA PARA O BANCO 0

Lembre-se, entretanto, de que esta dica só serve se você nunca acessar os bancos 2 e 3, ou seja, se você nunca alterar o valor de RP1, mantendo-o sempre em 0 (zero). De fato, o uso de outros bancos raramente será necessário, pois se você observar o mapa da memória (vide Capítulo 3), poderá observar que só existe algo diferente no banco 2, e trata-se dos últimos 48 bytes de registradores de uso geral. Por isso, você só se preocupará com este banco, caso seu programa necessite de mais de 176 bytes de RAM (valor disponível nos bancos 0 e 1).

Mesmo assim, caso seja necessário trabalhar com todos os bancos, recomendamos substituir o uso da implementação dos comandos BANK0 e BANK1 por outro sistema. Na verdade, o sistema que será apresentado a seguir poderá ser utilizado em qualquer caso, e para qualquer modelo de PIC, mantendo a padronização da programação, pois faz uso de uma diretriz do compilador. Trata-se da diretriz BANKSEL (vide apêndices).

O único problema é que o uso de BANKSEL muitas vezes consome mais memória de programa que a versão apresentada anteriormente, pois ela sempre acerta os valores de RP0 e RP1, independentemente da necessidade. Isso quer dizer que, se o sistema já se encontra no banco 0 e você deseja ir para o banco 1, bastava setar o bit RP0. Entretanto, a diretriz BANKSEL irá escrever duas linhas de código, tanto setando RP0 (1) quanto limpando RP1 (0).

Para utilizar BANKSEL, basta respeitar a seguinte sintaxe:

BANKSEL reg ;ONDE reg É O NOME DO REGISTRADOR COM A QUAL SE QUER ;TRABALHAR
--

Simples, não é mesmo? Assim sendo, se desejamos trabalhar com o registrador TRISA, por exemplo, basta antes escrever o seguinte comando:

BANKSEL TRISA ;TRISA ENCONTRA-SE NO BANCO 1

Como TRISA encontra-se no banco 1, o compilador irá escrever automaticamente as seguintes linhas no seu programa compilado:

BCF STATUS,RP1 ;LIMPA RP1 -> BANCO 0 OU 1
BSF STATUS,RP0 ;SET RP0 -> BANCO 1

Para tornar o uso dessa diretriz ainda mais fácil de entender ao ler o seu código, podemos definir quatro nomes com os quatro endereços iniciais de cada banco. Assim, para chavear entre os bancos não precisaremos associar o nome do registrador, e sim esses endereços iniciais dos bancos. Por exemplo:

BANK0 EQU 0X00 ;ENDEREÇO INICIAL DO BANCO 0
BANK1 EQU 0X80 ;ENDEREÇO INICIAL DO BANCO 1
BANK2 EQU 0X100 ;ENDEREÇO INICIAL DO BANCO 2
BANK3 EQU 0X180 ;ENDEREÇO INICIAL DO BANCO 3
BANKSEL BANK1 ;ALTERA PARA O BANCO 1
MOVlw B'00000000'
MOVWF TRISB ;DEFINE ENTRADAS E SAÍDAS DO PORTB
BANKSEL BANK3 ;ALTERA PARA O BANCO 3
BANKSEL BANK0 ;RETORNA AO BANCO 0

LIDANDO COM DADOS (MOVlw, MOVWF, MOVF, CLRF E CLRw)

Vamos aproveitar o exemplo anterior para conhecermos também os três comandos mais utilizados nos programas para PIC. Trata-se do **MOVlw**, **MOVWF** e **MOVF**. Vamos utilizar aqueles termos já estudados para conhecermos a verdadeira função desses comandos?

MOVlw = Move (MOV) uma literal (L) para o registrador work (W).

MOVWF = Move (MOV) o valor de work (W) para um registrador (F).

MOVF = Move (MOV) o valor de um registrador (F) para um local de destino passado como argumento (f ou w).

As sintaxes dessas instruções são:

MOVlw k ;ONDE k É O NÚMERO QUE SERÁ COLOCADO EM W.
MOVWF f ;ONDE f É O ENDEREÇO DA MEM. ONDE SERÁ GUARDADO O VALOR DE W.
MOVF f,d ;ONDE f É O REGISTRADOR QUE SERÁ MOVIDO PARA O DESTINO d ;LEMBRE-SE QUE EXISTEM DOIS DESTINOS POSSÍVEIS: W E F.

Assim sendo, para guardar um número em uma posição de memória, devemos primeiro movê-lo para o work (**MOVlw**) e depois movê-lo do work para o registrador propriamente dito (**MOVWF**).

Por exemplo, para definirmos as condições de operação da máquina, devemos ajustar os bits do registrador **OPTION_REG**. A maneira mais rápida de fazermos isso é movendo um número para o endereço desse registrador. O número é representado na forma binária para compreendermos melhor o estado de cada bit. Lembre-se de que pela sintaxe do comando **MOVWF** deveríamos especificar o

endereço do registrador. No entanto, no arquivo de include, esse endereço foi sobrecarregado no nome **OPTION_REG**.

MOVLW	B'10000100'	
MOVWF	OPTION_REG	; DEFINE OPÇÕES DE OPERAÇÃO

Já a instrução **MOVF** é utilizada para movermos o conteúdo de um registrador para outro registrador. Por exemplo, veja como fica o código para escrevermos em **TEMPO2** o mesmo valor existente em **TEMPO1**:

MOVF	TEMPO1,W	; MOVE O VALOR DE TEMPO 1 PARA O W (WORK)
MOVWF	TEMPO2	; MOVE O VALOR DE W (TEMPO1) PARA TEMPO2

Uma dúvida que pode surgir agora é em relação ao outro destino possível que pode ser utilizado com a instrução **MOVF**. Para que moveríamos o valor de um registrador para ele mesmo? Por mais idiota que possa parecer essa ação, ela tem alguma aplicação que será vista mais adiante.

Para limparmos um registrador, poderíamos mover 0 (zero) para ele, mas como você já deve ter observado, uma operação de movimento de um número para um registrador exige duas instruções (**MOVLW** e **MOVWF**), e portanto, dois ciclos de máquina. A fim de agilizar esse caso específico de atualização do registrador, existe o comando **CLRF**:

CLRF = Limpa (CLR) o registrador (F)

A sintaxe dessa instrução é:

CLRF	f	; ONDE f É O ENDEREÇO DA MEMÓRIA QUE SE DESEJA LIMPAR
------	---	---

Desta forma, as instruções:

MOVLW	B'00000000'	
MOVWF	TRISB	; DEFINE ENTRADAS E SAÍDAS DO PORTB

Poderiam ser substituídas por:

CLRF	TRISB	; DEFINE ENTRADAS E SAÍDAS DO PORTB
------	-------	-------------------------------------

De maneira análoga, existe um comando similar para a limpeza do work, apesar do fato de que para mover 0 (zero) para o work também só é necessária uma instrução. Entretanto, as duas instruções que executam essa ação são ligeiramente diferentes, pois uma afeta o **STATUS** de zero e a outra não.

MOVLW	0x00	; LIMPA O WORK SEM AFETAR O STATUS
CLRW		; TAMBÉM LIMPA O WORK E AFETA O STATUS DE ZERO

INICIALIZANDO O SISTEMA

Esta seção é destinada a uma melhor explicação e exemplificação da estrutura apresentada no arquivo-modelo.

DEFININDO LOCAIS PARA AS VARIÁVEIS

Seguindo a estrutura existente no nosso modelo, o primeiro bloco refere-se à definição das variáveis. Isso porque os SFRs já estão definidos no arquivo de include (vide P16F628A.INC).

Desta forma, devemos criar espaço e nomes para todas as variáveis que utilizaremos no programa. Recomendamos também comentar para que serve cada uma delas. Vejamos um exemplo:

```
;*****  
;*          VARIÁVEIS  
;* *****  
; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS  
; PELO SISTEMA  
  
    CBLOCK 0x0C      ;ENDEREÇO INICIAL DA MEMÓRIA DE USUÁRIO  
  
    CONTADOR        ;CONTADOR PARA O NÚMERO DE TRANSMISSÕES  
    BYTE_TX         ;BYTE QUE SERÁ TRANSMITIDO  
    BYTE_RX         ;BYTE QUE SERÁ RECEBIDO  
  
    ENDC           ;FIM DO BLOCO DE MEMÓRIA
```

Neste exemplo, estamos usando as diretrizes CBLOCK e ENDC. Sem sua utilização, o mesmo código seria escrito da seguinte maneira:

```
;*****  
;*          VARIÁVEIS  
;* *****  
; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS  
; PELO SISTEMA  
  
CONTADOR      EQU      0X0E      ;CONTADOR PARA O NÚMERO DE TRANSMISSÕES  
BYTE_TX       EQU      0X0F      ;BYTE QUE SERÁ TRANSMITIDO  
BYTE_RX       EQU      0X10     ;BYTE QUE SERÁ RECEBIDO
```

Lembre-se sempre de que a memória disponível ao usuário para o PIC16F628A vai de 0x20 até 0x7F (Banco 0), de 0xA0 até 0xEF (Banco 1) e de 0x120 até 0x14F (Banco 2).

RESERVANDO ESPAÇO PARA FLAGS

Flags são bits que definimos dentro de um byte para serem utilizados como chaves on/off. Desta forma, em um único endereço de memória (registrador) poderemos guardar até 8 flags que registrarão oito estados diferentes. Por exemplo, um flag

pode marcar se um byte já foi transmitido ou não, outro pode marcar se existe algum dado recebido ou não, e assim por diante.

A primeira ação para podermos trabalhar com flags é a definição de um registrador onde eles serão armazenados. Se muitos flags forem necessários, mais de um registrador pode ser utilizado.

```
;***** VARIÁVEIS *****
; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS
; PELO SISTEMA

CBLOCK 0x0C           ;ENDEREÇO INICIAL DA MEMÓRIA DE USUÁRIO

FLAG                 ;REGISTRADOR PARA FLAGS
CONTADOR             ;CONTADOR PARA O NÚMERO DE TRANSMISSÕES
BYTE_TX              ;BYTE QUE SERÁ TRANSMITIDO
BYTE_RX              ;BYTE QUE SERÁ RECEBIDO

ENDC                ;FIM DO BLOCO DE MEMÓRIA
```

Depois, fica mais fácil se definirmos nomes específicos para cada um dos flags por meio da diretriz **#DEFINE**:

```
;***** FLAGS INTERNOS *****
; DEFINIÇÃO DE TODOS OS FLAGS UTILIZADOS PELO SISTEMA

#define TRANSMITIDO FLAG,0    ;FLAG PARA INFORMAR QUE O DADO
                           ;FOI TRANSMITIDO:
                           ; 1 -> TRANSMITIDO
                           ; 0 -> NÃO TRANSMITIDO

#define RECEBIDO FLAG,1      ;FLAG PARA INFORMAR QUE O DADO
                           ;FOI RECEBIDO:
                           ; 1 -> RECEBIDO
                           ; 0 -> NÃO RECEBIDO
```

Desta forma, o flag TRANSMITIDO fica armazenado no bit 0 do registrador FLAG, e o RECEBIDO fica no bit 1.

CRIANDO CONSTANTES

As constantes são muito úteis para simplificar alterações em valores do sistema. Por exemplo, imagine que seu sistema deva possuir vários delays de atraso durante a execução. Vamos supor que você crie todos esses delays baseados em uma constante de tempo. Se ao final do projeto você descobrir que esses delays devem ser alterados, não fica muito mais fácil modificar somente a constante, ao invés de editar todo o código?

Com a criação de constantes, você poderá substituir um número (literal) por um nome qualquer:

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODAS AS CONTANSTES UTILIZADAS PELO SISTEMA
TEMPO_DELAY EQU .20 ;TEMPO DE DELAY DO SISTEMA EM SEG.
;PODE VARIAR DE 10 A 50 SEG.
FREQ_PISCA EQU .50 ;CONTANTE PARA A FREQUÊNCIA DE
;PISCADA DO LED (50=1Hz)
```

DEFININDO AS ENTRADAS E SAÍDAS

Antes de começar a escrever seu programa, é bom estar claro em sua mente (e de preferência também no papel) como será o hardware necessário ao sistema. Por isso, neste ponto você já deve saber em quais pinos do PIC serão ligadas as suas entradas e saídas. Para tornar tudo mais fácil e comprehensível, serão dados nomes a esses pinos por meio da diretiva **#DEFINE**.

Por exemplo, suponhamos que seu projeto necessite de um botão ligado ao pino RA0, um buzzer ligado ao pino RA1 e 8 LEDs ligados aos pinos de RB0 a RB7. Poderíamos então definir as entradas e saídas da seguinte maneira:

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO ENTRADA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)
#define BOTAO PORTA,0 ;BOTÃO DE START DO SISTEMA (PINO 17)
; 0 -> BOTÃO SOLTO
; 1 -> BOTÃO PRESSIONADO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO SAÍDA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)
#define SOM PORTA,1 ;BUZZER (PINO 18)
; 0 -> SEM SOM
; 1 -> COM SOM
#define LED0 PORTB,0 ;LED 0 - (PINO 6)
; 0 -> APAGADO
; 1 -> ACESO
#define LED1 PORTB,1 ;LED 1 - (PINO 7)
#define LED2 PORTB,2 ;LED 2 - (PINO 8)
#define LED3 PORTB,3 ;LED 3 - (PINO 9)
#define LED4 PORTB,4 ;LED 4 - (PINO 10)
#define LED5 PORTB,5 ;LED 5 - (PINO 11)
#define LED6 PORTB,6 ;LED 6 - (PINO 12)
#define LED7 PORTB,7 ;LED 7 - (PINO 13)
```

Lembre-se de que o demonstrado acima é somente uma definição de nomes para facilitar a programação. Não estamos ainda configurando os referidos pinos como entradas ou saídas propriamente ditas. Para tal, posteriormente configuraremos os registradores TRISA e TRISB.

O VETOR DE RESET

Como também já foi visto, o PIC possui um endereço para o qual o programa é desviado toda vez que um reset ocorre, seja ele pela energização do sistema, pelo Master Clear externo (/MCLR) ou pelo estouro de WDT. Esse endereço é chamado de vetor de reset. No caso do PIC16F628A, esse vetor localiza-se no endereço 0x00, mas em alguns modelos mais antigos, ele pode estar em outro lugar (final da área de programação). Portanto, a maneira que recomendamos para iniciar a escrita do programa pode ser vista no próximo exemplo:

No exemplo dado, a seguinte configuração foi efetuada:

- Todo o PORTB como saída ("0" em todos os bits do TRISB);
 - Pino RA0 como entrada ("1" no bit zero do TRISA);

- Os demais pinos de PORTA como saída ("0" nos demais bits do TRISA);
- Prescaler de 1:32 no TMRO e pull-ups desabilitados (veja bits do OPTION_REG);
- Todas as interrupções desligadas (veja bits do INTCON);
- Não se esqueça de que, para alterar esses registradores, é necessário mudar para o banco 1.

Para melhor entendimento, consulte o Capítulo 6 e Apêndice A.

INICIALIZANDO AS VARIÁVEIS

É muito importante que as variáveis também sejam inicializadas, mesmo que seus valores devam ser iguais a zero, pois nunca se sabe como a memória do PIC acordará. Inicializando as variáveis corretamente, muitos problemas podem ser evitados.

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
    INICIALIZAÇÃO DAS VARIÁVEIS  
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  
CLRF    PORTA          ;DESLIGA TODAS AS SAÍDAS DO PORT A  
CLRF    PORTB          ;DESLIGA TODAS AS SAÍDAS DO PORT B  
MOVlw   .10            ;  
MOVWF  CONTADOR       ;INICIA CONTADOR COM 10 (DECIMAL)
```

TRABALHANDO COM ROTINAS

Como em muitas outras linguagens de programação, no assembler do PIC existem dois tipos de rotinas: as rotinas de desvio e as rotinas de chamada, que também podem ser consideradas funções. As rotinas de desvio nada mais são que "pulos" no programa por meio da instrução GOTO, exatamente como na linguagem BASIC. Acontece que, no assembler do PIC, o número da linha é substituído por um nome (label). Já as rotinas de chamadas são acessadas através da instrução CALL. Essa instrução possibilita que o próximo ponto do programa (PC+1) seja guardado na pilha (stack), para que o sistema possa retornar a ele mais tarde, por meio da instrução RETURN (ou similar) que é utilizada para encerrar a rotina.

Para tornar mais fácil a visualização desses dois tipos de rotina, veja o esquema seguinte:

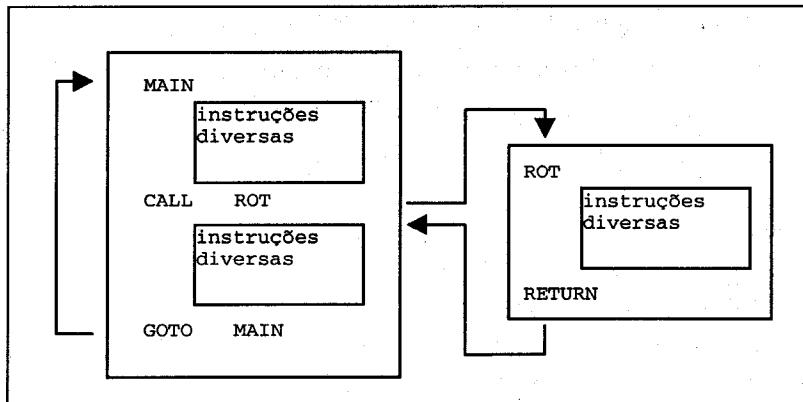


Figura 11.1 - Rotinas de Desvio e de Chamada.

ROTINAS DE DESVIO

As rotinas de desvio são utilizadas, geralmente, para deixar o programa mais estruturado e organizado. No entanto, a instrução GOTO é freqüentemente utilizada em qualquer programa para possibilitar "pulos" necessários à lógica do sistema. Na próxima seção (Tomando decisões e fazendo desvios), isso ficará muito mais claro.

A sintaxe dessa instrução é:

GOTO nome ;ONDE nome É A IDENTIFICAÇÃO DO LOCAL PARA ONDE SE DESEJA ;PULAR

Uma identificação de local nada mais é do que um nome localizado na primeira coluna de texto do arquivo-fonte. Para facilitar sua vida, recomendamos que os nomes sejam escritos logo na primeira coluna, e as instruções sejam indentadas à direita, por meio de uma tabulação, como pode ser observado no arquivo-modelo. Os nomes de rotinas não podem possuir espaços. Como é muito importante que esses nomes tenham algum sentido em relação à sua aplicação, recomendamos que o sublinhado (_) seja utilizado para ajudar na sua composição. Por exemplo:

GOTO BT_PRESS ;PULA PARA O LOCAL ONDE TRATARÁ O BOTÃO PRESSIONADO GOTO BT_LIB ;PULA PARA O LOCAL ONDE TRATARÁ O BOTÃO LIBERADO

Vejamos agora uma dica muito interessante para quando precisamos executar "pulos" muito curtos. Por exemplo, imagine uma situação em que é necessário pular somente duas ou três linhas, para cima ou para baixo. Poderíamos criar um nome para o local do pulo e utilizar o comando GOTO, como foi visto até agora, mas às vezes isso começa a se tornar um problema, pois você já não consegue mais inventar tantos nomes. Para estas, e para muitas outras situações que serão vistas adiante, é que o compilador possibilita o uso do \$. Ele é utilizado para representar a linha atual do programa. Veja, então, como fica o comando GOTO em conjunto com o \$:

GOTO \$+3	; PULA 3 LINHAS PARA BAIXO
GOTO \$-2	; PULA 2 LINHAS PARA CIMA

Não recomendamos essa técnica para "pulos" muito grandes, para não complicar o entendimento e a manutenção do programa.

ROTIAS DE CHAMADA

Já as rotinas de chamada são utilizadas quando uma tarefa deve ser repetida várias vezes e não se deseja reescrevê-la para não gastar memória de programa. Desta forma, a rotina pode ser usada como uma função, que é chamada de diversos pontos do programa sem gerar problemas. Por exemplo, se seu sistema precisa de um delay de atraso em diversos pontos do programa para funcionar corretamente, podemos criar uma rotina denominada **DELAY** e chamá-la quantas vezes forem necessárias, em quantos pontos precisarmos.

A instrução utilizada para chamar uma rotina desse tipo é a **CALL**.

A sintaxe dessa instrução é:

CALL nome ;ONDE nome É A IDENTIFICAÇÃO DA ROTINA
--

As observações feitas aos nomes empregados com a instrução **GOTO** também são válidas para a instrução **CALL**.

A grande vantagem de utilizar uma rotina de chamada é que o endereço seguinte ao ponto de chamada (linha abaixo da instrução **CALL**) é armazenado na pilha (Stack). Quando terminarmos a rotina com a instrução **RETURN**, o sistema voltará exatamente para o endereço armazenado na pilha. Simples, não é mesmo? No entanto, vale lembrá-lo de que outras rotinas podem ser chamadas de dentro da rotina atual, antes de darmos um **RETURN**. Isso irá gerar outros níveis de pilha com os respectivos endereços de retorno. Acontece que o PIC16F628A possui uma pilha de no máximo oito níveis. Se este limite for ultrapassado, o primeiro nível será sobreescrito, impossibilitando que o sistema retorne a todos os pontos de chamada, podendo gerar um grave erro no programa. Para os sistemas apresentados neste livro, e para muitos outros mais complexos, oito níveis de pilha são mais que o suficiente, mas é sempre bom ficar alerta, principalmente porque existem alguns PICs (mais antigos) que só possuem dois níveis de pilha.

Para retornar de uma rotina, devem ser utilizadas as instruções **RETURN** e **RETLW**.

As sintaxes dessas instruções são:

RETURN ;FINALIZA A ROTINA VOLTANDO AO ÚLTIMO ENDEREÇO DA PILHA.
RETLW k ;FINALIZA A ROTINA VOLTANDO AO ÚLTIMO ENDEREÇO DA PILHA COM ; O VALOR k (LITERAL) EM W.

Vale realçar que alguns modelos de PIC (12 bits) só possuem a instrução **RETLW**.

TOMANDO DECISÕES E FAZENDO DESVIOS

Esta seção irá apresentá-lo às instruções capazes de executar testes e tomar decisões dentro do PIC. Devido à filosofia RISC, o set de instruções é bem resumido, como já explicamos, e você verá que existem pouquíssimas instruções voltadas a essa finalidade. Aproveitaremos a ordem natural das coisas para mostrarmos também como alteramos diretamente bits e flags.

TESTANDO BITS E FLAGS (BTFSC E BTFSS)

As instruções empregadas para testar diretamente bits, que podem ser portas ou flags, são: **BTFSC** e **BTFSS**. Vamos utilizar o sistema de termos predefinidos para entendermos melhor suas funções:

BTFSC = Testa (T) o bit (B) do registrador (F) e pula (S) a próxima linha se a resposta for 0 (C).

BTFSS = Testa (T) o bit (B) do registrador (F) e pula (S) a próxima linha se a resposta for 1 (S).

As sintaxes dessas instruções são:

```
BTFSC f,b      ;ONDE f É O REGISTRADOR E b O BIT A SER TESTADO  
LINHA1        ;PASSA POR ESTA LINHA SE O BIT TESTADO FOR 1.  
LINHA2        ;PULA DIRETO PARA ESTA LINHA SE O BIT TESTADO FOR 0.  
  
BTFSS f,b      ;ONDE f É O REGISTRADOR E b O BIT A SER TESTADO  
LINHA1        ;PASSA POR ESTA LINHA SE O BIT TESTADO FOR 0.  
LINHA2        ;PULA DIRETO PARA ESTA LINHA SE O BIT TESTADO FOR 1.
```

Normalmente, as especificações do registrador e do bit podem ser substituídas por um **#DEFINE**, como no exemplo abaixo:

```
#DEFINE BOTAO PORTA,0    ;DEFINE O BOTÃO NO PINO RA0  
                  ; 0 -> LIBERADO  
                  ; 1 -> PRESSIONADO  
BTFSS  BOTAO          ;O BOTÃO ESTÁ PRESSIONADO?  
GOTO   BT_LIB           ;NÃO -> VAI TRATAR BOTÃO LIBERADO  
GOTO   BT_PRES          ;SIM -> VAI TRATAR BOTÃO PRESSIONADO
```

Os mesmos testes podem ser executados com flags para checar um certo estado do sistema.

MUDANDO BITS E FLAGS (BSF E BCF)

Agora que você já sabe como testar se um dado bit está em 1 (um) ou em 0 (zero), e também já sabe como executar uma ação específica conforme o resultado obtido, então está na hora de aprender como alterar o valor dos bits.

As instruções empregadas para a alteração de um bit são: **BSF** e **BCF**. Utilizando novamente a técnica de termos predefinidos teremos:

BSF = o bit (**B**) do registrador (**F**) deve ser setado (**S**) - igualado a 1

BCF = o bit (**B**) do registrador (**F**) deve ser zerado (**C**) - igualado a 0

As sintaxes dessas instruções são:

```
BSF    f,b      ;ONDE f É O REGISTRADOR E b O BIT QUE SERÁ SETADO.
```

```
BCF    f,b      ;ONDE f É O REGISTRADOR E b O BIT QUE SERÁ ZERADO.
```

Aqui também o registrador e o bit são comumente substituídos por **#DEFINES**.

Desta forma, podemos facilmente alterar o valor de um flag, ou então de um bit de configuração dentro dos SFRs.

TRABALHANDO COM AS PORTAS

Esta seção se dedicará às dicas de operação para a correta utilização das portas do PIC. As portas são provavelmente o recurso mais poderoso de um microcontrolador, pois é por meio delas que o PIC pode se comunicar com os demais componentes do sistema. Com essas portas podemos ler um botão, acender e apagar um LED, ativar um relé, controlar um motor, escrever em um display e muito mais. O importante, por enquanto, é você conhecer e dominar as maneiras de operar corretamente as portas para que seja possível concretizar suas idéias.

LENDÔ UMA PORTA

Uma porta pode ser lida de duas maneiras: como um conjunto, isto é, todos os seus pinos de uma só vez, ou individualmente. As instruções necessárias para ambas as maneiras já foram apresentadas.

Por exemplo, podemos ter ligado a todo o PORTB um barramento paralelo de dados, em que cada pino representa um bit e o PORTB representa o byte todo. Então, para sabermos qual valor está "escrito" no PORTB, basta lê-lo integralmente:

```
;CONSIDERAREMOS QUE A VARIÁVEL DADO JÁ FOI DEFINIDA ANTERIORMENTE
```

```
MOVF    PORTB,W      ;ESCREVE O VALOR DE PORTB EM W
MOVWF   DADO          ;ESCREVE O VALOR DE W EM DADO
                  ;DADO = VALOR DE PORTB
```

Neste exemplo, simplesmente copiamos o valor do registrador PORTB para o registrador DADO. Como o registrador PORTB expressa exatamente o estado de todos os pinos da porta B, então estamos lendo diretamente seus valores. E observe que isso serve tanto para os pinos configurados como entrada quanto para os configurados como saída. No caso de um pino do tipo entrada, quando lemos o seu

valor, estaremos obtendo o sinal imposto pelos circuitos externos ao PIC. No caso dos pinos do tipo saída, estaremos lendo o valor imposto pelo próprio PIC.

Por outro lado, na maior parte das vezes, uma entrada é associada a somente um pino, e não ao PORT inteiro. Neste caso, é muito mais fácil trabalharmos diretamente com o bit em questão. O exemplo dado anteriormente para a leitura de um botão, com a utilização das instruções BTFSS e/ou BTFSC, é o melhor exemplo disso.

ESCREVENDO EM UMA PORTA

A escrita nas portas é totalmente similar ao processo de leitura, com algumas ressalvas. Durante o processo de escrita em um pino, não estamos afetando diretamente o estado dele, mas sim um "latch" intermediário. Vamos explicar isso de uma forma mais simplificada. A cada pino das portas é associada uma chave interna (latch) que é alterada toda vez que escrevemos em um desses pinos. Se o pino estiver configurado como saída, então seu estado também é afetado, mas se ele for uma entrada, então só o latch é afetado. Devido a isso, é possível escrevermos um valor em um pino que é entrada, e quando o alteramos para saída, seu estado será garantido. Vejamos um exemplo:

```
BANK1          ;MUDA PARA BANCO 1
BSF      TRISB,0   ;TRANSFORMA RB0 EM ENTRADA
BANK0          ;RETORNA PARA BANCO 0

; AQUI PODE HAVER DIVERSOS COMANDOS

BSF      PORTB,0   ;ESCREVE 1 NO LATCH DO RB0. (QUE AINDA É ENTRADA)
BANK1          ;MUDA PARA BANCO 1
BCF      TRISB,0   ;TRANSFORMA RB0 EM SAÍDA. NESTE MOMENTO O PINO
                ;SERÁ INICIALIZADO EM 1 DEVIDO AO LATCH
BANK0          ;RETORNA PARA BANCO 0
```

Agora isso até pode parecer estranho, mas muitas vezes é totalmente necessário. Como já havia sido demonstrado, para afetar diretamente um pino da porta, utilizam-se as instruções BSF e BCF com o bit relacionado a esse pino.

Devido a essa estruturação interna, também nunca devemos escrever e ler na mesma porta em instruções seguidas. Deve ser garantido pelo menos um ciclo de máquina entre a escrita e a leitura, para assegurar as alterações executadas. Esse ciclo de máquina pode ser garantido pela instrução NOP, que consome um ciclo sem fazer absolutamente nada.

```
MOVLW .10        ;ESCREVE 10 EM W
MOVWF PORTB       ;TRANSFERE O VALOR DE W (10) PARA O PORT B
NOP              ;PERDE UM CICLO PARA ASSEGURAR A ESCRITA NO PORT
MOVF  PORTB,W     ;LÊ O VALOR DO PORT B E COLOCA EM W
MOVWF DADO        ;TRANSFERE O VALOR DE W (PORTB) PARA DADO
```

Se não fosse executada uma instrução de leitura logo após a escrita, a instrução NOP não seria necessária.

EXEMPLO 1 - BOTÃO E LED

Até aqui tudo bem, mas já está na hora de você ver algum exemplo prático e funcional para tudo isso, não é mesmo? Grande parte das informações até agora apresentadas refere-se às padronizações e dicas de programação, mas e os programas propriamente ditos? Pois nossa experiência afirma que é muito importante ter um bom conhecimento dessas bases para tornar-se um programador versátil e eficiente, e com os dados assimilados até aqui já é possível construirmos nosso primeiro programa. Obviamente será algo com uma lógica muito simples, para utilizarmos somente os comandos já vistos. O importante é que você analise e entenda a aplicação desses comandos, a formatação do arquivo, os comentários, enfim, o projeto como um todo, e não só a simplicidade das suas operações.

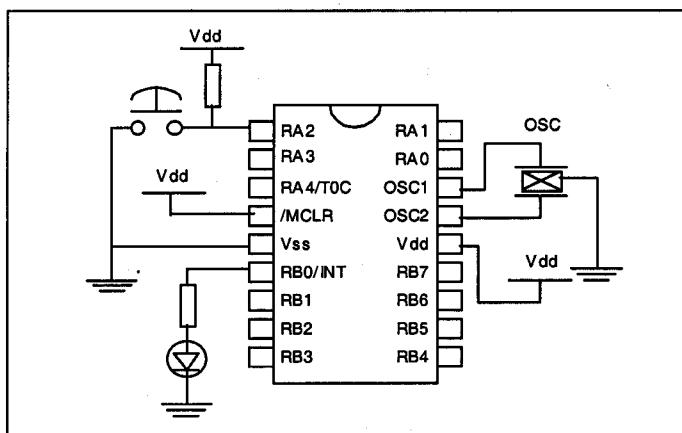


Figura 11.2 - Esquema do Exemplo 1.

Para começar, pensaremos em um sistema que possua somente um botão e um LED. O LED será utilizado para representar o estado do botão, isto é, aceso para o botão pressionado e apagado para o botão liberado. Para este primeiro exemplo, o esquema elétrico completo foi mostrado. Nos próximos, consideraremos o hardware proposto no Apêndice G.

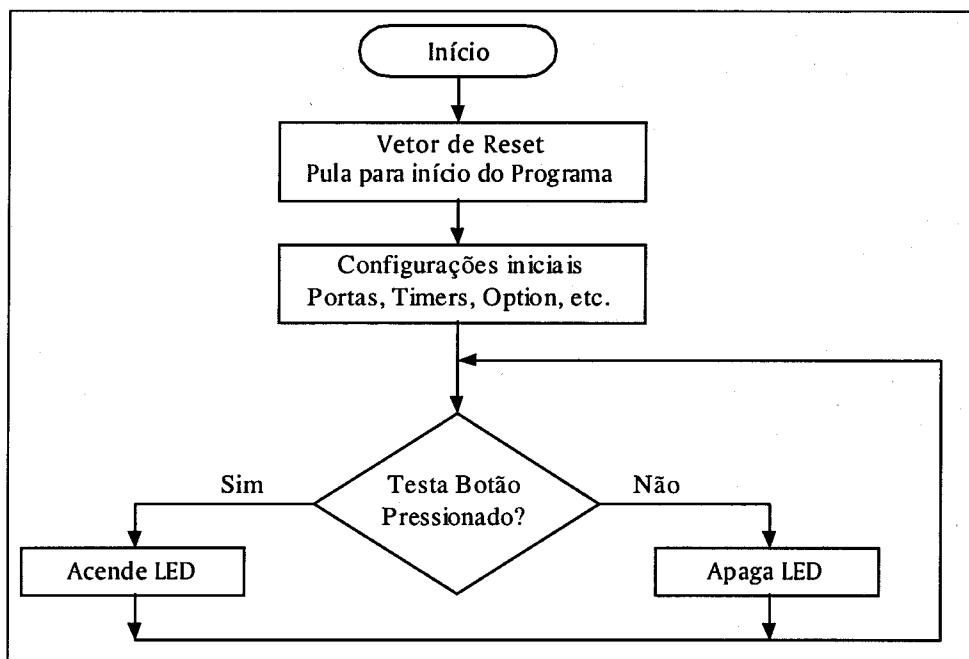


Figura 11.3 - Fluxograma do Exemplo 1.

```
;*                                BOTÃO E LED - EX1
;*                                DESBRAVANDO O PIC
;*      DESENVOLVIDO PELA MOSAICO ENGENHARIA E CONSULTORIA
;*      VERSÃO: 1.0                      DATA: 10/10/01
;*      DESCRIÇÃO DO ARQUIVO
;*-----*
;*      SISTEMA MUITO SIMPLES PARA REPRESENTAR O ESTADO DE
;*      UM BOTÃO ATRAVÉS DE UM LED.
;*-----*
;*      ARquivos DE DEFINIçõEs
;*-----*
#INCLUDE <P16F628.INC>          ;ARQUIVO PADRÃO MICROCHIP PARA 16F628
        _CONFIG _BODEN_ON & _CP_OFF & _PWRTE_ON & _WDT_OFF & _LVP_OFF &
        _MCLRE_ON & _XT_OSC

;*      PAGINAção DE MEMóRIA
;*-----*
;DEFINIção DE COMANDOS DE USUÁRIO PARA ALTERAÇÃO DA PÁGINA DE MEMóRIA

#define     BANK0    BCF STATUS,RP0 ;SETA BANK 0 DE MEMóRIA
#define     BANK1    BSF STATUS,RP0 ;SETA BANK 1 DE MAMóRIA
```



```
;*****  
;*          INICIO DO PROGRAMA  
;*****  
  
INICIO  
    CLRF    PORTA      ;LIMPA O PORTA  
    CLRF    PORTB      ;LIMPA O PORTB  
  
    BANK1      ;ALTERA PARA O BANCO 1  
    MOVLW  B'00000100' ;  
    MOVWF  TRISA      ;DEFINE RA2 COMO ENTRADA E DEMAIS  
                      ;COMO SAÍDAS  
    MOVLW  B'00000000' ;  
    MOVWF  TRISB      ;DEFINE TODO O PORTB COMO SAÍDA  
    MOVLW  B'10000000' ;  
    MOVWF  OPTION_REG ;PRESCALER 1:2 NO TMR0  
                      ;PULL-UPS DESABILITADOS  
                      ;AS DEMAIS CONFG. SÃO IRRELEVANTES  
    MOVLW  B'00000000' ;  
    MOVWF  INTCON     ;TODAS AS INTERRUPÇÕES DESLIGADAS  
    BANK0      ;RETORNA PARA O BANCO 0  
    MOVLW  B'00000111' ;  
    MOVWF  CMCON      ;DEFINE O MODO DE OPERAÇÃO DO COMPARADOR  
  
;*****  
;*          INICIALIZAÇÃO DAS VARIÁVEIS  
;*****  
  
;*****  
;*          ROTINA PRINCIPAL  
;*****  
  
MAIN  
    BTFSC  BOTAO      ;O BOTÃO ESTÁ PRESSIONADO?  
    GOTO   BOTAO_LIB  ;NÃO, ENTÃO TRATA BOTÃO LIBERADO  
    GOTO   BOTAO_PRES ;SIM, ENTÃO TRATA BOTÃO PRESSIONADO  
  
BOTAO_LIB  
    BCF    LED         ;APAGA O LED  
    GOTO   MAIN        ;RETORNA AO LOOP PRINCIPAL  
  
BOTAO_PRES  
    BSF    LED         ;ACENDE O LED  
    GOTO   MAIN        ;RETORNA AO LOOP PRINCIPAL  
  
;*****  
;*          FIM DO PROGRAMA  
;*****  
  
END          ;OBRIGATÓRIO
```

Agora, aproveite para fazer alguns exercícios propostos como alterações do exemplo apresentado:

1. Inverta a lógica do LED, fazendo ele acender quando o botão está solto e apagar quando o botão for pressionado.
2. Implemente mais um botão e um LED, totalmente independente do anterior.
3. Faça cada conjunto de botão e LED operar com uma lógica diferente.

FAZENDO OPERAÇÕES ARITMÉTICAS BÁSICAS

O próximo passo deve ser dado na direção dos cálculos. Basicamente, todos os programas implementados nos microcontroladores necessitam de algum tipo de conta para que sua lógica funcione corretamente. Neste capítulo, veremos que a matemática não é o ponto forte das instruções do PIC16F628A. Não diretamente, mas veremos também que ele possui todos os recursos necessários para que possamos implementar nossas próprias funções para cálculos muito mais avançados; basta um pouco de conhecimento e muita criatividade. Bem, mas vamos começar pelos cálculos mais simples possíveis, as contas básicas de adição e subtração.

SOMANDO (INCF, INCFSZ, ADDWF E ADDLW)

Para operações de adição, o assembler do PIC possui dois grupos de instruções, sendo um usado para adições unitárias e o outro para adições diversas. Dentro desses grupos, possuímos um total de quatro instruções que serão mostradas juntamente com suas descrições por meio da técnica de termos predefinidos.

- **INCF:** Incremento unitário (INC) do registrador (F).
- **INCFSZ:** Incremento (INC) do registrador (F) pulando a próxima linha (S) se o resultado for 0 (Z).
- **ADDWF:** Soma (ADD) o valor de work (W) ao registrador (F).
- **ADDLW:** Soma (ADD) um número (L) ao valor de work (W).

A sintaxe correta para cada uma dessas instruções pode ser vista em seguida:

INCF	f, d	;ONDE f É O REGISTRADOR E d O DESTINO ONDE SERÁ ;GUARDADO O RESULTADO DA CONTA (f + 1 -> d)
INCFSZ	f, d	;ONDE f É O REGISTRADOR E d O DESTINO ONDE SERÁ ;GUARDADO O RESULTADO DA CONTA (f + 1 -> d)
ADDWF	f, d	;ONDE f É O REGISTRADOR E d O DESTINO ONDE SERÁ ;GUARDADO O RESULTADO DA CONTA (f + w -> d)
ADDLW	k	;ONDE k É O NÚMERO QUE SERÁ SOMADO AO W ;O RESULTADO É MANTIDO EM W (w + k -> w)

A instrução INCF é utilizada normalmente para controlar contadores unitários dentro do sistema ou para funções específicas. Por exemplo, vamos olhar um código que chama uma rotina denominada BIP por oito vezes seguidas:

	CRLF	CONTA	; ZERA O CONTADOR
LOOP	CALL	BIP	; CHAMA A ROTINA BIP QUE EMITIRÁ UM SOM
	INCF	CONTA, F	; INCREMENTA O CONTADOR (CONTA = CONTA + 1)
	BTFS	CONTA, 3	; TESTA O BIT NÚMERO 3 DO CONTADOR.
			; QUANDO ESTE BIT FOR IGUAL A 1,
			; SIGNIFICA QUE CONTADOR = 8.
	GOTO	LOOP	; CONTADOR AINDA NÃO É 8, RETORNA PARA LOOP
FIM			; CONTADOR = 8, ACABOU O EXEMPLO

Observe que nós checamos o valor do contador por meio da análise de um de seus bits. Esta é uma maneira muito utilizada pelos programadores, mas também muito restrita, pois só permite comparações com os valores relacionados a cada um dos bits pela regra da potência de 2 (1, 2, 4, 8, 16, 32, 64 e 128). Na verdade, a comparação poderia ser feita em relação a qualquer valor, mas teríamos de utilizar instruções ainda não estudadas, por isso essa técnica foi empregada neste exemplo.

Já a instrução INCFSZ é bem mais poderosa, pois, além do incremento, ela faz também uma comparação para poder tomar uma decisão. Após cada incremento, ela verifica se o resultado é igual a zero. Bem, mas quando uma soma de um número positivo resulta em zero? Na matemática convencional, isso nunca acontecerá, mas na aritmética de 8 bits, como no PIC, este é um resultado muito normal. Isso porque quando trabalhamos com números de 8 bits, sempre que o limite de 255 é ultrapassado, o valor retorna para zero. Desta forma fica fácil criarmos um contador específico utilizando a instrução INCFSZ, bastando para isso iniciarmos o contador com a diferença entre 256 e o número de vezes que desejamos contar. Vejamos o exemplo anterior, só que agora com um looping de dez vezes:

	MOVLW	.246	
LOOP	MOVWF	CONTA	; INICIA O CONTADOR COM 246 (256 - 10)
	CALL	BIP	; CHAMA A ROTINA BIP QUE EMITIRÁ UM SOM
	INCFSZ	CONTA, F	; INCREMENTA O CONTADOR (CONTA = CONTA + 1)
			; RESULTADO = 0 (ESTOROU?)
	GOTO	LOOP	; NÃO, RETORNA PARA LOOP POIS NÃO PASSOU 10 VEZES
FIM			; SIM, ACABOU O EXEMPLO POIS JÁ PASSOU 10 VEZES

Veja agora um exemplo para as instruções ADD, que são utilizadas para somatórios diversos.

SOMA1	MOVLW	.10	
	MOVWF	NUM_1	; INICIA NUM_1 COM 10
	MOVLW	.20	
	MOVWF	NUM_2	; INICIA NUM_2 COM 20
	CLRF	RESULTADO	; INICIA RESULTADO COM 0
			; RESPOSTA = NUM_1 + 5
	MOVF	NUM_1,W	; COLOCA O VALOR DE NUM_1 (10) EM W
	ADDLW	.5	; SOMA 5 AO W (10+5=15)
	MOVWF	RESULTADO	; COLOCA A RESPOSTA EM RESULTADO = 15
SOMA2			
	MOVF	NUM_1,W	; COLOCA O VALOR DE NUM_1 (10) EM W
	ADDWF	NUM_2,W	; SOMA O VALOR DE W (10) A NUM_2 (20)
			; GUARD.EM W
	MOVWF	RESULTADO	; COLOCA A RESPOSTA EM RESULTADO = 30
FIM			; FIM DO EXEMPLO

Essas instruções afetam os três flags relacionados a ULA do registrador de STATUS: C, DC e Z. Podemos checar o estado desses registradores para tomar decisões conforme o resultado da conta efetuada. Por exemplo, imaginemos que precisamos somar três registradores quaisquer. Como cada registrador possui 8 bits, para que não haja erros, o resultado deveria ser guardado em um registrador de pelo menos 10 bits. Mas como fazer isso se as instruções só operam com 8 bits? O flag de carry (STATUS, C) irá nos auxiliar nessa tarefa. O exemplo seguinte mostrará a soma de quatro números quaisquer (NUM_1... NUM_3), guardando a resposta em dois bytes (BYTE_ALTO e BYTE_BAIXO):

```
; IMAGINEMOS QUE NUM_1, NUM_2, E NUM_3 ESTÃO COM VALORES QUAISQUER
CLRF    BYTE_BAIXO      ; INICIA BYTE_BAIXO COM 0
CLRF    BYTE_ALTO       ; INICIA BYTE_ALTO COM 0

MOVF    NUM_1,W         ; MOVE O VALOR DE NUM_1 PARA W
ADDWF   NUM_2,W         ; SOMA O VALOR DE W A NUM_2 GUARDANDO NO PRÓPRIO W
BTFSC   STATUS,C        ; A CONTA RESULTOU EM UM ESTOURO?
INCFL   BYTE_ALTO,F     ; SIM, INCREMENTA O BYTE_ALTO
                  ; NÃO

MOVWF   BYTE_BAIXO      ; COLOCA A RESPOSTA DA CONTA (W) EM BYTE_BAIXO
ADDWF   NUM_3,W         ; SOMA O RESULTADO ANTERIOR (W) A NUM_3 GUARDANDO
                  ; NO PRÓPRIO W
BTFSC   STATUS,C        ; A CONTA RESULTOU EM UM ESTOURO?
INCFL   BYTE_ALTO,F     ; SIM, INCREMENTA O BYTE_ALTO
                  ; NÃO

MOVWF   BYTE_BAIXO      ; COLOCA A RESPOSTA DA CONTA EM BYTE_BAIXO
FIM
```

SUBTRAINDO (DECFSZ, SUBWF E SUBLW)

As instruções para subtração seguem os mesmos princípios das vistas em relação à adição. Vejamos então quais são elas:

- **DECFSZ**: Decremento unitário (DEC) do registrador (F).
- **DECFSZ**: Decremento (DEC) do reg. (F) pulando a próxima linha (S) se o resultado for 0 (Z)
- **SUBWF**: Subtrai (SUB) o valor de work (W) do registrador (F).
- **SUBLW**: Subtrai (SUB) de um número (L) o valor de work (W).

A sintaxe correta para cada uma dessas instruções pode ser vista em seguida:

DECFSZ	f,d	;ONDE f É O REGISTRADOR E d O DESTINO ONDE SERÁ ;GUARDADO O RESULTADO DA CONTA (f - 1 -> d)
SUBWF	f,d	;ONDE f É O REGISTRADOR E d O DESTINO ONDE SERÁ ;GUARDADO O RESULTADO DA CONTA (f - w -> d)
SUBLW	k	;ONDE k É O NÚMERO DE ONDE SERÁ SUBTRAÍDO O VALOR DE W ;O RESULTADO É MANTIDO EM W (k - w -> w)

Analogamente aos exemplos dados na seção relativa à adição, vejamos a aplicação da instrução DECFSZ para rodar a função BIP por dez vezes:

```

MOVlw .10
MOVwf CONTA      ;INICIA O CONTADOR COM 10
LOOP
  CALL BIP        ;CHAMA A ROTINA BIP QUE EMITIRÁ UM SOM
  DECfsz CONTA,f ;DECREMENTA O CONTADOR (CONTA = CONTA - 1)
                  ;RESULTADO = 0 (ACABOU?)
  GOTO LOOP      ;NÃO, RETORNA PARA LOOP POIS NÃO PASSOU 10 VEZES
FIM               ;SIM, ACABOU O EXEMPLO

```

Observe que para esse tipo de contador a utilização da instrução DECFSZ no lugar da INCFSZ torna o programa mais fácil de entender.

As instruções SUB são utilizadas para subtrações diversas:

```

MOVlw .10
MOVwf NUM_1       ;INICIA NUM_1 COM 10
MOVLw .20
MOVwf NUM_2       ;INICIA NUM_2 COM 20
CLRF RESULTADO   ;INICIA RESULTADO COM 0
;RESPOSTA = 30 - NUM_1
SUB1
  MOVF NUM_1,W    ;COLOCA O VALOR DE NUM_1 (10) EM W
  SUBlw .30        ;SUBTRAI W DE 30 (30 - 10 = 20)
  MOVwf RESULTADO ;COLOCA A RESPOSTA EM RESULTADO = 20
;RESPOSTA = NUM_2 - NUM_1
SUB2
  MOVF NUM_1,W    ;COLOCA O VALOR DE NUM_1 (10) EM W
  SUBwf NUM_2,W    ;SUBTRAI O VALOR DE W (10) DE NUM_2 (20) => W
  MOVwf RESULTADO ;COLOCA A RESPOSTA EM RESULTADO = 10
;FIM DO EXEMPLO
FIM

```

Aqui também podemos observar que, ao contrário das instruções de adição, a ordem dos fatores para a subtração afeta diretamente o resultado. Em uma soma, o resultado pode ser zero ou positivo, mas na subtração ele pode ser zero, positivo ou negativo. Por meio da análise do flag de carry podemos concluir qual o resultado correto da subtração:

- **Negativo:** Sempre que o resultado da subtração for negativo, o carry será zero (0). Neste caso, o valor da resposta não será diretamente o número negativo, e sim sua diferença para 256.
- **Positivo:** Sempre que o resultado for positivo, o carry será um (1).
- **Zero:** Sempre que o resultado for zero, o carry será um (1). Neste caso, o flag de zero também será um (1).

Vejamos então um exemplo para a subtração de dois números quaisquer: NUM_1 e NUM_2. O módulo do resultado será colocado em RESP e o flag NEG será setado sempre que o resultado for negativo:

```

;IMAGINEMOS QUE NUM_1 E NUM_2 ESTÃO COM VALORES QUAISQUER
SUB1
  CLRF RESP        ;LIMPA O REGISTRADOR ONDE SERÁ COLOCADA A RESPOSTA
  ;NUM_2 - NUM_1 = RESP
  MOVF NUM_1,W    ;MOVE O VALOR DE NUM_1 PARA W
  SUBwf NUM_2,W    ;SUBTRAI O VALOR DE W (NUM_1) DE
                  ;NUM_2 GUARDANDO EM W

```

BTFS S	STATUS, C	; TESTA CARRY. RESULTADO NEGATIVO?
GOTO	TRATA_NE	; SIM, PULA PARA TRATAR RESULTADO NEGATIVO
		; NÃO, RESULTADO POSITIVO OU ZERO
MOVWF	RESP	; COLOCA O RESULTADO DIRETAMENTE EM RESP
BCF	NEG	; LIMPA FLAG DE NÚMERO NEGATIVO
GOTO	FIM	; FINALIZA
TRATA_NEG		; COMO O RESULTADO FOI NEGATIVO, ENTÃO RESP = 256 - W
		; COMO O NÚMERO MÁXIMO PARA 8 BITS É 255, ENTÃO
		256 -> 0
SUBLW	.0	; 0 - W -> W
MOVWF	RESP	; COLOCA O RESULTADO EM RESP
BSF	NEG	; SETA O FLAG DE NÚMERO NEGATIVO
FIM		; FIM DO EXEMPLO, COM O MÓDULO DO RESULTADO EM RESP

As instruções SUB afetam também os flags DC e Z do registrador STATUS.

AS COMPARAÇÕES MAIOR QUE, MENOR QUE E IGUAL

Como acabamos de ver, as instruções SUB afetam diretamente o flag de carry, e por meio dele podemos identificar se o resultado foi negativo, positivo ou zero. Desta forma podemos identificar também se um número é maior, menor ou igual a outro. Para isso, utilizaremos as instruções SUBWF conforme o exemplo:

```
; IMAGINEMOS QUE NUM_1 E NUM_2 SÃO DOIS REGISTRADORES COM VALORES QUAISQUER

COMPARA1
    MOVF   NUM_1,W      ; MOVE O VALOR DE NUM_1 PARA W
    SUBWF  NUM_2,W      ; SUBTRAII O VALOR DE W (NUM_1) DE
                        ; NUM_2 GUARD. EM W
    BTFSS  STATUS,C     ; TESTA CARRY. RESULTADO NEGATIVO?
    GOTO   RESP1        ; SIM, ENTÃO NUM_2 < NUM_1 (NUM_1 > NUM_2)
    GOTO   RESP2        ; NÃO, ENTÃO NUM_2 >= NUM_1 (NUM_1 <= NUM_2)
FIM

; IMAGINEMOS AGORA QUE NUM_1 É UM NÚMERO QUALQUER (LITERAL)
; E NUM_2 É UM REGISTRADOR COM VALOR QUALQUER
```

```
COMPARA2
    MOVLW  NUM_1          ; MOVE O NÚMERO NUM_1 PARA W
    SUBWF  NUM_2,W        ; SUBTRAII O VALOR DE W (NUM_1) DE NUM_2
                        ; GUARD. EM W
    BTFSS  STATUS,C      ; TESTA CARRY. RESULTADO NEGATIVO?
    GOTO   RESP1        ; SIM, ENTÃO NUM_2 < NUM_1 (NUM_1 > NUM_2)
    GOTO   RESP2        ; NÃO, ENTÃO NUM_2 >= NUM_1 (NUM_1 <= NUM_2)
FIM
```

MULTIPLICANDO (RLF)

Não existem instruções específicas de multiplicação dentro da linguagem do PIC16F628A. No entanto, elas podem ser "construídas" utilizando as instruções já vistas. Além disso, uma nova instrução, a RLF, ajudará muito na hora de multiplicar um número. Vamos dar uma olhada na descrição desta instrução:

- **RLF:** Rotaciona (R) um bit para a esquerda (L) do registrador (F).

A sintaxe dessa instrução é a seguinte:

RLF	<code>f,d ;ONDE f É O REGISTRADOR E d O DESTINO ONDE SERÁ ;GUARDADO O RESULTADO DA ROTAÇÃO</code>
-----	---

Puxa, mas o que a rotação tem a ver com a multiplicação? O que esta instrução faz realmente? Na verdade, ela simplesmente desloca todos os bits do registrador para a esquerda, colocando o valor de carry no bit zero e depois "jogando" o valor do bit 7 em carry. Veja a próxima figura para uma melhor compreensão.

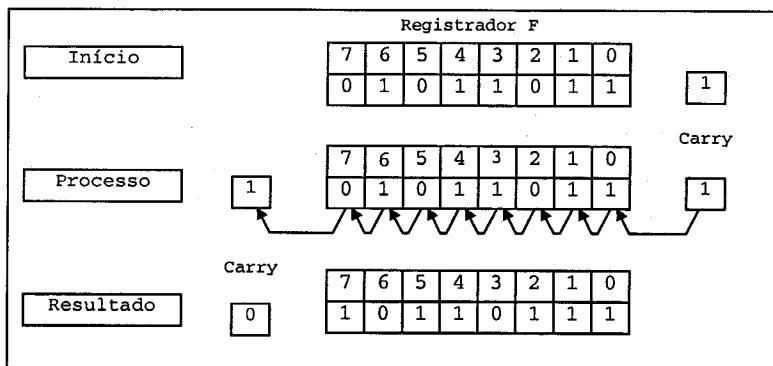


Figura 11.4 - Rotação de 1 Bit para a Esquerda.

Acontece que, matematicamente falando, se garantirmos que o bit que entra na posição menos significativa (bit0) seja zero, então cada vez que rotacionamos o byte para a esquerda, estamos multiplicando seu valor por 2. Desta forma, se o byte for rotacionado três vezes, seu valor será multiplicado por 8 ($2 \times 2 \times 2$). Esta é uma forma simples e rápida de multiplicação, mas só podemos efetuar contas com potências de 2. Neste caso, assim como nas contas de adição, o resultado da multiplicação pode precisar de mais de 8 bits.

O próximo exemplo mostra um registrador sendo multiplicado por oito:

<code>;IMAGINEMOS QUE NUM_1 É UM REGISTRADOR COM VALOR QUALQUER</code>		
CLRF	BYTE_BAIXO	<code>;LIMPA O REGISTRADOR BYTE_BAIXO</code>
CLRF	BYTE_ALTO	<code>;LIMPA O REGISTRADOR BYTE_ALTO</code>
MULT8		
BCF	STATUS,C	<code>;LIMPA O CARRY</code>
RLF	NUM_1,F	<code>;MULTIPLICA POR 2 (NUM_1 = NUM_1 X 2)</code>
RLF	BYTE_ALTO,F	<code>;ROTAÇÃO BYTE_ALTO PARA PEGAR O BYTE "PERDIDO" ;NA CONTA</code>
RLF	NUM_1,F	<code>;MULTIPLICA POR 2 (NUM_1 = NUM_1 X 4)</code>
RLF	BYTE_ALTO,F	<code>;ROTAÇÃO BYTE_ALTO PARA PEGAR O BYTE "PERDIDO" ;NA CONTA</code>
RLF	NUM_1,F	<code>;MULTIPLICA POR 2 (NUM_1 = NUM_1 X 8)</code>
RLF	BYTE_ALTO,F	<code>;ROTAÇÃO BYTE_ALTO PARA PEGAR O BYTE "PERDIDO" ;NA CONTA</code>
MOVF	NUM_1,W	<code>;MOVE A PARTE BAIXA DA CONTA PARA W</code>

MOVWF	BYTE_BAIXO	; ACERTA O VALOR DE BYTE_BAIXO ; NESTE EXEMPLO O VALOR DE NUM_1 FOI ; ALTERADO
FIM		

Uma outra forma de efetuarmos multiplicações com fatores que podem ser diferentes das potências de 2 é por meio de somas sucessivas. Essa técnica, porém, é muito mais lenta, variando o tempo de multiplicação de acordo com o valor de um dos números (no exemplo = NUM_1):

; IMAGINEMOS QUE NUM_1 E NUM_2 SÃO REGISTRADORES COM VALORES QUAISQUER		
--	--	--

CLRF	BYTE_BAIXO	; LIMPA O REGISTRADOR BYTE_BAIXO
CLRF	BYTE_ALTO	; LIMPA O REGISTRADOR BYTE_ALTO
MULT		; NUM_1 X NUM_2
MOVF	NUM_2,W	; MOVE O VALOR DE NUM_2 PARA W
ADDWF	BYTE_BAIXO,F	; SOMA NUM_2 AO VALOR JÁ EXISTENTE
BTFSC	STATUS,C	; HOUVE ESTOURO?
INCF	BYTE_ALTO,F	; SIM, INCREMENTA BYTE_ALTO ; NÃO
DECFSZ	NUM_1,F	; DECREMENTA NUM_1. ACABOU?
GOTO	MULT	; NÃO, CONTINUA SOMANDO ; SIM, SOMA SUCESSIVA TERMINADA
FIM		; NESTE EXEMPLO O VALOR DE NUM_1 FOI ALTERADO

Neste exemplo, a variável NUM_1 não pode ser zero, pois neste caso a primeira vez que a instrução DECFSZ fosse aplicada, NUM_1 equivaleria a 255, e a multiplicação ficaria totalmente errada. Para corrigir este problema, antes de entrarmos na soma sucessiva, o valor de NUM_1 deve ser testado. No caso de zero, a multiplicação nem é feita.

DIVIDINDO (RRF)

A divisão é a pior de todas as contas básicas que podemos fazer dentro de um microcontrolador (e fora dele também), principalmente porque aqui só podemos trabalhar com números inteiros. Analogamente ao que foi visto na seção de multiplicação, podemos utilizar a instrução RRF para conseguirmos um efeito contrário à instrução RLF.

Vamos dar uma olhada na descrição dessa instrução:

- **RRF:** Rotaciona (R) um bit para a direita (R) do registrador (F).

A sintaxe dessa instrução é a seguinte:

RRF	f,d	; EM QUE f É O REGISTRADOR E d O DESTINO ONDE SERÁ ; GUARDADO O RESULTADO DA ROTAÇÃO
-----	-----	---

O esquema mostra a rotação com a utilização do carry:

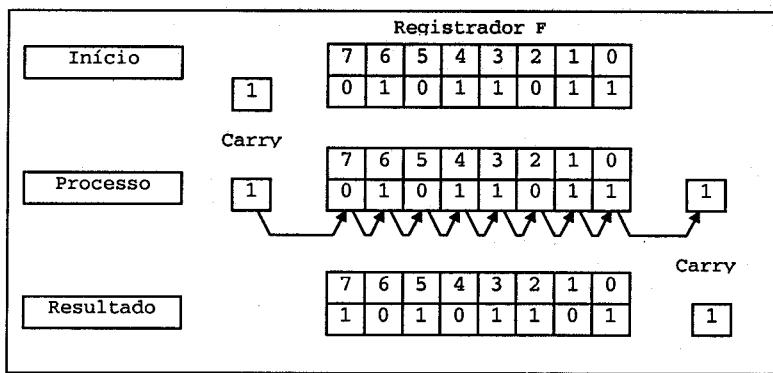


Figura 11.5 - Rotação de 1 Bit para a Direita.

No caso da divisão, o resultado (parte inteira) caberá em um único registrador de 8 bits, mas o resto da divisão poderá ser perdido por meio do carry.

```
; IMAGINEMOS QUE NUM_1 É UM REGISTRADOR COM VALOR QUALQUER
DIV4
    BCF    STATUS,C      ; LIMPA O CARRY
    RRF    NUM_1,F       ; DIVIDE POR 2 (NUM_1 = NUM_1 / 2)
    RRF    NUM_1,F       ; DIVIDE POR 2 (NUM_1 = NUM_1 / 4)
FIM
    ; NESTE EXEMPLO O VALOR DE NUM_1 FOI ALTERADO
    ; E O RESTO DA DIVISÃO FOI PERDIDO
```

Já a divisão por números que não sejam potências de 2 é extremamente mais complexa que a multiplicação, pois não existe uma maneira análoga ao processo de somas sucessivas, sendo o algoritmo necessário para tal operação, detalhado demais para o escopo deste livro.

EXEMPLO 2 - CONTADOR SIMPLIFICADO

Neste exemplo, utilizaremos as funções de incremento e decremento para implementar um contador bem simplificado. Duas constantes (MIN e MAX) são utilizadas para especificar os valores limites para a contagem. A técnica de comparação, vista na seção anterior, é aplicada para checar esses limites. Em relação ao EXEMPLO 1, melhoramos a checagem do botão pela implementação de um filtro muito simples. Para considerarmos que o botão foi realmente pressionado, o pino relativo ao botão é checado várias vezes seguidas. Quanto mais vezes checamos, mais tempo o botão tem de ser realmente pressionado para efetuar a função. A constante T_FILTER ajusta a quantidade de vezes da checagem. Ao atingir o valor máximo, o contador passa a decrementar o valor até que o limite mínimo seja novamente alcançado, voltando ao início do processo. O valor do contador é mostrado de forma binária no PORTB.

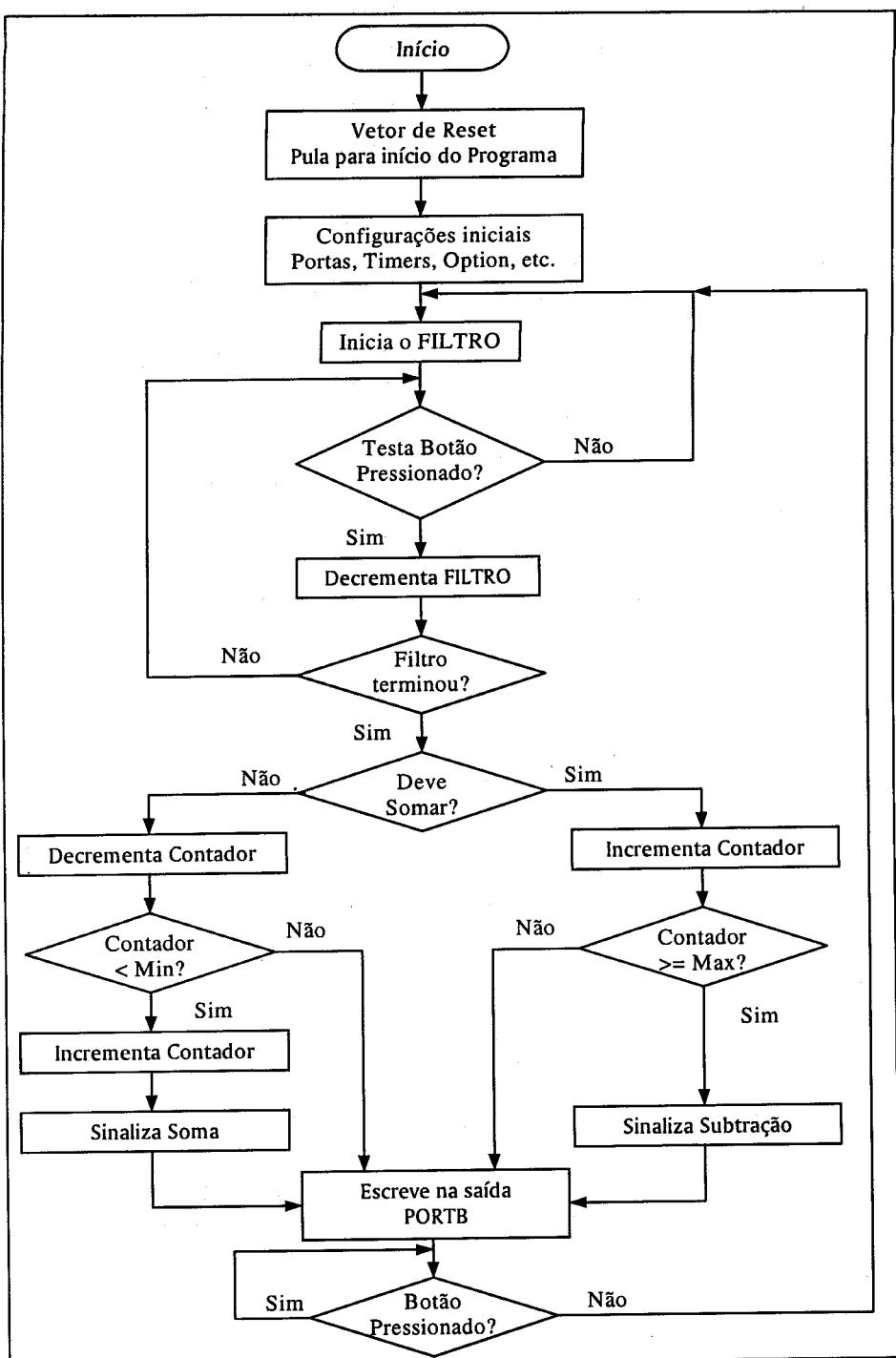


Figura 11.6 - Fluxograma do Exemplo 2.

```

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                               CONTADOR SIMPLIFICADO - EX2
;*                               DESBRAVANDO O PIC
;*           DESENVOLVIDO PELA MOSAICO ENGENHARIA E CONSULTORIA
;*           VERSÃO: 1.0          DATA: 17/06/03
;*           DESCRIÇÃO DO ARQUIVO
;*-
;*           SISTEMA MUITO SIMPLES PARA INCREMENTAR ATÉ UM DETERMINADO
;*           VALOR (MAX) DE DEPOIS DECREMENTAR ATÉ OUTRO (MIN).
;*
;*           -----
;*           ARQUIVOS DE DEFINIÇÕES
;*           -----
;* INCLUDE <P16F628A.INC>      ;ARQUIVO PADRÃO MICROCHIP PARA 16F628A
;* _CONFIG _BOREN_ON & _CP_OFF & _PWRTE_ON & _WDT_OFF & _LVP_OFF &
;* _MCLRE_ON & _XT_OSC

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           PAGINAÇÃO DE MEMÓRIA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;DEFINIÇÃO DE COMANDOS DE USUÁRIO PARA ALTERAÇÃO DA PÁGINA DE MEMÓRIA

#define      BANK0  BCF STATUS,RP0 ;SETA BANK 0 DE MEMÓRIA
#define      BANK1  BSF STATUS,RP0 ;SETA BANK 1 DE MAMÓRIA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           VARIÁVEIS
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS
; PELO SISTEMA

        CBLOCK 0x20          ;ENDEREÇO INICIAL DA MEMÓRIA DE
                           ;USUÁRIO
        CONTADOR          ;ARMAZENA O VALOR DA CONTAGEM
        FLAGS             ;ARMAZENA OS FLAGS DE CONTROLE
        FILTRO            ;FILTRAGEM PARA O BOTÃO
        ENDC              ;FIM DO BLOCO DE MEMÓRIA
        *
;*           FLAGS INTERNOS
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS FLAGS UTILIZADOS PELO SISTEMA

#define      SENTIDO FLAGS,0    ;FLAG DE SENTIDO
                           ; 0 -> SOMANDO
                           ; 1 -> SUBTRAINDO

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           CONSTANTES
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODAS AS CONSTANTES UTILIZADAS PELO SISTEMA

MIN      EQU    .10      ;VALOR MÍNIMO PARA O CONTADOR
MAX      EQU    .30      ;VALOR MÁXIMO PARA O CONTADOR

```

```

T_FILTRO EQU .230 ;FILTRO PARA BOTÃO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* ENTRADAS
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO ENTRADA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)

#define BOTAO PORTA,2 ;PORTA DO BOTÃO
; 0 -> PRESSIONADO
; 1 -> LIBERADO

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* SAÍDAS
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO SAÍDA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* VETOR DE RESET
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

ORG 0x00 ;ENDEREÇO INICIAL DE PROCESSAMENTO
GOTO INICIO

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* INÍCIO DA INTERRUPÇÃO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; AS INTERRUPÇÕES NÃO SERÃO UTILIZADAS, POR ISSO PODEMOS SUBSTITUIR
; TODO O SISTEMA EXISTENTE NO ARQUIVO MODELO PELO APRESENTADO ABAIXO
; ESTE SISTEMA NÃO É OBRIGATÓRIO, MAS PODE EVITAR PROBLEMAS FUTUROS

ORG 0x04 ;ENDEREÇO INICIAL DA INTERRUPÇÃO
RETFIE ;RETORNA DA INTERRUPÇÃO

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* INICIO DO PROGRAMA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

INICIO
BANK1 ;ALTERA PARA O BANCO 1
MOVLW B'00000100'
MOVWF TRISA ;DEFINE RA2 COMO ENTRADA E DEMAIS
;COMO SAÍDAS
MOVLW B'00000000'
MOVWF TRISB ;DEFINE TODO O PORTB COMO SAÍDA
MOVLW B'10000000'
MOVWF OPTION_REG ;PRESCALER 1:2 NO TMRO
;PULL-UPS DESABILITADOS
;AS DEMAIS CONFIG. SÃO IRRELEVANTES
MOVLW B'00000000'
MOVWF INTCON ;TODAS AS INTERRUPÇÕES DESLIGADAS
BANK0 ;RETORNA PARA O BANCO 0
MOVLW B'00000111'
MOVWF CMCON ;DEFINE O MODO DE OPERAÇÃO DO COMPARADOR

```

```
;*****  
;*          INICIALIZAÇÃO DAS VARIÁVEIS  
;*****  
  
    CLRF  PORTA      ;LIMPA O PORTA  
    CLRF  PORTB      ;LIMPA O PORTB  
    MOVLW  MIN         ;  
    MOVWF  CONTADOR   ;INICIA CONTADOR = V_INICIAL  
  
;*****  
;*          ROTINA PRINCIPAL  
;*****  
  
MAIN  
    MOVLW  T_FILTRO   ;INICIALIZA FILTRO = T_FILTRO  
    MOVWF  FILTRO     ;  
  
CHECA_BT  
    BTFSC  BOTAO      ;O BOTÃO ESTÁ PRESSIONADO?  
    GOTO   MAIN        ;NÃO, ENTÃO CONTINUA ESPERANDO  
                      ;SIM  
    DECFSZ FILTRO,F   ;DECREMENTA O FILTRO DO BOTÃO  
                      ;TERMINOU?  
    GOTO   CHECA_BT    ;NÃO, CONTINUA ESPERANDO  
                      ;SIM  
  
TRATA_BT  
    BTFSS  SENTIDO     ;DEVE SOMAR (SENTIDO=0)?  
    GOTO   SOMA        ;SIM  
                      ;NÃO  
SUBTRAI  
    DECF   CONTADOR,F  ;DECREMENTA O CONTADOR  
  
    MOVLW  MIN         ;MOVE O VALOR MÍNIMO PARA W  
    SUBWF  CONTADOR,W  ;SUBTRAI O VALOR DE W (MIN) DE CONTADOR  
    BTFSC  STATUS,C    ;TESTA CARRY. RESULTADO NEGATIVO?  
    GOTO   ATUALIZA    ;NÃO, ENTÃO CONTA >= MIN  
                      ;SIM, ENTÃO CONTA < MIN  
  
    INCF  CONTADOR,F  ;INCREMENTA CONTADOR NOVAMENTE  
                      ;POIS PASSOU DO LIMITE  
    BCF    SENTIDO      ;MUDA SENTIDO PARA SOMA  
    GOTO   MAIN        ;VOLTA AO LOOP PRINCIPAL  
  
SOMA  
    INCF  CONTADOR,F  ;INCREMENTA O CONTADOR  
  
    MOVLW  MAX         ;MOVE O VALOR MÁXIMO PARA W  
    SUBWF  CONTADOR,W  ;SUBTRAI O VALOR DE W (MIN) DE CONTADOR  
    BTFSS  STATUS,C    ;TESTA CARRY. RESULTADO NEGATIVO?  
    GOTO   ATUALIZA    ;SIM, ENTÃO CONTA < MAX  
                      ;NÃO, ENTÃO CONTA >= MAX  
  
    BSF    SENTIDO      ;MUDA SENTIDO PARA SUBTRAÇÃO  
    GOTO   MAIN        ;VOLTA AO LOOP PRINCIPAL  
  
ATUALIZA  
    MOVF   CONTADOR,W  ;COLOCA CONTADOR EM W  
    MOVWF  PORTB       ;ATUALIZA O PORTB PARA  
                      ;VISUALIZARMOS O VALOR DE CONTADOR
```

```

BTFSS  BOTAO      ; O BOTÃO CONTINUA PRESSIONADO?
GOTO   $-1        ; SIM, ENTÃO ESPERA LIBERAÇÃO PARA
                  ; QUE O CONTADOR NÃO DISPARE
GOTO   MAIN       ; NÃO, VOLTA AO LOOP PRINCIPAL

; * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
END          ; FIM DO PROGRAMA
              ; OBRIGATÓRIO
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

Treine agora com os exercícios propostos:

1. Inicie o contador corretamente, escrevendo o valor do contador na saída antes mesmo de pressionar o botão a primeira vez.
2. Cheque se o contador está mostrando os valores extremos (Máx. e Min.). Se não estiver, acerte esta condição. Altere esse limites para checar novamente.
3. Implemente um contador com dois botões, um para incrementar e outro para decrementar.

TRABALHANDO DIRETAMENTE COM BYTES

Muitas vezes, pelos mais diversos motivos, precisamos operar diretamente com os valores dos bytes guardados nos registradores, seja para efetuar uma conta, uma máscara ou filtro ou ainda uma comparação.

Assim como em qualquer outra linguagem, no assembler do PIC possuímos instruções para as operações lógicas que trabalham com bytes, operando bit a bit. Cada uma delas, e também suas aplicações, serão vistas separadamente nas seções seguintes.

AND (ANDWF E ANDLW)

A operação AND é um "E lógico" entre dois bytes. Cada bit do primeiro byte é comparado com o seu bit relativo no segundo byte, resultando em um terceiro valor, conforme a tabela seguinte, que expressa uma multiplicação:

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

Observando os resultados mostrados na tabela, podemos compreender o porquê do nome AND (E). É que o resultado só é 1 quando o primeiro E o segundo bit são 1. Vejamos então o resultado da operação AND em dois bytes quaisquer:

Byte 1	0	1	1	0	0	1	1	1
Byte 2	0	0	1	0	1	1	1	0
AND	0	0	1	0	0	1	1	0

As instruções responsáveis por esta operação dentro do assembler do PIC são:

- ANDLW: Operação "E" (AND) entre um número (L) e o valor de work (W).
- ANDWF: Operação "E" (AND) entre o valor de work (W) e o registrador (F).

A sintaxe dessas instruções é a seguinte:

ANDLW k ;ONDE k É UM DOS NÚMEROS E W É O OUTRO ;O RESULTADO É MANTIDO EM W
ANDWF f,d ;ONDE f É O REGISTRADOR E d O DESTINO ONDE SERÁ ;GUARDADO O RESULTADO DA OPERAÇÃO

Utilizamos a operação AND sempre que desejamos "mascarar" um valor, forçando determinados bits para zero. Por exemplo: imagine que você está trabalhando com um determinado registrador, mas antes de efetuar o processamento necessário, deve ser garantido que o valor dele nunca seja maior que 15. Uma maneira muito eficiente de conseguir isso é por meio do mascaramento do registrador, isto é, forçaremos os 4 bits mais significativos para zero. Veja o código para entender corretamente o processo:

;IMAGINEMOS QUE NUM_1 É UM REGISTRADOR COM VALOR QUALQUER
MOVLW B'00001111' ;COLOCAMOS A "MÁSCARA" EM W. NESTE CASO, ;ZERAREMOS OS 4 BITS MAIS SIGNIFICATIVOS (DIREITA)
ANDWF NUM_1,F ;APLICA A MÁSCARA AO NUM_1 ;A PARTIR DE AGORA NUM_1 ESTÁ ENTRE 0 E 15

A regra de aplicação da máscara para a operação AND é a seguinte: os bits em 1 (um) não afetam o resultado, já os bits em 0 (zero) forçam o resultado para zero também (lembre-se da relação de multiplicação).

OR (IORWF E IORLW)

A operação OR é um "OU lógico" entre dois bytes. Exatamente como na operação AND, cada bit do primeiro byte é comparado com o seu bit relativo no segundo byte, resultando em um terceiro valor. Neste caso, entretanto, a tabela-verdade é diferente, expressando os resultados semelhantes a uma soma:

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Observando novamente os resultados mostrados na tabela, podemos compreender também o motivo do nome OR (OU). Desta vez, o resultado é 1 quando o primeiro é OU o segundo bit é 1. Vejamos, então, o resultado da operação OR nos mesmos bytes mostrados na operação AND:

Byte 1	0	1	1	0	0	1	1	1
Byte 2	0	0	1	0	1	1	1	0
OR	0	1	1	0	1	1	1	1

As instruções responsáveis por esta operação dentro do assembler do PIC são:

- **IORLW**: Operação OU (IOR) entre um número (L) e o valor de work (W).
- **IORWF**: Operação OU (IOR) entre o valor de work (W) e o registrador (F).

A sintaxe dessas instruções é a seguinte:

```
IORLW k      ;ONDE k É UM DOS NÚMEROS E W É O OUTRO
              ;O RESULTADO É MANTIDO EM W
IORWF f,d    ;ONDE f É O REGISTRADOR E d O DESTINO ONDE SERÁ
              ;GUARDADO O RESULTADO DA OPERAÇÃO
```

Assim como a operação AND, esta operação também é utilizada para "mascarar" um valor, só que com efeito contrário, pois ela força os bits para 1 (um). A regra de aplicação da máscara para a operação OR é a seguinte: os bits em 0 (zero) não afetam o resultado, já os bits em 1 (um) forçam o resultado para 1 (um) também.

XOR (XORWF E XORLW)

Existe uma terceira operação lógica que é o XOR, ou "OU exclusivo". Esta operação é muito semelhante ao OR já analisado, com a diferença de que dois bits em 1 resultam em 0 (zero). Veja a tabela-verdade seguinte:

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

A lógica simplificada dessa operação é que dois bits iguais resultam em 0 (zero), enquanto bits diferentes resultam em 1 (um). Vejamos então o resultado da operação XOR nos mesmos bytes mostrados nas operações anteriores:

Byte 1	0	1	1	0	0	1	1	1
Byte 2	0	0	1	0	1	1	1	0
XOR	0	1	0	0	1	0	0	1

As instruções responsáveis por esta operação dentro do assembler do PIC são:

- **XORLW**: Operação OU exclusiva (XOR) entre um número (L) e o valor de work (W).
- **XORWF**: Operação OU exclusiva (XOR) entre o valor de work (W) e o registrador (F).

A sintaxe dessas instruções é a seguinte:

XORLW k	;ONDE k É UM DOS NÚMEROS E W É O OUTRO
	;O RESULTADO É MANTIDO EM W
XORWF f,d	;ONDE f É O REGISTRADOR E d O DESTINO ONDE SERÁ
	;GUARDADO O RESULTADO DA OPERAÇÃO

Uma aplicação muito comum para a operação XOR é a comparação entre dois números. Como a comparação é feita bit a bit, e sempre que os bits forem iguais o resultado é zero; então, se os números forem iguais, o resultado final será zero também:

CRLF	CONTA	;ZERA O CONTADOR
LOOP		
CALL	BIP	;CHAMA A ROTINA BIP QUE EMITIRÁ UM SOM
INCF	CONTA,F	;INCREMENTA O CONTADOR (CONTA = CONTA + 1)
MOVLW	.5	;MOVE 5 PARA O WORK
XORWF	CONTA,W	;XOR ENTRE CONTA E 5 (COMPARAÇÃO)
BTFSZ	STATUS,Z	;O RESULTADO DA OPERAÇÃO FOI ZERO?
GOTO	LOOP	;NÃO, ENTÃO CONTADOR DIFERENTE DE 5.
		;CONTINUA
FIM		;SIM, CONTADOR = 5, ACABOU O EXEMPLO

Para testar se o valor de um registrador é igual a zero, existe uma maneira ainda mais simples e que utiliza menos instruções, sendo, portanto, mais rápida. Você se lembra de que, quando vimos a instrução **MOVF**, deixamos de explicar o porquê da possibilidade de movermos um registrador para ele mesmo? Pois aqui está o motivo. Sempre que um registrador com valor zero é movido, mesmo que seja para ele mesmo, o flag de zero é setado. Desta forma, fica muito simples descobrirmos se um registrador é zero ou não:

;IMAGINEMOS QUE NUM_1 É UM REGISTRADOR COM VALOR QUALQUER		
MOVF	NUM_1,F	;MOVE O VALOR DO REGISTRADOR PARA ELE MESMO
BTFSZ	STATUS,Z	;O RESULTADO DA OPERAÇÃO FOI ZERO?
GOTO	DIF	;NÃO, ENTÃO NUM_1 DIFERENTE DE ZERO
GOTO	ZERO	;SIM, ENTÃO NUM_1 IGUAL A ZERO

Outra aplicação também muito eficiente para a operação XOR é a inversão de um flag ou de um pino de saída. Imagine uma função que deve simplesmente inverter o estado de um sistema por meio de um flag denominado LIGADO, independentemente do estado anterior:

```

#define LIGADO FLAGS,0 ;FLAG QUE DETERMINA O ESTADO DO SISTEMA
; 0 -> DESLIGADO
; 1 -> LIGADO

;IMAGINEMOS QUE LIGADO É UM FLAG QUE PODE ESTAR EM QUALQUER ESTADO (0 OU 1)

INVERSAO
    MOVLW B'00000001' ;COLOCAMOS A "MÁSCARA" EM W. NESTE CASO DEVEMOS
                        ;COLOCAR 1 NOS BITS QUE DESEJAMOS INVERTER
    XORWF FLAGS,F ;DEVEMOS APPLICAR A OPERAÇÃO DIRETAMENTE AO
                    ;REGISTRADOR FLAGS, ONDE ESTÁ ARMAZENADO
                    ;O FLAG QUE DEVEMOS INVERTER
    RETURN           ;FINALIZA A ROTINA

```

Este mesmo tipo de código pode ser aplicado também a uma porta para a inversão da saída, para fazer, por exemplo, um LED ficar piscando.

COMPLEMENTO (COMF)

A operação de complemento, ao contrário das anteriores, trabalha com apenas um byte, invertendo o valor de cada um de seus bits. Ela recebe este nome porque, ao fazermos essa operação, estamos encontrando o complemento do número em relação a 255 (0xFF). Por exemplo, o complemento de 0 é 255, de 1 é 254, de 2 é 253 e assim sucessivamente. Veja o efeito prático da operação sobre o valor de um byte qualquer:

Byte	0	0	1	0	1	1	1	0
Complemento	1	1	0	1	0	0	0	1

A instrução responsável por essa operação dentro do assembly do PIC é:

- *COMF: Operação de complemento (COM) do registrador (F).*

A sintaxe dessa instrução é a seguinte:

COMF f,d ;ONDE f É O REGISTRADOR E d O DESTINO ONDE SERÁ ;GUARDADO O RESULTADO DA OPERAÇÃO

INVERSÃO (SWAPF)

A última operação implementada dentro do assembler do PIC é a Swap, que em português significa troca. Seu efeito sobre um byte é a inversão entre os 4 bits mais significativos e os quatro menos significativos. Veja o esquema seguinte para um melhor entendimento:

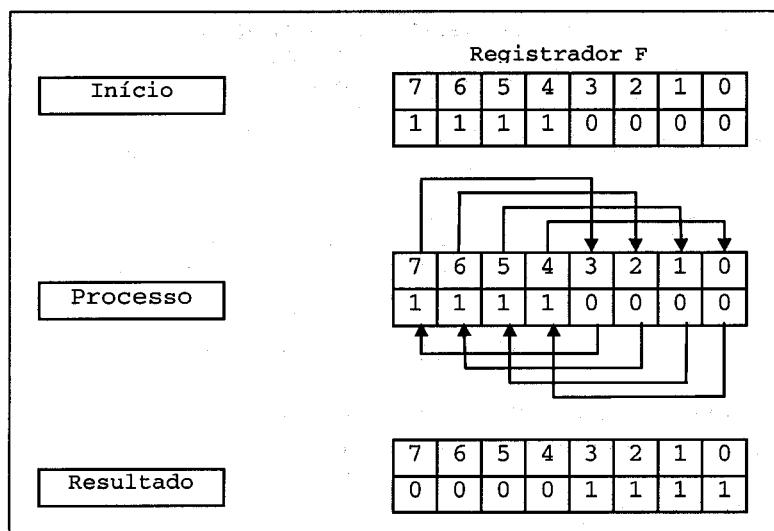


Figura 11.7 - Esquema da Instrução Swap.

CONTANDO TEMPO E CRIANDO DELAYS

A contagem de tempo dentro de sistemas microcontrolados é extremamente utilizada e importante. Uma vez que o sistema é digital e preciso (o nível de precisão depende do oscilador), o tempo pode ser facilmente mensurado, facilitando o gerenciamento de eventos, a freqüência de funcionamento de determinadas tarefas e muito mais.

O tempo dentro do PIC pode ser medido de três maneiras:

- Contando os ciclos de máquina por meio de loopings;
- Contando os ciclos de máquina por intermédio do contador Timer0 (TMR0) ou Timer1 (TMR1) ou Timer2 (TMR2); ou
- Contando pulsos externos por meio da entrada TOCKI e do TMR0 ou da entrada T1CKI e do TMR1.

As duas primeiras técnicas são as mais utilizadas e dependem diretamente da freqüência de trabalho do oscilador do sistema. A terceira, por sua vez, depende exclusivamente de um sinal externo, e pode ser facilmente utilizada para contar tempo com base nos 60Hz da rede elétrica.

UTILIZANDO REGISTROS TEMPORÁRIOS PARA CRIAR DELAYS

Inicialmente vamos estudar somente a primeira técnica comentada. O uso dos Timers serão vistos mais adiante.

A idéia aqui é criar contadores dentro de outros contadores para contar a quantidade certa de ciclos de máquina e, consequentemente, o tempo. O segredo é

calcular corretamente a quantidade de ciclos utilizados para cada looping. Para isso, lembre-se de que instruções que geram desvios, tais como: GOTO, CALL, BTFSS, BTFSC, DECFSZ e INCFSZ, podem utilizar dois ciclos, e não um.

Veja o código para uma rotina de 1ms:

```
DELAY          ; ROTINA DE 1ms
    MOVLW .250
    MOVWF TEMP1 ; INICIA TEMP1 COM 250
DL_1           ; PERDE UM CICLO SEM EFETUAR AÇÃO
    DECFSZ TEMP1,F ; DECREMENTA O CONTADOR TEMP1. ACABOU?
    GOTO DL_1      ; NÃO, CONTINUA ESPERANDO
    RETURN         ; SIM, FINALIZA A ROTINA
                ; LEMBRE-SE QUE MAIS 6 CICLOS SERÃO GASTOS NAS
                ; INSTRUÇÕES CALL, RETURN E NA INICIALIZAÇÃO DO TEMP1
```

Desta forma, cada looping (DL_1) dura quatro ciclos, que dura $4\mu s$. Como o contador TEMP1 será decrementado 250 vezes, a rotina dura 1ms. Observe que, para acertar a duração do looping em quatro ciclos, foi utilizada a instrução NOP. Ela simplesmente não faz nada, mas gasta um ciclo. Essa instrução deve ser usada quantas vezes forem necessárias, sempre que você precisar "gastar" algum tempinho.

Dica: 2 NOPs seguidos podem ser substituídos por GOTO \$+1, que também perde dois ciclos de máquina, mas consome apenas uma posição na memória de programa.

Caso a rotina tenha que durar mais tempo, outros contadores auxiliares podem ser criados, iniciados e decrementados. Basta estruturar o código corretamente e calcular os valores exatos.

EXEMPLO 3 - PISCA-PISCA

Este exemplo dedica-se ao emprego de delays e inversão de estados por meio da operação XOR. Ao invés de piscarmos um simples LED, piscaremos um grupo inteiro, definido pela constante DISPLAY. Desta forma, será possível mantermos um número, ou outro símbolo qualquer, piscando no display.

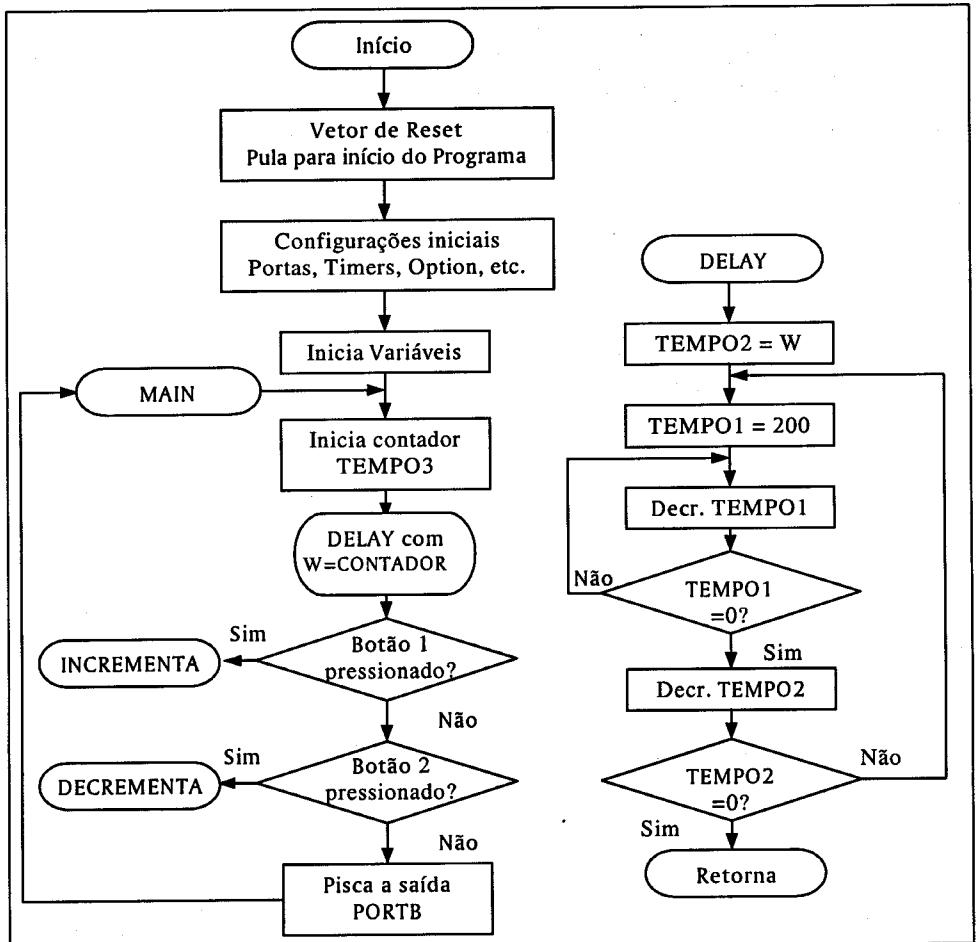


Figura 11.8 - Fluxograma do Exemplo 3 (Parte 1).

A constante DISPLAY será definida conforme os segmentos que devemos acender para construir o símbolo desejado. Os botões 1 e 2 (RA1 e RA2) serão utilizados para alterar o valor da variável CONTADOR, mudando com isso a freqüência do pisca-pisca. As constantes MIN e MAX definem os limites para o CONTADOR e, consequentemente, para a freqüência.

Observe que a rotina DELAY foi feita para múltiplos de milissegundos, mas não foi precisamente calibrada, pois desconsiderou os tempos gastos para entrada, saída e alguns processamentos internos.

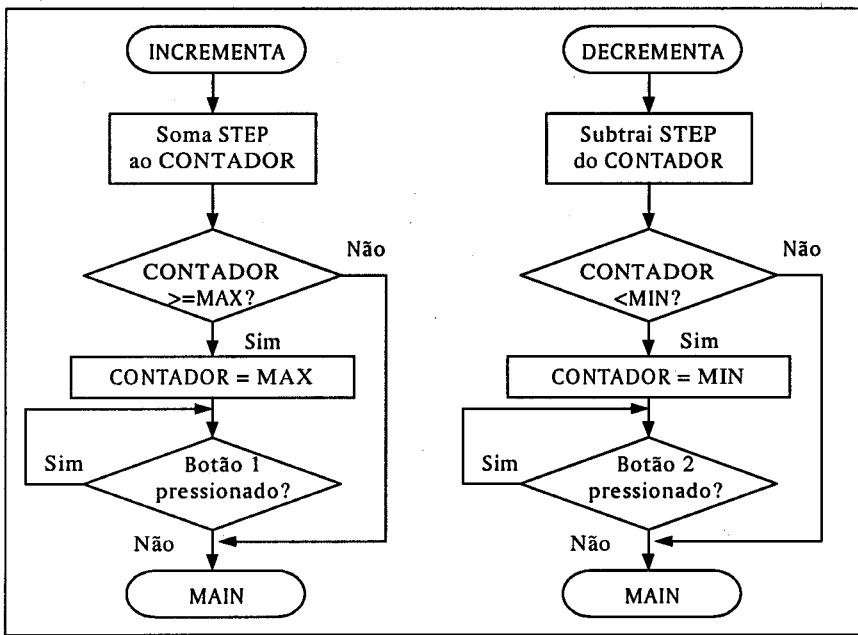


Figura 11.9 - Fluxograma do Exemplo 3 (Parte 2).

```

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*          PISCA-PISCA - EX3
;*          DESBRAVANDO O PIC
;*          DESENVOLVIDO PELA MOSAICO ENGENHARIA E CONSULTORIA
;*          VERSÃO: 1.0          DATA: 17/06/03
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*          DESCRIÇÃO DO ARQUIVO
;* -----
;*          PISCA-PISCA VARIÁVEL PARA DEMONSTRAR A IMPLEMENTAÇÃO DE
;*          DELAYS E A INVERSÃO DE PORTAS.
;*
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*          ARQUIVOS DE DEFINIÇÕES
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* INCLUDE <P16F628A.INC>      ;ARQUIVO PADRÃO MICROCHIP PARA 16F628A
;* _CONFIG _BOREN_ON & _CP_OFF & _PWRTE_ON & _WDT_OFF & _LVP_OFF &
;* _MCLRE_ON & _XT_OSC
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*          PAGINAÇÃO DE MEMÓRIA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* DEFINIÇÃO DE COMANDOS DE USUÁRIO PARA ALTERAÇÃO DA PÁGINA DE MEMÓRIA
;* 
;* #DEFINE      BANK0  BCF STATUS,RP0 ;SETA BANK 0 DE MEMÓRIA
;* #DEFINE      BANK1  BSF STATUS,RP0 ;SETA BANK 1 DE MAMÓRIA
;* 
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*          VARIÁVEIS
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

```

; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS
; PELO SISTEMA

```
CBLOCK 0x20           ;ENDEREÇO INICIAL DA MEMÓRIA DE
                      ;USUÁRIO
CONTADOR            ;BASE DE TEMPO PARA A PISCADA
FILTRO              ;FILTRAGEM PARA O BOTÃO
TEMPO1               ;REGISTRADORES AUXILIARES DE TEMPO
TEMPO2
TEMPO3
ENDC                ;FIM DO BLOCO DE MEMÓRIA
```

```
*****  
*  
*          FLAGS INTERNOS  
*  
*****  
; DEFINIÇÃO DE TODOS OS FLAGS UTILIZADOS PELO SISTEMA
```

```
*****  
*  
*          CONSTANTES  
*  
*****  
; DEFINIÇÃO DE TODAS AS CONSTANTES UTILIZADAS PELO SISTEMA
```

MIN	EQU	.10
MAX	EQU	.240
STEP	EQU	.5
MULTIPLIO	EQU	.5

; A CONSTANTE DISPLAY REPRESENTA O SÍMBOLO QUE APARECERÁ PISCANDO NO
DISPLAY. 1=LED LIGADO E 0=LED DESLIGADO. A RELAÇÃO ENTRE BITS E
SEGMENTOS É A SEGUINTE: 'EDC.BAFG'

```
;      a
;      *****
;      *
;      f *      * b
;      *      g *
;      *****
;      *
;      e *      * c
;      *      d *
;      *****
;
```

DISPLAY EQU B'10101011'; (LETRA H)

```
*****  
*
```

;* ENTRADAS
*

```
*****  
*
```

; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO ENTRADA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)

```
#DEFINE     BT1     PORTA,1      ;BOTÃO 1 - INCREMENTA
                      ; 0 -> PRESSIONADO
                      ; 1 -> LIBERADO
#DEFINE     BT2     PORTA,2      ;BOTÃO 2 - DECREMENTA
```

; 0 -> PRESSIONADO
 ; 1 -> LIBERADO

```
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  

;:  

; SAÍDAS  

; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO SAÍDA  

; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)
```

```
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  

;:  

; VETOR DE RESET  

;***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

ORG 0x00 ;ENDEREÇO INICIAL DE PROCESSAMENTO
 GOTO INICIO

```
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  

;:  

; INÍCIO DA INTERRUPÇÃO  

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

; AS INTERRUPÇÕES NÃO SERÃO UTILIZADAS, POR ISSO PODEMOS SUBSTITUIR  

; TODO O SISTEMA EXISTENTE NO ARQUIVO MODELO PELO APRESENTADO ABAIXO  

; ESTE SISTEMA NÃO É OBRIGATÓRIO, MAS PODE EVITAR PROBLEMAS FUTUROS
```

ORG 0x04 ;ENDEREÇO INICIAL DA INTERRUPÇÃO
 RETFIE ;RETORNA DA INTERRUPÇÃO

```
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *  

;:  

; ROTINA DE DELAY  

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

; ESTA ROTINA AGUARDA TANTOS MILISEGUNDOS QUANTO O VALOR PASSADO  

; POR W. POR EXEMPLO, SE W = .200, ELA AGUARDARÁ 200 MILISEGUNDOS.  

;  

; O DELAY PRINCIPAL DURA 1ms, POIS POSSUI 5 INSTRUÇÕES (5us) E É  

; RODADO 200 VEZES (TEMPO1). PORTANTO 200 * 5us = 1ms.  

; O DELAY PRINCIPAL É RODADO TANTAS VEZES QUANTO FOR O VALOR DE  

; TEMPO2, O QUAL É INICIADO COM O VALOR PASSADO EM W.
```

DELAY

MOVWF TEMPO2 ;INICIA TEMPO 2 COM O VALOR
;PASSADO EM W

DL1

MOVLW .200
 MOVWF TEMPO1

DL2

;ESTE DELAY DURA 1ms (5*200)

NOP
 NOP
 DECFSZ TEMPO1,F ;DECREMENTA TEMPO1. ACABOU?
 GOTO DL2 ;NÃO, CONTINUA AGUARDANDO
;SIM

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

RETURN

GOTO	MAIN	; COMEÇA NOVAMENTE
 DECREMENTA		
MOVLW	STEP	
SUBWF	CONTADOR, F	; DECREMENTA O CONTADOR EM STEP
MOVLW	MIN	; MOVE O VALOR MÍNIMO PARA W
SUBWF	CONTADOR, W	; SUBTRAI O VALOR DE W (MIN) DE CONTADOR
BTFSC	STATUS, C	; TESTA CARRY. RESULTADO NEGATIVO?
GOTO	MAIN	; NÃO, ENTÃO CONTA >= MIN ; SIM, ENTÃO CONTA < MIN
MOVLW	MIN	
MOVWF	CONTADOR	; ACERTA CONTADOR NO MÍNIMO, POIS ; PASSOU DO VALOR
BTFSS	BT2	; BOTÃO 2 CONTINUA PRESSIONADO?
GOTO	\$-1	; SIM, AGUARDA LIBERAÇÃO ; NÃO
GOTO	MAIN	; VOLTA AO LOOP PRINCIPAL
 INCREMENTA		
MOVLW	STEP	
ADDWF	CONTADOR, F	; INCREMENTA O CONTADOR EM STEP
MOVLW	MAX	; MOVE O VALOR MÁXIMO PARA W
SUBWF	CONTADOR, W	; SUBTRAI O VALOR DE W (MIN) DE CONTADOR
BTFSS	STATUS, C	; TESTA CARRY. RESULTADO NEGATIVO?
GOTO	MAIN	; SIM, ENTÃO CONTA < MAX ; NÃO, ENTÃO CONTA >= MAX
MOVLW	MAX	
MOVWF	CONTADOR	; ACERTA CONTADOR NO MÁXIMO, POIS ; PASSOU DO VALOR
BTFSS	BT1	; BOTÃO 1 CONTINUA PRESSIONADO?
GOTO	\$-1	; SIM, AGUARDA LIBERAÇÃO ; NÃO
GOTO	MAIN	; VOLTA AO LOOP PRINCIPAL
, * * * * *		* * * * *
;		*
;	FIM DO PROGRAMA	*
END	; OBRIGATÓRIO	* * * * *

Agora experimente seus conhecimentos com os exercícios propostos:

1. Altere os valores das constantes MIN, MAX e STEP.
2. Crie rotinas de DELAYs com tempos ou estruturas diferentes.

OPERANDO DIRETAMENTE COM O PROGRAM COUNTER

Como já dissemos anteriormente, existe um SFR denominado PCL que sempre contém o endereço da memória de programa que será executado em seguida. O interessante desse registrador é que, se alterarmos diretamente o seu valor, mudamos também o ponto de execução do programa. Mas tome muito cuidado com essas alterações para que seu programa não vá parar em um ponto desconhecido.

USANDO O PCL PARA ESCOLHER ENTRE VÁRIAS ROTINAS

Uma das aplicações ideais para trabalhar diretamente com o PCL é a necessidade de desviar para uma rotina específica de acordo com um determinado índice. Por exemplo, suponhamos que um sistema possua um teclado de seis botões que são monitorados por uma rotina específica. O número do botão pressionado (1 a 6) é colocado na variável TECLA. Se nenhum botão está pressionado, então TECLA = 0. Veja agora o código que chamará a rotina certa, dependendo da tecla pressionada:

TRATA_TECLA		; ROTINA DE TRATAMENTO DA TECLA
MOVLW B'00000111'		
ANDWF TECLA,W		; MASCARA TECLA PARA LIMITAR VALOR EM 7
ADDWF PCL,F		; SOMA TECLA AO PCL PARA PULAR PARA O PONTO CERTO
GOTO SEM_TECLA		; TECLA = 0 -> NÃO HÁ TECLA PRESSIONADA
GOTO TECLA1		; TECLA = 1 -> PULA PARA ROTINA TECLA1
GOTO TECLA2		; TECLA = 2 -> PULA PARA ROTINA TECLA2
GOTO TECLA3		; TECLA = 3 -> PULA PARA ROTINA TECLA3
GOTO TECLA4		; TECLA = 4 -> PULA PARA ROTINA TECLA4
GOTO TECLA5		; TECLA = 5 -> PULA PARA ROTINA TECLA5
GOTO TECLA6		; TECLA = 6 -> PULA PARA ROTINA TECLA6
GOTO ERRO		; TECLA = 7 -> NÃO EXISTE

O mascaramento do índice para limitar seu valor antes de somá-lo ao PCL é extremamente importante para evitar que um erro de cálculo trave o sistema.

Como o PCL controla somente os 8 bits menos significativos do Program Counter, essa rotina nunca pode ser implementada entre blocos da memória de programação (cada bloco possui 256 endereços).

USANDO O PCL PARA MONTAR UMA TABELA DE VALORES

Outro exemplo de aplicação para alteração do PCL é a criação de uma tabela na área de programação. O conceito é muito similar ao apresentado anteriormente, só que em vez de desviar para rotinas específicas, um número é retornado diretamente para cada valor do índice. Podemos utilizar essas tabelas, por exemplo, para converter valores nas mais diversas aplicações:

CONVERTE		; ROTINA DE CONVERSÃO (NUM * 3 + 2)
MOVLW B'00000111'		
ANDWF NUM,W		; MASCARA NUM PARA LIMITAR VALOR EM 7
ADDWF PCL,F		; SOMA TECLA AO PCL PARA PULAR PARA O PONTO CERTO
RETLW .2		; NUM = 0 -> RETORNA 2
RETLW .5		; NUM = 1 -> RETORNA 5
RETLW .8		; NUM = 2 -> RETORNA 8
RETLW .11		; NUM = 3 -> RETORNA 11
RETLW .14		; NUM = 4 -> RETORNA 14
RETLW .17		; NUM = 5 -> RETORNA 17
RETLW .20		; NUM = 6 -> RETORNA 20
RETLW .23		; NUM = 7 -> RETORNA 23

EXEMPLO 4 - CONTADOR MELHORADO

O Exemplo 4 é um aperfeiçoamento do Exemplo 2, com a utilização de dois botões para incrementar e decrementar o contador unitariamente. Outra grande diferença entre os exemplos em questão é que agora o valor do contador é mostrado no display de forma hexadecimal, e não mais nos LEDs de forma binária. Para isso, utilizamos o recurso de soma do Program Counter (PC) para criarmos uma tabela de conversão. Como só temos um display, temos de limitar nosso contador em 0F (15).

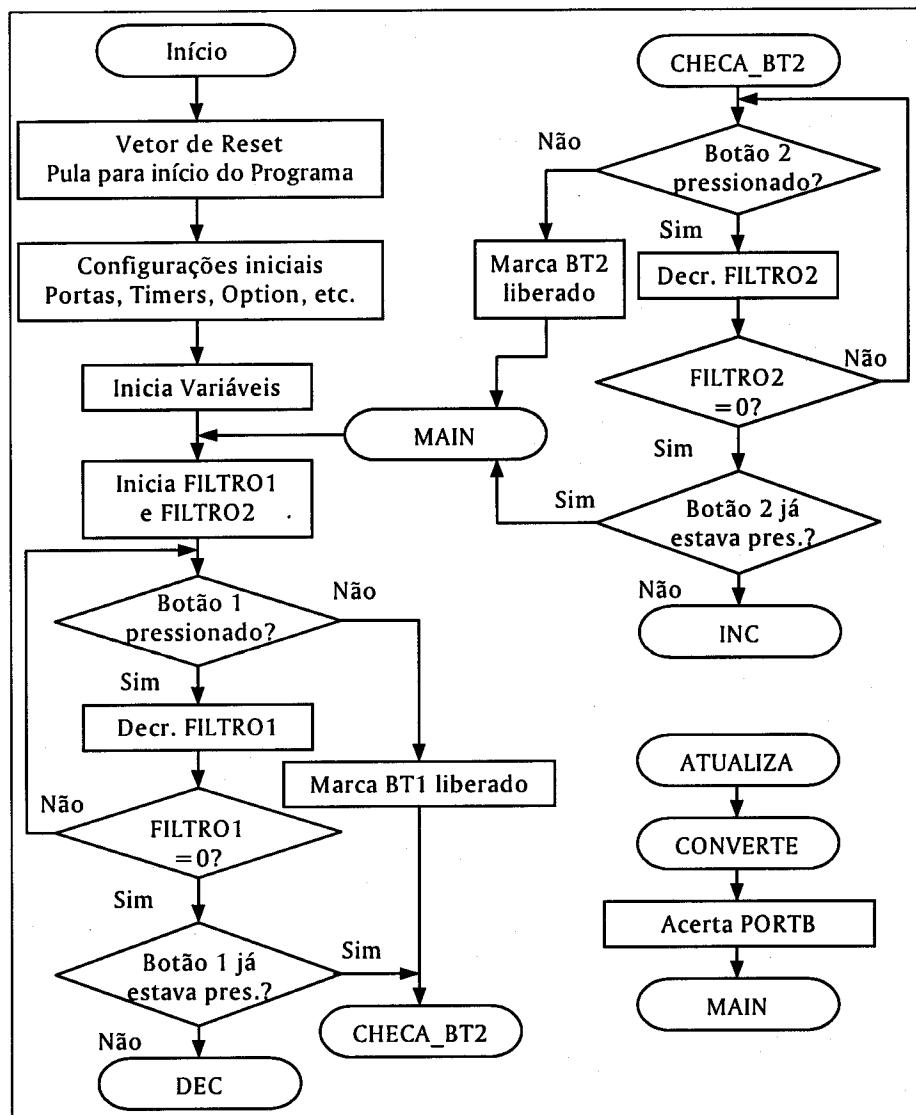


Figura 11.10 - Fluxograma do Exemplo 4 (Parte 1).

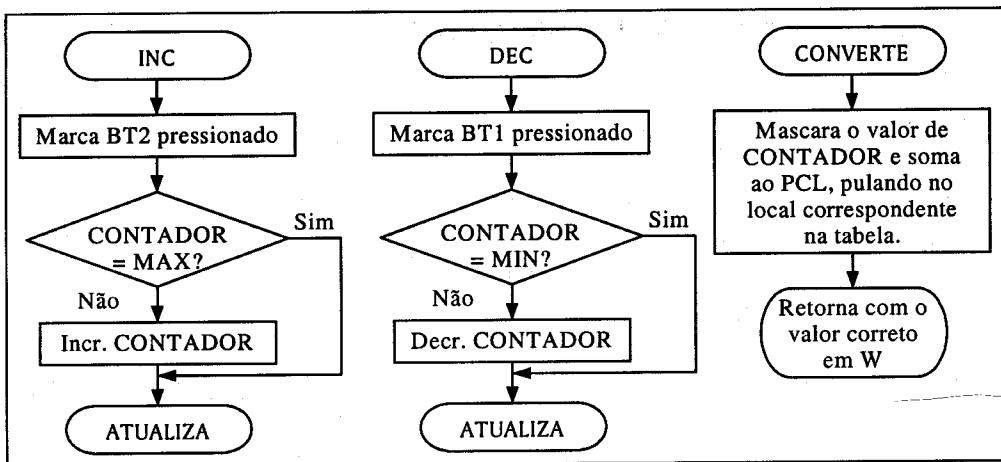


Figura 11.11 - Fluxograma do Exemplo 4 (Parte 2).

```
        ;USUÁRIO
CONTADOR      ;ARMAZENA O VALOR DA CONTAGEM
FLAGS          ;ARMAZENA OS FLAGS DE CONTROLE
FILTRO1        ;FILTRAGEM PARA O BOTÃO 1
FILTRO2        ;FILTRAGEM PARA O BOTÃO 2
ENDC          ;FIM DO BLOCO DE MEMÓRIA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           FLAGS INTERNOS           *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS FLAGS UTILIZADOS PELO SISTEMA

#define      ST_BT1 FLAGS,0 ; STATUS DO BOTÃO 1
#define      ST_BT2 FLAGS,1 ; STATUS DO BOTÃO 2

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           CONSTANTES            *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODAS AS CONSTANTES UTILIZADAS PELO SISTEMA

MIN          EQU     .0      ;VALOR MÍNIMO PARA O CONTADOR
MAX          EQU     .15     ;VALOR MÁXIMO PARA O CONTADOR
T_FILTRO    EQU     .255    ;FILTRO PARA BOTÃO

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           ENTRADAS             *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO ENTRADA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)

#define      BOTAO1 PORTA,1       ;PORTA DO BOTÃO
;          ; 0 -> PRESSIONADO
;          ; 1 -> LIBERADO
#define      BOTAO2 PORTA,2       ;PORTA DO BOTÃO
;          ; 0 -> PRESSIONADO
;          ; 1 -> LIBERADO

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           SAÍDAS              *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO SAÍDA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)

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

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           INÍCIO DA INTERRUPÇÃO   *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; AS INTERRUPÇÕES NÃO SERÃO UTILIZADAS, POR ISSO PODEMOS SUBSTITUIR
; TODO O SISTEMA EXISTENTE NO ARQUIVO MODELO PELO APRESENTADO ABAIXO
; ESTE SISTEMA NÃO É OBRIGATÓRIO, MAS PODE EVITAR PROBLEMAS FUTUROS

ORG          0x04      ;ENDEREÇO INICIAL DA INTERRUPÇÃO
```

RETFIE

; RETORNA DA INTERRUPÇÃO

```
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*          ROTINA DE CONVERSÃO BINÁRIO -> DISPLAY
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; ESTA ROTINA IRÁ RETORNAR EM W, O SÍMBOLO CORRETO QUE DEVE SER
; MOSTRADO NO DISPLAY PARA CADA VALOR DE CONTADOR. O RETORNO JÁ ESTÁ
; FORMATADO PARA AS CONDIÇÕES DE LIGAÇÃO DO DISPLAY AO PORTB.
```

```
;      a
;      *****
;      *
;      f *      * b
;      *      g *
;      *****
;      *
;      e *      * c
;      *      d *
;      *****
;
```

CONVERTE

MOVF	CONTADOR,W	; COLOCA CONTADOR EM W	
ANDLW	B'00001111'	; MASCARA VALOR DE CONTADOR	
		; CONSIDERAR SOMENTE ATÉ 15	
ADDWF	PCL,F		
		; 'EDC.BAFG'	; POSIÇÃO CORRETA DOS SEGUIMENTOS
RETLW	B'11101110'	; 00 - RETORNA SÍMBOLO CORRETO 0	
RETLW	B'00101000'	; 01 - RETORNA SÍMBOLO CORRETO 1	
RETLW	B'11001101'	; 02 - RETORNA SÍMBOLO CORRETO 2	
RETLW	B'01101101'	; 03 - RETORNA SÍMBOLO CORRETO 3	
RETLW	B'00101011'	; 04 - RETORNA SÍMBOLO CORRETO 4	
RETLW	B'01100111'	; 05 - RETORNA SÍMBOLO CORRETO 5	
RETLW	B'11100111'	; 06 - RETORNA SÍMBOLO CORRETO 6	
RETLW	B'00101100'	; 07 - RETORNA SÍMBOLO CORRETO 7	
RETLW	B'11101111'	; 08 - RETORNA SÍMBOLO CORRETO 8	
RETLW	B'01101111'	; 09 - RETORNA SÍMBOLO CORRETO 9	
RETLW	B'10101111'	; 10 - RETORNA SÍMBOLO CORRETO A	
RETLW	B'11100011'	; 11 - RETORNA SÍMBOLO CORRETO b	
RETLW	B'11000110'	; 12 - RETORNA SÍMBOLO CORRETO C	
RETLW	B'11101001'	; 13 - RETORNA SÍMBOLO CORRETO d	
RETLW	B'11000011'	; 14 - RETORNA SÍMBOLO CORRETO E	
RETLW	B'10000011'	; 15 - RETORNA SÍMBOLO CORRETO F	

```
***** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*          INICIO DO PROGRAMA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

INICIO

BANK1		; ALTERA PARA O BANCO 1
MOVlw	B'00000010'	
MOVWF	TRISA	; DEFINE RA1 E 2 COMO ENTRADA E DEMAIS ; COMO SAÍDAS
MOVlw	B'00000000'	
MOVWF	TRISB	; DEFINE TODO O PORTB COMO SAÍDA
MOVlw	B'10000000'	
MOVWF	OPTION_REG	; PRESCALER 1:2 NO TMR0 ; PULL-UPS DESABILITADOS

```

        ;AS DEMAIS CONFIG. SÃO IRRELEVANTES
MOVLW B'00000000'
MOVWF INTCON      ;TODAS AS INTERRUPÇÕES DESLIGADAS
BANK0             ;RETORNA PARA O BANCO 0
MOVLW B'00000111'
MOVWF CMCON       ;DEFINE O MODO DE OPERAÇÃO DO COMPARADOR

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           INICIALIZAÇÃO DAS VARIÁVEIS
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

CLRFB PORTA        ;LIMPA O PORTA
CLRF PORTB         ;LIMPA O PORTB
CLRF FLAGS         ;LIMPA TODOS OS FLAGS
MOVLW MIN          ;INICIA CONTADOR = MIN
MOVWF CONTADOR     ;INICIA CONTADOR = MIN
GOTO ATUALIZA      ;ATUALIZA O DISPLAY INICIALMENTE

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           ROTINA PRINCIPAL
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

MAIN
    MOVLW T_FILTRO
    MOVWF FILTRO1     ;INICIALIZA FILTRO1 = T_FILTRO
    MOVWF FILTRO2     ;INICIALIZA FILTRO2 = T_FILTRO

CHECA_BT1
    BTFSC BOTAO1      ;O BOTÃO 1 ESTÁ PRESSIONADO?
    GOTO BT1_LIB       ;NÃO, ENTÃO TRATA COMO LIBERADO
    ;SIM
    DECFSZ FILTRO1,F  ;DECREMENTA O FILTRO DO BOTÃO
    ;TERMINOU?
    GOTO CHECA_BT1    ;NÃO, CONTINUA ESPERANDO
    ;SIM
    BTFSS ST_BT1      ;BOTÃO JÁ ESTAVA PRESSIONADO?
    GOTO DEC          ;NÃO, EXECUTA AÇÃO DO BOTÃO
    GOTO CHECA_BT2    ;SIM, CHECA BOTÃO 2

BT1_LIB
    BCF ST_BT1        ;MARCA BOTÃO 1 COMO LIBERADO

CHECA_BT2
    BTFSC BOTAO2      ;O BOTÃO 2 ESTÁ PRESSIONADO?
    GOTO BT2_LIB       ;NÃO, ENTÃO TRATA COMO LIBERADO
    ;SIM
    DECFSZ FILTRO2,F  ;DECREMENTA O FILTRO DO BOTÃO
    ;TERMINOU?
    GOTO CHECA_BT2    ;NÃO, CONTINUA ESPERANDO
    ;SIM
    BTFSS ST_BT2      ;BOTÃO JÁ ESTAVA PRESSIONADO?
    GOTO INC          ;NÃO, EXECUTA AÇÃO DO BOTÃO
    GOTO MAIN          ;SIM, VOLTA AO LOOPING

BT2_LIB
    BCF ST_BT2        ;MARCA BOTÃO 2 COMO LIBERADO
    GOTO MAIN          ;RETORNA AO LOOPING

```

DEC	BSF ST_BT1	;AÇÃO DE DECREMENTAR
	MOVF CONTADOR,W	;MARCA BOTÃO 1 COMO JÁ PRESSIONADO
	XORLW MIN	;COLOCA CONTADOR EM W
		;APLICA XOR ENTRE CONTADOR E MIN
		;PARA TESTAR IGUALDADE. SE FOREM
		;IGUAIS, O RESULTADO SERÁ ZERO
	BTFSC STATUS,Z	;RESULTOU EM ZERO?
	GOTO MAIN	;SIM, RETORNA SEM AFETAR CONT.
		;NÃO
	DECFSZ CONTADOR,F	;DECREMENTA O CONTADOR
	GOTO ATUALIZA	;ATUALIZA O DISPLAY
INC	BSF ST_BT2	;AÇÃO DE INCREMENTAR
	MOVF CONTADOR,W	;MARCA BOTÃO 2 COMO JÁ PRESSIONADO
	XORLW MAX	;COLOCA CONTADOR EM W
		;APLICA XOR ENTRE CONTADOR E MAX
		;PARA TESTAR IGUALDADE. SE FOREM
		;IGUAIS, O RESULTADO SERÁ ZERO
	BTFSC STATUS,Z	;RESULTOU EM ZERO?
	GOTO MAIN	;SIM, RETORNA SEM AFETAR CONT.
		;NÃO
	INCF CONTADOR,F	;INCREMENTA O CONTADOR
	GOTO ATUALIZA	;ATUALIZA O DISPLAY
ATUALIZA	CALL CONVERTE	;CONVERTE CONTADOR NO NÚMERO DO
		;DISPLAY
	MOVWF PORTB	;ATUALIZA O PORTB PARA
		;VISUALIZARMOS O VALOR DE CONTADOR
		;NO DISPLAY
	GOTO MAIN	;NÃO, VOLTA AO LOOP PRINCIPAL
;	*****	*****
;	FIM DO PROGRAMA	*
;	*****	*
END	OBRIGATÓRIO	

Aproveite o momento para treinar um pouco mais tudo o que já foi possível aprender. Faça as seguintes alterações no exemplo apresentado:

1. Altere as rotinas INC e DEC para que o contador seja alterado de 2 em 2 unidades.
 2. Inverta o botão 1 com o botão 2.
 3. Implemente um terceiro botão que habilita/desabilita o contador, isto é, a ação dos outros dois botões já existentes. Um LED pode indicar se o contador está habilitado ou não.

EXPLORANDO AS INTERRUPÇÕES

Finalmente, você poderá conhecer e explorar as potencialidades das interrupções do PIC. Nesta seção, você irá descobrir como são poderosas essas interrupções e para

que realmente elas servem. No decorrer deste, já foi apresentada uma introdução e várias referências às interrupções, mas agora chegou a hora de nos aprofundarmos no assunto, aprendendo a trabalhar com elas. Mais detalhes sobre a finalidade de cada uma das interrupções existentes podem ser encontrados no Capítulo 4.

LIGANDO AS CHAVES CORRETAS

A primeira operação relativa ao uso das interrupções refere-se às suas chaves de habilitação. Existem três tipos de chaves:

- **Individuais:** São as chaves que habilitam e desabilitam cada uma das interrupções individualmente (geralmente denominadas Máscaras).
- **Grupo:** No PIC 16F628A, e em muitos outros modelos, existe um tipo especial de chave que controla a habilitação de todo um grupo de interrupções, denominadas interrupções de periféricos.
- **Geral:** Desabilita todas as interrupções simultaneamente ou habilita todas aquelas que estão com suas chaves individuais ligadas.

Portanto, para utilizarmos uma determinada interrupção, devemos primeiramente ligar sua chave individual, ligando depois a chave de grupo (se existir) e por último a chave geral. Os exemplos práticos dessas chaves serão vistos logo adiante.

A ESTRUTURA BÁSICA DA ROTINA DE INTERRUPÇÃO (RETFIE)

Quando uma interrupção estiver habilitada (ligada), sempre que a ação responsável pela geração dela ocorrer, o programa será paralisado e automaticamente desviado para o endereço 0x04, independentemente do tipo de interrupção. Mas não se preocupe, o sistema guarda (na pilha) o local do programa que estava sendo executado para que ele possa retornar ao mesmo ponto após a interrupção ter sido tratada. Entretanto, para retornar da rotina de interrupção, não devemos utilizar as instruções de retorno convencionais, e sim a instrução RETFIE.

Antes de efetuar qualquer tarefa dentro da rotina de interrupção, lembre-se de que tudo que for alterado em relação às variáveis do sistema permanecerá desta forma, quando o sistema voltar a ser executado no ponto em que estava. Por isso, devemos ter muito cuidado com as variáveis, principalmente as temporárias, utilizadas enquanto estamos tratando a interrupção. Por exemplo, se no meio do programa estamos utilizando uma variável denominada TEMP para guardar um valor qualquer temporariamente, e dentro da interrupção utilizarmos essa mesma variável para outra finalidade, quando voltarmos ao programa principal, poderemos ter alterado toda a lógica do sistema.

Com base nesta mesma observação, é necessário também que os registradores importantes ao funcionamento do programa, tais como: o W e o STATUS, não sejam alterados indevidamente. Mas como não afetar esses registradores, se alguns de seus bits são controlados pelo próprio hardware? Não existe uma maneira de evitarmos que esses registradores sejam afetados durante o tratamento da inter-

-rupção, pois eles são necessários para a execução de qualquer código. A única maneira de resolvemos o impasse, então, é guardarmos seus valores em registradores temporários logo que entramos na interrupção, para recuperá-los na hora de sairmos.

Vejamos como fica a estrutura básica de uma rotina de tratamento de interrupções:

Observe a maneira como W e STATUS são gravados na memória em W_TEMP e STATUS_TEMP. Este código deve ser sempre repetido desta mesma maneira, para evitarmos alterações nos registradores enquanto estamos tentando gravá-los. A utilização do SWAPF em substituição ao MOVF evita a alteração do STATUS. Para alguns modelos de PIC que possuem outros registradores importantes ao funcionamento, que também não podem ser afetados pela interrupção, tal como o PCLATCH, esta estrutura deve ser incrementada para salvar e recuperar todos os registradores necessários.

Outra dica extremamente útil é que, como a interrupção pode ocorrer a qualquer momento, não temos como saber qual era o banco de memória selecionado no dado momento. Por isso, recomendamos que, após gravar os valores de W e STATUS, você selecione um banco específico com o qual irá trabalhar.

CHECANDO QUAL FOI A INTERRUPÇÃO OCORRIDA

Agora que já temos uma estrutura básica, que é única para tratar todo tipo de interrupção, você deve estar se perguntando: Como vou saber, então, qual foi a interrupção que aconteceu? Este será nosso próximo passo e você descobrirá como é fácil.

Quando a ação responsável pela interrupção acontece, o microcontrolador processa o seguinte algoritmo:

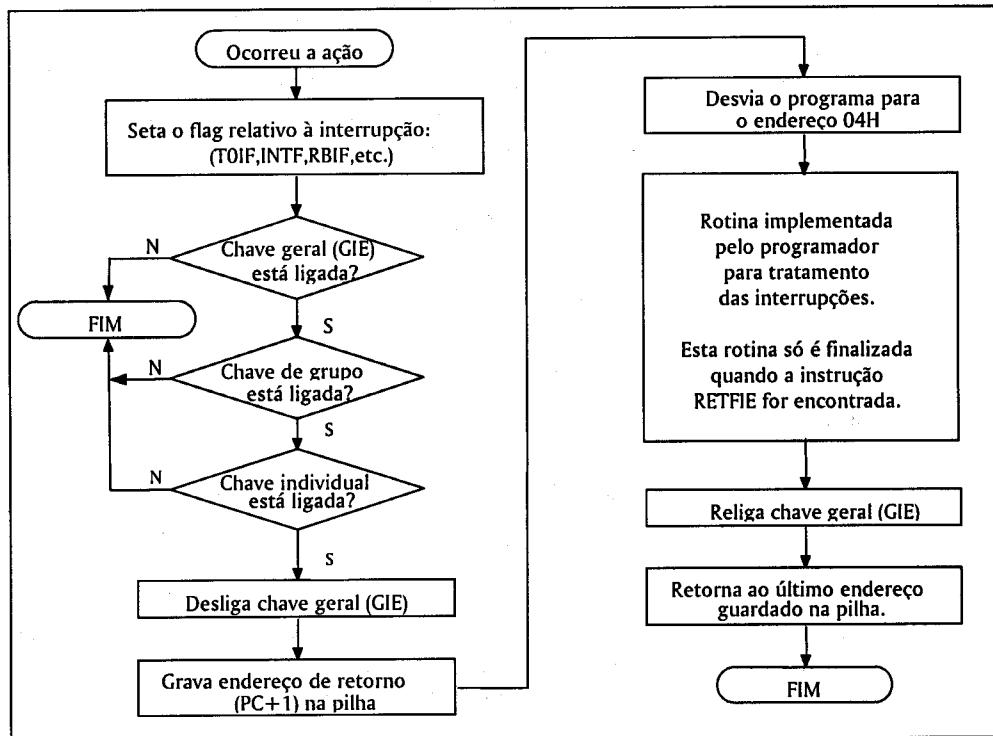


Figura 11.12 - Fluxograma das Interrupções.

Desta maneira, quando uma interrupção acontece, o flag relativo a ela é imediatamente setado. Por meio desse flag será possível, então, descobrir qual foi a interrupção ocorrida. Observe que o flag é alterado automaticamente, mesmo que as chaves de habilitação da interrupção estejam desligadas, evitando que o desvio aconteça. Isso pode ser utilizado, por exemplo, para executarmos uma tarefa relativa à ação da interrupção, sem, contudo, entrarmos na interrupção propriamente dita.

Observe também que a chave geral (INTCON,GIE) é desligada automaticamente antes de o programa desviar para o tratamento da interrupção, sendo religada, também automaticamente, através do RETFIE, antes de voltarmos dela. Isso é necessário para que uma nova interrupção não aconteça durante o tratamento. No

entanto, é possível religarmos o GIE manualmente dentro da rotina de interrupção, para sobrecarregarmos o processo, mas isso deve ser feito com muito cuidado para não travar o sistema. Neste caso, por exemplo, o W e o STATUS deveriam ser gravados novamente em outras duas variáveis que não o W_TEMP e o STATUS_TEMP.

Ao contrário do GIE, os flags relativos às interrupções são setados automaticamente pelo hardware, mas não são limpos por ele. A limpeza desses flags deve ser feita manualmente pelo software, como será visto em seguida. Se isso não for feito, uma nova interrupção acontecerá tão logo o programa saia da interrupção atual.

Por último vale comentar que nem todas as interrupções pertencem ao grupo de interrupções de periféricos. Por isso, a checagem da chave de grupo só será realizada caso a interrupção ocorrida faça parte desse grupo. As únicas interrupções que não fazem parte do grupo de periféricos são: interrupção externa, interrupção por mudança de estado e interrupção de Timer 0.

CONHECENDO MELHOR O TIMER 0 E O PRESCALER

O Timer 0 é um dos melhores auxiliares para a contagem do tempo dentro do PIC. Ele nada mais é que um contador automático de 8 bits para os ciclos de máquina ou pulsos externos. O Timer 0 é armazenado diretamente na RAM do sistema na posição 0x01 e é denominado internamente como TMRO.

Para que o TMRO seja realmente poderoso na sua função de contagem do tempo, o bit TOCS do registrador OPTION permite escolher a maneira como ele será incrementado: por meio dos ciclos de máquina (TOCS = 0) ou de pulsos externos (TOCS = 1). No caso do TMRO estar sendo utilizado para a contagem de pulsos externos, por meio do pino 3, é importante checar também o estado do bit TOSE (OPTION) que irá dizer ao sistema se o incremento deverá ser feito na borda de subida (TOSE=0) ou na borda de descida (TOSE=1) do pulso. O TMRO incrementado por pulsos externos pode ser utilizado para muitas outras finalidades além da contagem de tempo, tais como: freqüencímetros, contadores etc.

CONHECENDO O PRESCALER

Acabamos de ver que o TMRO pode ser incrementado pelo clock da máquina ou por um sinal externo, assim como o WDT é incrementado automaticamente e estoura a cada 18 ms. Desta maneira, é fácil efetuarmos cálculos de tempo com esses contadores. Só que ambos executam ações específicas no caso de estouro (interrupção ou reset). Como alterar então o tempo de estouro de cada um desses contadores? Para o TMRO fica mais fácil, pois alterando o oscilador, alteramos também os seus incrementos. Mas isso é viável? E para o WDT, como podemos alterar seu tempo? Esta é a função do Prescaler. Trata-se de um divisor configurável que pode ser aplicado a um dos dois contadores. Temos aqui então a maior limitação do Prescaler, pois ele só pode ser aplicado a um dos contadores de cada

vez (veja bit PSA do OPTION). Com esse divisor ativo, o tempo de estouro é multiplicado na ordem inversa. Vejamos um exemplo:

Um sistema está rodando a 4 MHz. Desta forma, temos um ciclo de máquina e, consequentemente, uma instrução sendo rodada a cada 1 μ s. Sem o prescaler, o TMRO também seria incrementado a cada 1 μ s, estourando em 256 μ s. Se configurarmos o prescaler em 1:4 (veja bits PS2, PS1 e PS0 do OPTION), o TMRO só será incrementado a cada quatro ciclos de máquina, que neste caso correspondem a 4 μ s. O estouro acontecerá então em 1024 μ s.

A mesma analogia pode ser aplicada ao WDT. Sem prescaler, o WDT estoura em aproximadamente 18 ms. Se configurarmos o prescaler em 1:4, ele estourará em aproximadamente 72 ms.

Observe ainda que não existe uma opção de configuração do prescaler de 1:1 para o TMRO (veja a tabela dos bits PS2, PS1 e PS0 do OPTION). Para conseguir isso, é necessário aplicar o prescaler ao WDT. Desta maneira, o TMRO ficará sem prescaler, o que resultará em um incremento de 1:1.

UTILIZANDO O TIMER PARA MARCAR TEMPO

Uma vez definida e ajustada a forma como o TMRO será incrementado, o próximo passo é definirmos a base de tempo que irá compor o sistema. Para isso, precisamos inicialmente da freqüência do oscilador. Em todos os exemplos deste livro, utilizaremos uma freqüência padrão de 4 MHz, mas os valores podem ser ajustados de acordo com suas necessidades. Como já explicamos anteriormente, os ciclos de máquinas do PIC rodam numa freqüência quatro vezes menor que o clock principal. Neste caso, então, a freqüência interna é de 1 MHz, ou seja, um ciclo acontece a cada 1 μ s. Esta é, portanto, a nossa primeira base de tempo. Mas suponhamos que o que realmente queremos é uma rotina que demore 1ms. Para atingirmos esse objetivo, devemos trabalhar com duas variáveis: o valor do TMRO e o prescaler (PS). Por exemplo, inicialmente regularemos o PS para 1:4. Nestas condições, seriam necessários 250 incrementos do TMRO para contar 1 ms, certo? Acompanhe a conta:

$$\text{Tempo do ciclo (1}\mu\text{s)} \times \text{Prescale (4)} \times \text{TMRO (250)} = 1 \text{ ms}$$

Vejamos, então, o código de uma função que aguarda 1 ms:

```
;CONSIDEREMOS QUE O PS ESTÁ CONFIGURADO EM 1:4

DELAY          ;ROTINA DE 1ms
    CLRFL TMRO      ;LIMPA O TIMER 0. ESTE COMANDO LIMPA TAMBÉM O PS
DL_1
    MOVLW .250
    SUBWF TMRO,W    ;SUBTRAI O TMRO DE 250 GUARDANDO EM W
    BTFSS STATUS,C   ;TMRO >= 250?
    GOTO DL_1        ;NÃO, AINDA NÃO ACABOU O TEMPO. CONTINUA ESPERANDO
    RETURN           ;SIM, FINALIZA A ROTINA
```

Esta rotina não é extremamente precisa, pois o TMRO pode ser incrementado em qualquer uma das linhas. Por isso não devemos arriscar a confiar simplesmente na comparação de igualdade, pois, dependendo do PS e do tamanho da rotina, o TMRO pode ultrapassar o valor-limite antes de passar pela instrução de comparação. A maneira precisa de contar será atingida através da interrupção de Timer 0.

Uma outra maneira de implementar esta rotina é iniciarmos o TMR0 com 6 (256-250) e depois ficar esperando o TMR0 chegar em zero. Neste caso, fica mais difícil efetuar a comparação "maior ou igual".

Outra observação importante que podemos fazer sobre este código é o fato de que, como o TMRO é incrementado automaticamente, não é obrigatório ficarmos parados dentro de uma rotina para contarmos o tempo. Isso pode ser feito na rotina principal do sistema, desde que façamos uso da interrupção.

TRATANDO A INTERRUPÇÃO DE TIMER

Como ela ocorre quando o TMRO estoura, pode ser usada para criar bases de tempo com extrema precisão. Acertando o prescaler e o valor inicial do TMRO, podemos mensurar tempos nos mais diversos valores.

Para que essa interrupção possa ocorrer, sua chave individual INTCON.T0IE deve estar ligada (1). Dentro da rotina de tratamento devemos testar o flag INTCON.T0IF, para sabermos se foi essa interrupção que ocorreu. Antes de sairmos do tratamento, devemos limpar esse mesmo flag manualmente.

Veja, então, como fica um tratamento de interrupção de TMRO que gera uma base de tempo para piscar um LED. O resto da estrutura necessária é o mesmo apresentado anteriormente.

```

LED_ON      BSF    LED          ; ACENDE O LED
FIM_TMR0   MOVLW  .100
            MOVWF TEMPO1      ; REINICIA BASE DE TEMPO
            GOTO   SAI_INT     ; FINALIZA A INTERRUPÇÃO

```

A variável TEMPO1 foi utilizada como um contador auxiliar para aumentarmos a base de tempo contada. Desta forma, podemos considerar a seguinte tabela para podermos encontrar os valores corretos para uma base de tempo qualquer. Por exemplo, imaginemos que uma ação deva ocorrer a cada 1 s em um sistema rodando a 4 MHz (clico de máquina = 1μs):

Ciclo	Prescaler	TIMER	TEMPO1	Total (μs)
1μs	1	250	250	62.500
	2	250	250	125.000
	4	250	250	250.000
	8	250	250	500.000
	16	250	250	1.000.000
	32	125	250	1.000.000
	64	125	125	1.000.000

Como o ciclo de máquina é constante e os valores de Prescaler são limitados e conhecidos, basta jogar com os valores de TIMER e TEMPO1 para que a multiplicação de todas as colunas (Ciclo x Prescaler x TIMER x TEMPO1) dê o valor desejado na base de tempo (Total). Observe que podem existir várias combinações para o mesmo tempo. Você só não pode esquecer que TIMER e TEMPO1 são limitados ao valor máximo de 256 (8 bits). Por isso, bases de tempo pequenas podem nem precisar da variável TEMPO1, enquanto bases de tempo grandes podem precisar de outras variáveis de multiplicação (TEMPO2, TEMPO3, etc). Não se esqueça também de que a coluna TIMER indica o número de contagens feitas pelo Timer 0 antes dele estourar. Por isso, o TMRO deve ser inicializado com 256 - TIMER.

TRATANDO A INTERRUPÇÃO EXTERNA

A interrupção externa é uma das mais usadas. Podemos empregá-la em vias de recepção de dados, sinais externos importantíssimos que devem ser tratados rapidamente, vias de sincronismo e muito mais. Essa interrupção pode acontecer na borda de subida ou de descida do sinal. Para configurarmos qual das duas bordas deve gerar a interrupção, devemos acertar o valor do bit INTEDG do registrador OPTION. Quando esse bit está em 1, a interrupção acontecerá na subida do sinal e quando está em 0, na descida dele.

Para que essa interrupção possa ocorrer, sua chave individual INTCON,INTIE deve estar ligada (1). Dentro da rotina de tratamento devemos testar o flag INTCON,INTF, para sabermos se foi essa interrupção que ocorreu. Antes de sairmos do tratamento, devemos limpar esse mesmo flag manualmente.

TRATANDO A INTERRUPÇÃO DE MUDANÇA DE ESTADO

Essa interrupção deve ser usada em sinais que devam ser tratados tanto na subida como na descida. Um sincronismo com a rede elétrica é um bom exemplo de aplicação.

Para que essa interrupção possa ocorrer, sua chave individual INTCON,RBIE deve estar ligada (1). Dentro da rotina de tratamento devemos testar o flag INTCON,RBIF, para sabermos se foi essa interrupção que ocorreu. Antes de sairmos do tratamento, devemos limpar esse mesmo flag manualmente.

OUTRAS INTERRUPÇÕES

As demais interrupções, todas pertencentes ao grupo e periféricos, serão comentadas mais adiante, juntamente com o recurso em questão.

EXEMPLO 5 - TIMER SIMPLIFICADO

O timer simplificado foi colocado como exemplo para demonstrar o uso das interrupções. Neste caso, estamos utilizando apenas a interrupção de TMRO para gerarmos um contador de segundos. Desta forma, nosso timer terá seu valor decrementado a cada segundo, indo de um valor inicial (V_INICIO) até zero. Um botão é utilizado para disparar o timer e outro para paralisá-lo. Enquanto o timer estiver rodando, o LED permanece aceso.

A contagem de tempo é feita com base no prescaler (1:64), na inicialização do TMRO (256-125) e em um contador auxiliar (TEMP1) que é inicializado em 125. Desta forma, a interrupção acontecerá a cada 8 ms ($64 \text{ us} \times 125$). O contador auxiliar multiplicará este tempo por 125, resultando em 1 segundo ($8 \text{ ms} \times 125 = 1 \text{ s}$).

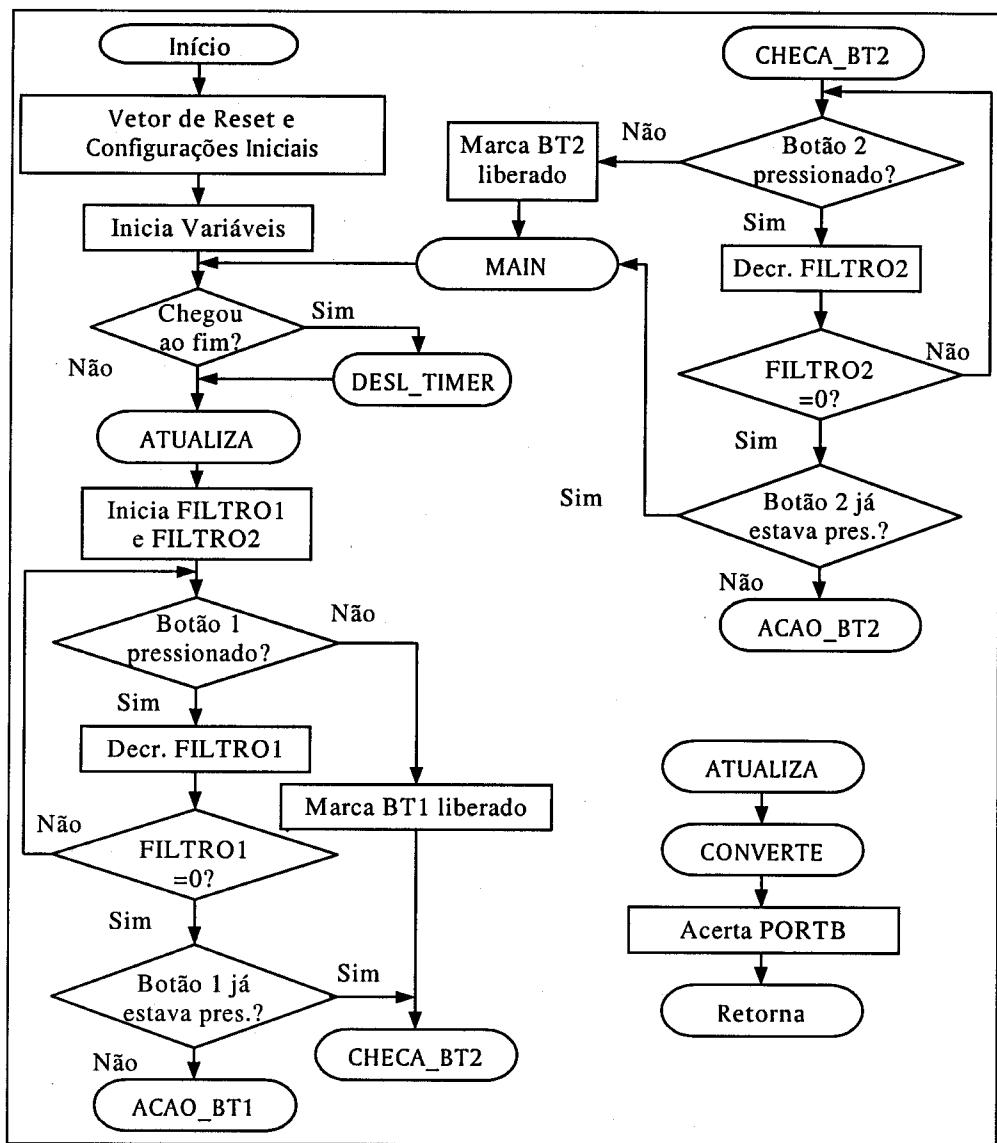


Figura 11.13 - Fluxograma do Exemplo 5 (Parte 1).

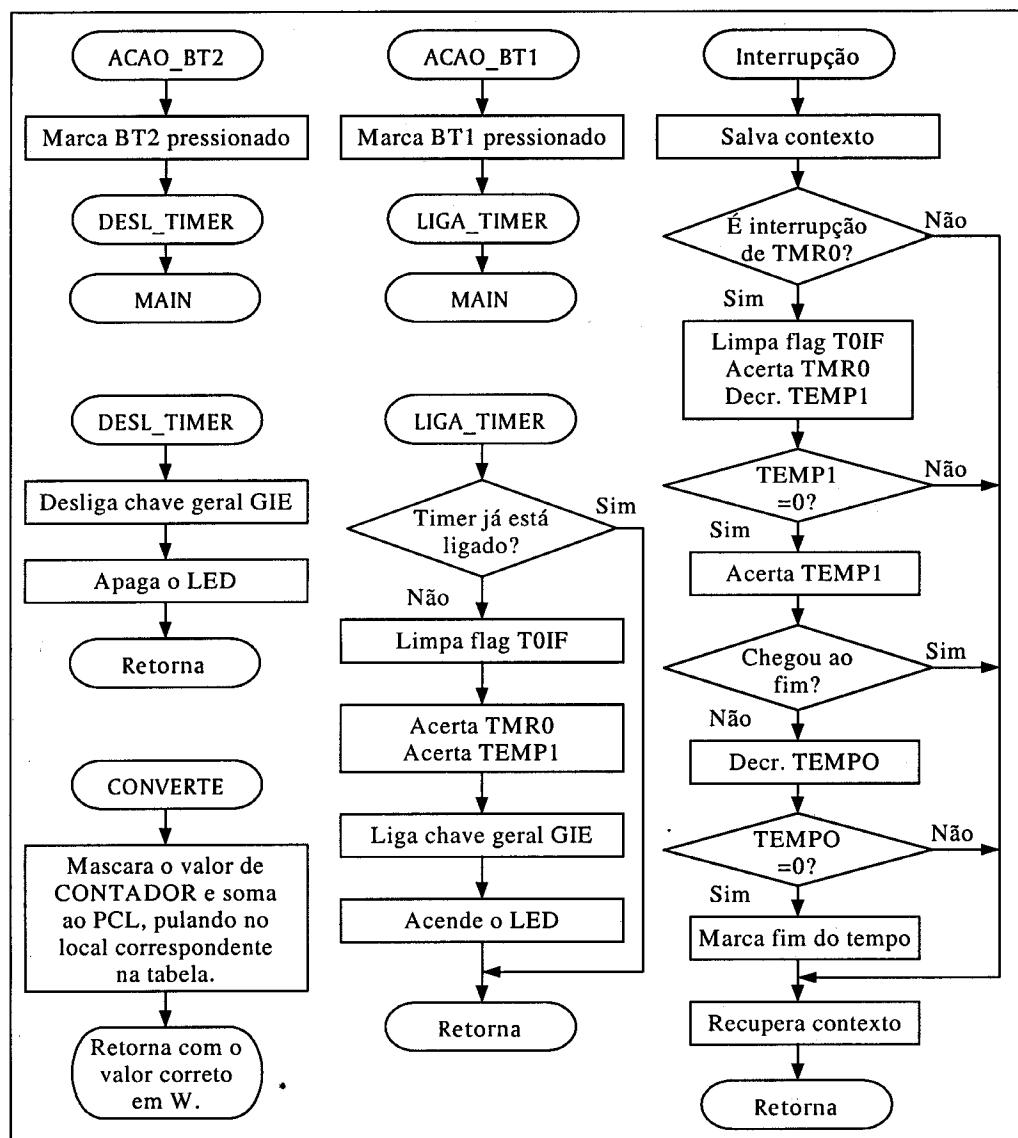


Figura 11.14 - Fluxograma do Exemplo 5 (Parte 2).

```

;* DISPLAY. O BOTÃO 2 PARALIZA O TIMER. O LED É UTILIZADO PARA *  

;* INDICAR O ESTADO ATUAL DO TIMER: ACESO=RODANDO E APAGADO=PARADO *  

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

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

;* ARQUIVOS DE DEFINIÇÕES  

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

;  

;#INCLUDE <P16F628A.INC> ;ARQUIVO PADRÃO MICROCHIP PARA 16F628A  

;_CONFIG _BODEN_ON & _CP_OFF & _PWRTE_ON & _WDT_OFF & _LVP_OFF &  

;_MCLRE_ON & _XT_OSC  

;  

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

;* PAGINAÇÃO DE MEMÓRIA  

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

;DEFINIÇÃO DE COMANDOS DE USUÁRIO PARA ALTERAÇÃO DA PÁGINA DE MEMÓRIA  

;  

#define BANK0 BCF STATUS,RP0 ;SETA BANK 0 DE MEMÓRIA  

#define BANK1 BSF STATUS,RP0 ;SETA BANK 1 DE MAMÓRIA  

;  

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

;* VARIÁVEIS  

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

;DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS  

;PELO SISTEMA  

;  

CBLOCK 0x20 ;ENDEREÇO INICIAL DA MEMÓRIA DE  

;USUÁRIO  

;  

W_TEMP ;REGISTRADORES TEMPORÁRIOS PARA  

STATUS_TEMP ;INTERRUPÇÕES  

TEMPO ;ARMAZENA O VALOR DO TEMPO  

FLAGS ;ARMAZENA OS FLAGS DE CONTROLE  

TEMP1 ;REGISTRADORES AUXILIARES  

TEMP2  

FILTRO1 ;FILTROS DOS BOTÕES  

FILTRO2  

;  

ENDC ;FIM DO BLOCO DE MEMÓRIA  

;  

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

;* FLAGS INTERNOS  

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

;DEFINIÇÃO DE TODOS OS FLAGS UTILIZADOS PELO SISTEMA  

;  

#define F_FIM FLAGS,0 ;FLAG DE FIM DE TEMPO  

#define ST_BT1 FLAGS,1 ;STATUS DO BOTÃO 1  

#define ST_BT2 FLAGS,2 ;STATUS DO BOTÃO 2  

;  

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

;* CONSTANTES  

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

;DEFINIÇÃO DE TODAS AS CONSTANTES UTILIZADAS PELO SISTEMA  

;  

V_INICIO EQU .15 ;VALOR INICIAL DO TIMER (1 A 15 SEG.)  

T_FILTRO EQU .255 ;VALOR DO FILTRO DOS BOTÕES

```



```
RETLW B'11101111' ; 08 - RETORNA SÍMBOLO CORRETO 8
RETLW B'01101111' ; 09 - RETORNA SÍMBOLO CORRETO 9
RETLW B'10101111' ; 10 - RETORNA SÍMBOLO CORRETO A
RETLW B'11100011' ; 11 - RETORNA SÍMBOLO CORRETO b
RETLW B'11000110' ; 12 - RETORNA SÍMBOLO CORRETO C
RETLW B'11101001' ; 13 - RETORNA SÍMBOLO CORRETO d
RETLW B'11000111' ; 14 - RETORNA SÍMBOLO CORRETO E
RETLW B'10000111' ; 15 - RETORNA SÍMBOLO CORRETO F

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* ROTINA DE ATUALIZAÇÃO DO DISPLAY *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; ESTA ROTINA CONVERTE O VALOR DE TEMPO ATRAVÉS DA ROTINA CONVERTE
; E ATUALIZA O PORTB PARA ACENDER O DISPLAY CORRETAMENTE

ATUALIZA
    CALL    CONVERTE      ;CONverte CONTADOR NO NÚMERO DO
                           ;DISPLAY
    MOVWF   PORTB         ;ATUALIZA O PORTB PARA
                           ;VISUALIZARMOS O VALOR DE CONTADOR
                           ;NO DISPLAY
    RETURN             ;NÃO, RETORNA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* ROTINA DE DESLIGAR O TIMER *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; ESTA ROTINA EXECUTA AS AÇÕES NECESSÁRIAS PARA DESLIGAR O TIMER

DESL_TIMER
    BCF    INTCON,GIE      ;DESLIGA CHAVE GERAL DE INT.
    BCF    LED              ;APAGA O LED
    RETURN           ;RETORNA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* ROTINA DE LIGAR O TIMER *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; ESTA ROTINA EXECUTA AS AÇÕES NECESSÁRIAS PARA LIGAR O TIMER

LIGA_TIMER
    BTFSC  INTCON,GIE      ;TIMER JÁ ESTA LIGADO?
    RETURN           ;SIM, RETORNA DIRETO
                   ;NÃO
    BCF    INTCON,T0IF      ;LIMPA FLAG DE INT. DE TMR0
    MOVLW  .256-.125
    MOVWF   TMR0            ;INICIA TMR0 CORRETAMENTE
    MOVLW  .125
    MOVWF   TEMP1           ;INICIA TEMP1 CORRETAMENTE
    BSF    INTCON,GIE      ;LIGA CHAVE GERAL DE INTERRUPÇÕES
    BSF    LED              ;ACENDE O LED
    RETURN           ;RETORNA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* INICIO DO PROGRAMA *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

INICIO
    BANK1            ;ALTERA PARA O BANCO 1
```



```
        ; SIM
DECFSZ FILTRO2,F    ; DECREMENTA O FILTRO DO BOTÃO
                     ; TERMINOU?
                     ; NÃO, CONTINUA ESPERANDO
                     ; SIM
GOTO   CHECA_BT2    ; BOTÃO JÁ ESTAVA PRESSIONADO?
                     ; NÃO, EXECUTA AÇÃO DO BOTÃO
                     ; SIM, VOLTA AO LOOPING

BT2_LIB
BCF    ST_BT2       ; MARCA BOTÃO 2 COMO LIBERADO
GOTO   MAIN          ; RETORNA AO LOOPING

ACAO_BT1
BSF    ST_BT1       ; AÇÃO PARA O BOTÃO 1
CALL   LIGA_TIMER   ; MARCA BOTÃO 1 COMO JÁ PRESSIONADO
                     ; LIGA O TIMER
GOTO   MAIN          ; NÃO, VOLTA AO LOOP PRINCIPAL

ACAO_BT2
BSF    ST_BT2       ; AÇÃO PARA O BOTÃO 2
CALL   DESL_TIMER   ; MARCA BOTÃO 2 COMO JÁ PRESSIONADO
                     ; DESLIGA O TIMER
GOTO   MAIN          ; NÃO, VOLTA AO LOOP PRINCIPAL

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*                               FIM DO PROGRAMA                         *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

END ; OBRIGATÓRIO

Trabalhe agora com os exercícios propostos:

1. Altere a base de tempo do timer de segundos para minutos.
2. Para não parecer que o timer está parado durante o transcorrer de um minuto, faça um LED ficar piscando para indicar os segundos.
3. Altere a lógica de timer regressivo para progressivo.

UTILIZANDO A EEPROM

Uma das grandes vantagens de utilizarmos o PIC16F628A é o fato de que ele possui uma EEPROM interna com 128 bytes. Como já dissemos, essa memória não é volátil e com ela podemos guardar dados, mesmo quando o sistema fica sem alimentação. Isso é muito utilizado para sistemas que precisam manter dados ou configurações programadas pelo usuário. Uma discadora telefônica é um bom exemplo disso, pois o usuário só precisa programar os números que serão chamados uma única vez. Muitas vezes, uma memória EEPROM externa é necessária, principalmente quando os 128 bytes disponíveis não são suficientes. Neste caso, porém, além das rotinas de escrita e leitura serem mais complexas, são necessários também pelo menos dois pinos para interligar a memória ao microcontrolador. Por isso, já que o PIC16F628A

possui a EEPROM interna, estudaremos somente a comunicação com ela, deixando as memórias externas para uma futura oportunidade.

ESCREVENDO NA EEPROM

A primeira ação que faremos em relação à EEPROM é a operação de escrita. Na verdade, ela é ligeiramente mais complexa que a operação de leitura, mas cronologicamente é mais interessante aprendermos primeiro a escrever. Afinal, como poderemos ler alguma coisa se não escrevermos nada?

A complexibilidade da escrita é devido ao sistema de proteção que o PIC possui, para evitar escritas acidentais na memória. Por isso, a inicialização da escrita parecerá um pouco confusa e desnecessária, mas isso torna o sistema muito robusto e seguro.

A escrita da EEPROM deve acompanhar o seguinte roteiro:

1. O endereço para a escrita deve ser colocado em **EEADR** (banco 1). Como existem 128 bytes disponíveis, esse endereço deve estar entre 0 e 64.
2. O dado a ser escrito deve ser colocado em **EEDATA** (banco 1). Só podemos escrever um byte de cada vez.
3. As interrupções devem ser desligadas para evitarmos conflitos.
4. A escrita deve ser habilitada por meio do bit **EECON1,WREN** (1) (banco 1).
5. O registrador **EECON2** (banco 1) deve ser carregado com os valores **0x55** e **0xAA**, seqüencialmente. Este procedimento é obrigatório e utilizado para a proteção da escrita.
6. A escrita deve ser iniciada setando o bit **EECON1,WR** (1) e limpando o bit **EECON1,WREN** (0).
7. As interrupções podem ser novamente ligadas.
8. A operação de escrita é um pouco demorada, e ela só terá terminado quando o bit **EECON1,WR** tiver sido limpo automaticamente pelo hardware. Por isso, normalmente ficamos esperando que isso aconteça. No caso de não podermos ficar esperando pelo fim da escrita, podemos ligar a interrupção de escrita na EEPROM por meio do bit **INTCON,EEIE**, e esperarmos que ela aconteça para considerarmos finalizada a escrita.
9. Caso algum erro ocorra durante a operação de escrita, o bit **EECON1,WRERR** será setado (1). No caso de sucesso na operação, esse bit será mantido em zero (0).

Veja então como fica uma rotina completa de escrita na EEPROM sem uso da interrupção:

```
*****  
;* ROTINA DE ESCRITA NA E2PROM *  
;* *****  
;O ENDEREÇO ONDE QUEREMOS ESCREVER DEVE SER INICIALMENTE COLOCADO EM  
;EEADR ANTES DE CHAMAR ESTA ROTINA.  
;O DADO A SER ESCRITO DEVE SER COLOCADO EM W ANTES DE CHAMAR A ROTINA  
  
ESCR_E2PROM  
    BANK1 ;BANCO 1  
    MOVWF EEDATA ;ACERTA DADO PASSADO EM W PARA EEDATA  
    BCF INTCON, GIE ;DESABILITA INTERRUPÇÕES  
    BSF EECON1, WREN ;HABILITA ESCRITA  
    MOVLW 0X55 ;INICIALIZAÇÃO DA ESCRITA  
    MOVWF EECON2 ;(PROCEDIMENTO OBRIGATÓRIO)  
    MOVLW 0XAA  
    MOVWF EECON2  
    BSF EECON1, WR ;INICIA ESCRITA  
    BCF EECON1, WREN  
    BTFSC EECON1, WRERR ;HOUVE ERRO?  
    GOTO ESCR_E2PROM+2 ;SIM, TENTA NOVAMENTE SEM AFETAR EEDATA  
    BTFSC EECON1, WR ;NÃO, ACABOU ESCRITA?  
    GOTO $-3 ;NÃO, CONTINUA AGUARDANDO  
    BANK0 ;SIM, BANCO 0  
    BSF INTCON, GIE ;HABILITA INTERRUPÇÕES NOVAMENTE  
    RETURN ;RETORNA
```

Esta rotina não é perfeita, pois, se ocorrer um erro, o sistema tenta escrever o dado novamente, mas não limita a quantidade de tentativas. Por isso, se um erro sério ocorrer e a memória não conseguir mais ser gravada, o sistema ficará travado. O certo talvez fosse informar ao usuário por meio de um código de erro.

TRATANDO A INTERRUPÇÃO DE FINAL DE ESCRITA DA EEPROM

Essa interrupção deve ser usada em sistemas que não podem ficar parados esperando a operação de escrita terminar. Por isso, tão logo a operação de escrita seja iniciada, a rotina deve ser finalizada (não deve possuir o teste do bit EECON,WR, como no exemplo anterior). Quando a escrita for terminada, uma interrupção irá ocorrer e o sistema poderá tomar as ações pertinentes.

Para que essa interrupção possa ocorrer, sua chave individual INTCON,EEIE deve estar ligada (1). Dentro da rotina de tratamento devemos testar o flag EECON1,EEIF, para sabermos se foi essa interrupção que ocorreu. Antes de sairmos do tratamento, devemos limpar esse mesmo flag manualmente.

LENDÔ DA EEPROM

Agora que você já sabe escrever alguma coisa nos endereços disponíveis da EEPROM, poderemos ler essa informação de volta por intermédio de uma rotina de leitura. A leitura é muito mais simples, pois não necessita de tanta proteção, e também muito mais rápida.

O roteiro para a criação da rotina de leitura é o seguinte:

1. O endereço para a leitura deve ser colocado em EEADR (banco 1). Como existem 128 bytes disponíveis, este endereço deve estar entre 0 e 127.
 2. A leitura deve ser ligada por meio do bit EECON1.RD (1). A leitura terminará quando este bit voltar automaticamente para zero, o que acontece quase que imediatamente.
 3. O dado lido será colocado em EEDATA (banco 1).

A rotina de exemplo para a leitura é bem simples:

EXEMPLO 6 - CONTADOR FINAL

Esta é a versão final do contador iniciado no Exemplo 2. Possui todos os recursos do Exemplo 4 e mais o armazenamento do valor do contador na memória não volátil (EEPROM), para que esse dado não seja perdido mesmo no caso de desligamento da alimentação.

Observe que a EEPROM é inicializada por intermédio da diretriz DE.

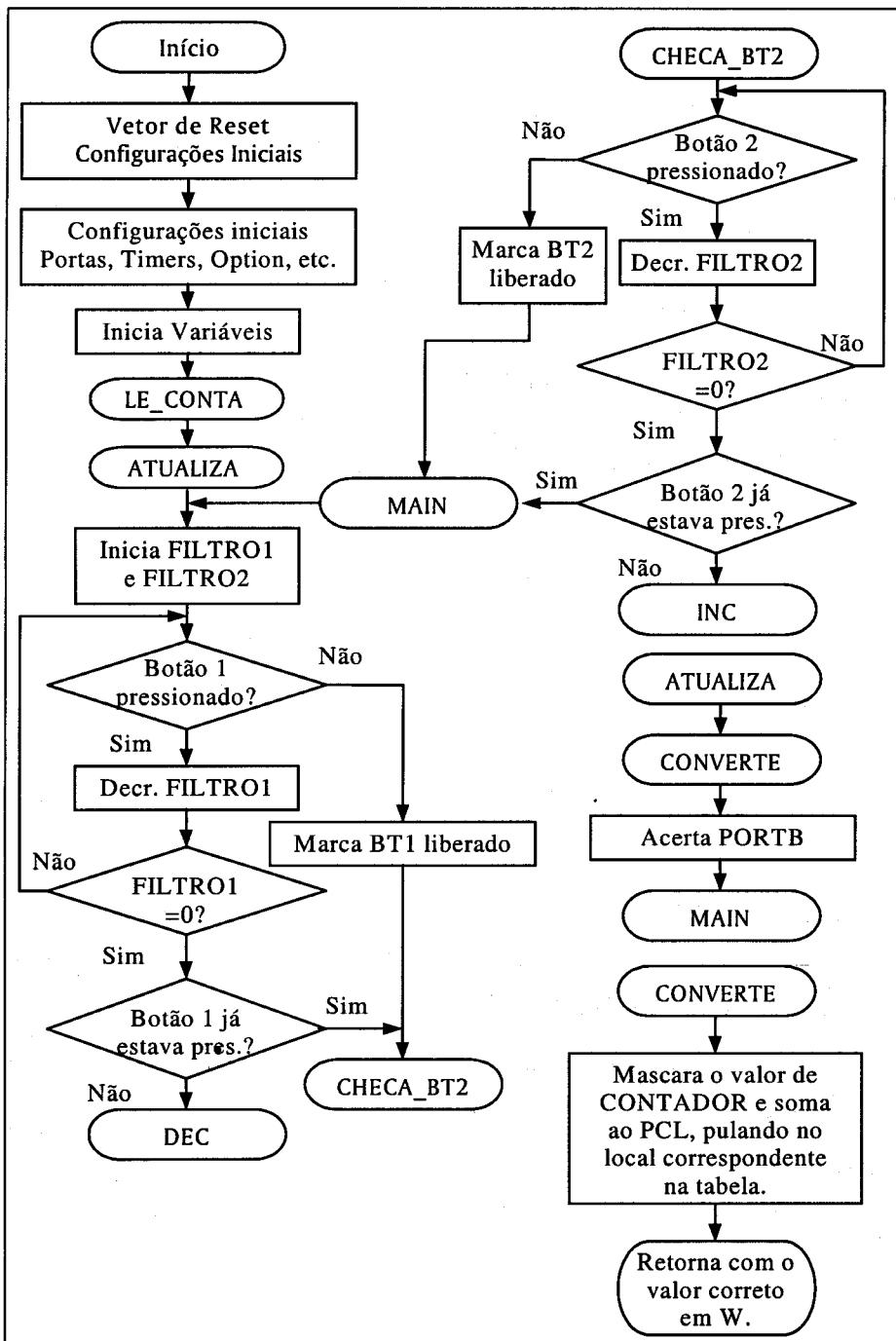


Figura 11.15 - Fluxograma do Exemplo 6 (Parte 1).

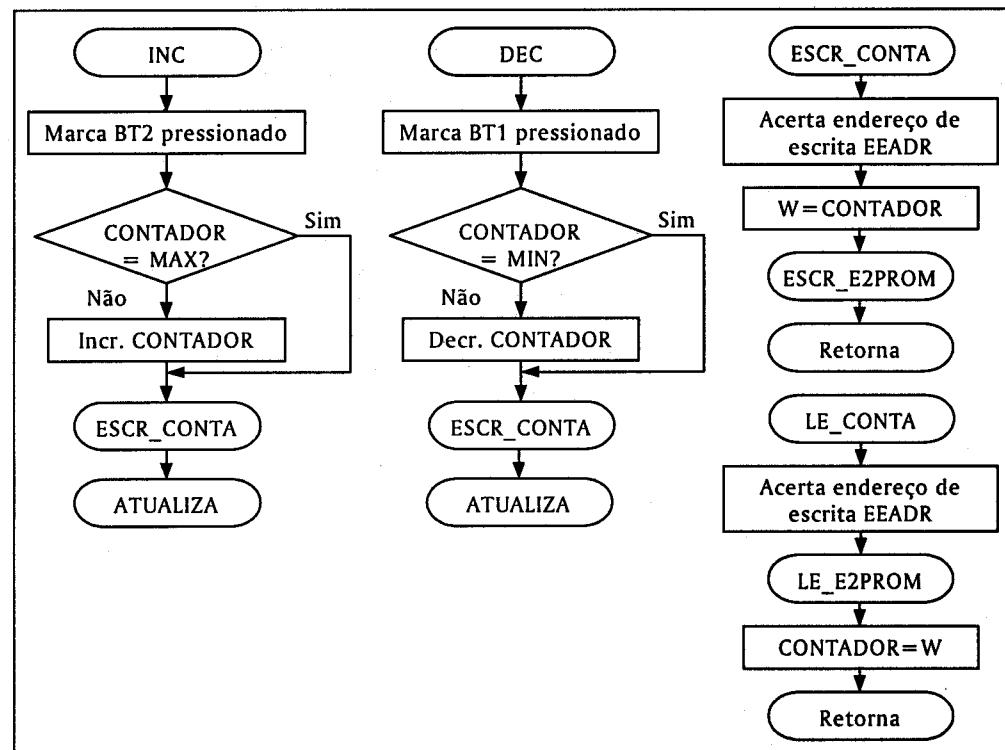


Figura 11.16 - Fluxograma do Exemplo 6 (Parte 2).

```
*****  
;*                      CONTADOR FINAL - EX6  
;*                      DESBRAVANDO O PIC  
;*          DESENVOLVIDO PELA MOSAICO ENGENHARIA E CONSULTORIA  
;*          VERSÃO: 1.0          DATA: 17/06/03  
;*          DESCRIÇÃO DO ARQUIVO  
;  
;*          CONTADOR QUE UTILIZA DOIS BOTÕES PARA INCREMENTAR E DECRE-  
;*          MENTAR O VALOR CONTROLADO PELA VARIÁVEL "CONTADOR". ESTA  
;*          VARIÁVEL ESTÁ LIMITADA PELAS CONSTANTES "MIN" E "MAX".  
;*          O VALOR DO CONTADOR É MOSTRADO NO DISPLAY E ARMAZENADO NA  
;*          EEPROM PARA NÃO SER PERDIDO MESMO NO CASO DE RESET.  
;  
;*          ARQUIVOS DE DEFINIÇÕES  
;  
#INCLUDE <P16F628A.INC>      ;ARQUIVO PADRÃO MICROCHIP PARA 16F628A  
    __CONFIG _BODEN_ON & _CP_OFF & _PWRTE_ON & _WDT_OFF & _LVP_OFF &  
    _MCLRE_ON & _XT_OSC  
;  
;*          PAGINAÇÃO DE MEMÓRIA  
;  
;*          DEFINIÇÃO DE COMANDOS DE USUÁRIO PARA ALTERAÇÃO DA PÁGINA DE MEMÓRIA
```

```

#define      BANK0  BCF STATUS,RPO ;SETA BANK 0 DE MEMÓRIA
#define      BANK1  BSF STATUS,RPO ;SETA BANK 1 DE MAMÓRIA

;***** VARIÁVEIS *****
; DEFINIÇÃO DOS NOMES E ENDEREÇOS DE TODAS AS VARIÁVEIS UTILIZADAS
; PELO SISTEMA

CBLOCK 0x20           ;ENDEREÇO INICIAL DA MEMÓRIA DE
                      ;USUÁRIO
CONTADOR    ;ARMAZENA O VALOR DA CONTAGEM
FLAGS        ;ARMAZENA OS FLAGS DE CONTROLE
FILTRO1     ;FILTRAGEM PARA O BOTÃO 1
FILTRO2     ;FILTRAGEM PARA O BOTÃO 2
ENDC         ;FIM DO BLOCO DE MEMÓRIA

;***** FLAGS INTERNOS *****
; DEFINIÇÃO DE TODOS OS FLAGS UTILIZADOS PELO SISTEMA

#define      ST_BT1  FLAGS,0      ;STATUS DO BOTÃO 1
#define      ST_BT2  FLAGS,1      ;STATUS DO BOTÃO 2

;***** CONSTANTES *****
; DEFINIÇÃO DE TODAS AS CONSTANTES UTILIZADAS PELO SISTEMA

MIN          EQU    .0      ;VALOR MÍNIMO PARA O CONTADOR
MAX          EQU    .15     ;VALOR MÁXIMO PARA O CONTADOR
T_FILTRO    EQU    .255    ;FILTRO PARA BOTÃO
POS_MEM      EQU    .0      ;ENDEREÇO DA EEPROM ONDE SERÁ
                           ;ARMAZENADO O VALOR DO CONTADOR

;***** ENTRADAS *****
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO ENTRADA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)

#define      BOTAO1  PORTA,1    ;PORTA DO BOTÃO
                           ; 0 -> PRESSIONADO
                           ; 1 -> LIBERADO
#define      BOTAO2  PORTA,2    ;PORTA DO BOTÃO
                           ; 0 -> PRESSIONADO
                           ; 1 -> LIBERADO

;***** SAÍDAS *****
; DEFINIÇÃO DE TODOS OS PINOS QUE SERÃO UTILIZADOS COMO SAÍDA
; RECOMENDAMOS TAMBÉM COMENTAR O SIGNIFICADO DE SEUS ESTADOS (0 E 1)

;***** INICIALIZAÇÃO DA EEPROM *****
; INÍCIO DA EEPROM

ORG      H'2100'+POS_MEM

```

DE .5 ; VALOR INICIAL PARA CONTADOR = 5
; VIDE DIRETRIZ "DE" NO APÊNDICE B

;*****
;* VETOR DE RESET
;*****

ORG 0x00 ;ENDEREÇO INICIAL DE PROCESSAMENTO
GOTO INICIO

;*****
;* INÍCIO DA INTERRUPÇÃO
;*****
; AS INTERRUPÇÕES NÃO SERÃO UTILIZADAS, POR ISSO PODEMOS SUBSTITUIR
; TODO O SISTEMA EXISTENTE NO ARQUIVO MODELO PELO APRESENTADO ABAIXO
; ESTE SISTEMA NÃO É OBRIGATÓRIO, MAS PODE EVITAR PROBLEMAS FUTUROS

ORG 0x04 ;ENDEREÇO INICIAL DA INTERRUPÇÃO
RETFIE ;RETORNA DA INTERRUPÇÃO

;*****
;* ROTINA DE CONVERSÃO BINÁRIO -> DISPLAY
;*****
; ESTA ROTINA IRÁ RETORNAR EM W, O SÍMBOLO CORRETO QUE DEVE SER
; MOSTRADO NO DISPLAY PARA CADA VALOR DE CONTADOR. O RETORNO JÁ ESTÁ
; FORMATADO PARA AS CONDIÇÕES DE LIGAÇÃO DO DISPLAY AO PORTB.

```

;      a
;*****
;      *
;      * b
;      *
;      g   *
;*****
;      *
;      *
;      e   *      c
;      * d   *
;*****
;
```

CONVERTE

MOVF	CONTADOR,W	; COLOCA CONTADOR EM W
ANDLW	B'00001111'	; MASCARA VALOR DE CONTADOR ; CONSIDERAR SOMENTE ATÉ 15
ADDWF	PCL,F	
;		
; EDC.BAFG' ; POSIÇÃO CORRETA DOS SEGUIMENTOS		
RETLW	B'11101110'	; 00 - RETORNA SÍMBOLO CORRETO 0
RETLW	B'00101000'	; 01 - RETORNA SÍMBOLO CORRETO 1
RETLW	B'11001101'	; 02 - RETORNA SÍMBOLO CORRETO 2
RETLW	B'01101101'	; 03 - RETORNA SÍMBOLO CORRETO 3
RETLW	B'00101011'	; 04 - RETORNA SÍMBOLO CORRETO 4
RETLW	B'01100111'	; 05 - RETORNA SÍMBOLO CORRETO 5
RETLW	B'11100111'	; 06 - RETORNA SÍMBOLO CORRETO 6
RETLW	B'00101100'	; 07 - RETORNA SÍMBOLO CORRETO 7
RETLW	B'11101111'	; 08 - RETORNA SÍMBOLO CORRETO 8
RETLW	B'01101111'	; 09 - RETORNA SÍMBOLO CORRETO 9
RETLW	B'10101111'	; 10 - RETORNA SÍMBOLO CORRETO A
RETLW	B'11100011'	; 11 - RETORNA SÍMBOLO CORRETO b
RETLW	B'11000110'	; 12 - RETORNA SÍMBOLO CORRETO C
RETLW	B'11101001'	; 13 - RETORNA SÍMBOLO CORRETO d
RETLW	B'11000111'	; 14 - RETORNA SÍMBOLO CORRETO E
RETLW	B'10000111'	; 15 - RETORNA SÍMBOLO CORRETO F

```

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* ROTINA DE LEITURA NA E2PROM * *
;* ESTA ROTINA LÊ O BYTE DO ENDEREÇO ACERTADO POR EEADR E COLOCA
; O VALOR EM W.
;

LE_E2PROM
    BANK1      ;BANCO 1
    BSF        EECON1, RD   ;PREPARA LEITURA
    MOVF       EEDATA, W   ;COLOCA DADO EM W
    BANK0
    RETURN     ;RETORNA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* ROTINA DE ESCRITA NA E2PROM * *
;* ESTA ROTINA ESCREVE O DADO PASSADO EM W NO ENDEREÇO ACERTADO
; ANTERIORMENTE EM EEADR
;

ESCR_E2PROM
    BANK1      ;BANCO 1
    MOVWF     EEDATA      ;ACERTA DADO PASSADO EM W
    BCF       INTCON, GIE  ;DESABILITA INTERRUPÇÕES
    BSF       EECON1, WREN  ;HABILITA ESCRITA
    MOVLW    0X55          ;INICIALIZAÇÃO DA ESCRITA
    MOVWF     EECON2
    MOVLW    0XAA
    MOVWF     EECON2
    BSF       EECON1, WR   ;INICIA ESCRITA
    BCF       EECON1, WREN  ;ACABOU ESCRITA?
    BTFSR    EECON1, WR
    GOTO     $-1           ;NÃO, AGUARDA
    BANK0
    BSF       INTCON, GIE  ;SIM, BANCO 0
    RETURN     ;HABILITA INTERRUPÇÕES
    ;RETORNA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* ROTINA DE LEITURA DO VALOR DO CONTADOR * *
;* ESTA ROTINA LÊ O VALOR DA MEMÓRIA E COLOCA O RESULTADO NA
; VARIÁVEL "CONTADOR".
;

LE_CONTA
    MOVLW    POS_MEM
    BANK1
    MOVWF    EEADR        ;ACERTA O ENDEREÇO PARA LEITURA
    CALL     LE_E2PROM    ;EFETUA A LEITURA DA EEPROM
    MOVF     CONTADOR     ;ATUALIZA O CONTADOR
    RETURN     ;RETORNA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* ROTINA DE ESCRITA DO VALOR DO CONTADOR * *
;* ESTA ROTINA ESCREVE O VALOR ATUAL DE CONTADOR NA MEMÓRIA EEPROM
;

ESCR_CONTA
    MOVLW    POS_MEM
    BANK1
    MOVWF    EEADR        ;ACERTA O ENDEREÇO PARA LEITURA
    BANK0
    MOVF     CONTADOR, W  ;COLOCA CONTADOR EM W

```

```

CALL ESCR_E2PROM ;EFETUA A ESCRITA EEPROM
RETURN ;RETORNA

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*
      INICIO DO PROGRAMA
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

INICIO
    BANK1          ;ALTERA PARA O BANCO 1
    MOVLW B'00000110'
    MOVWF TRISA    ;DEFINE RA1 E 2 COMO ENTRADA E DEMAIS
                    ;COMO SAÍDAS
    MOVLW B'00000000'
    MOVWF TRISB    ;DEFINE TODO O PORTB COMO SAÍDA
    MOVLW B'10000000'
    MOVWF OPTION_REG ;PRESCALER 1:2 NO TMRO
                    ;PULL-UPS DESABILITADOS
                    ;AS DEMAIS CFG. SÃO IRRELEVANTES
    MOVLW B'00000000'
    MOVWF INTCON   ;TODAS AS INTERRUPÇÕES DESLIGADAS
    BANK0          ;RETORNA PARA O BANCO 0
    MOVLW B'00000111'
    MOVWF CMCON    ;DEFINE O MODO DE OPERAÇÃO DO COMPARADOR

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           INICIALIZAÇÃO DAS VARIÁVEIS
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

CLRFB PORTA        ;LIMPA O PORTA
CLRF PORTB         ;LIMPA O PORTB
CLRF FLAGS         ;LIMPA TODOS OS FLAGS
CALL LE_CONTA      ;INICIALIZA CONTADOR COM VALOR
                    ;DA EEPROM
GOTO ATUALIZA      ;ATUALIZA O DISPLAY INICIALMENTE

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           ROTINA PRINCIPAL
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

MAIN
    MOVLW T_FILTRO
    MOVWF FILTRO1    ;INICIALIZA FILTRO1 = T_FILTRO
    MOVWF FILTRO2    ;INICIALIZA FILTRO2 = T_FILTRO

CHECA_BT1
    BTFSC BOTAO1     ;O BOTÃO 1 ESTÁ PRESSIONADO?
    GOTO BT1_LIB     ;NÃO, ENTÃO TRATA COMO LIBERADO
                    ;SIM
    DECFSZ FILTRO1,F ;DECREMENTA O FILTRO DO BOTÃO
                    ;TERMINOU?
    GOTO CHECA_BT1   ;NÃO, CONTINUA ESPERANDO
                    ;SIM
    BTFSS ST_BT1     ;BOTÃO JÁ ESTAVA PRESSIONADO?
    GOTO DEC         ;NÃO, EXECUTA AÇÃO DO BOTÃO
    GOTO CHECA_BT2   ;SIM, CHECA BOTÃO 2

BT1_LIB
    BCF ST_BT1       ;MARCA BOTÃO 1 COMO LIBERADO

CHECA_BT2
    BTFSC BOTAO2     ;O BOTÃO 2 ESTÁ PRESSIONADO?
    GOTO BT2_LIB     ;NÃO, ENTÃO TRATA COMO LIBERADO

```

Os exercícios propostos para este exemplo são:

1. Altere o valor inicial do contador (inicialização da EEPROM).

2. Implemente um terceiro botão para efetuar a gravação do valor de CONTADOR na EEPROM. Atualmente este valor é salvo toda vez que é alterado.
3. Volte ao exemplo inicial e implemente o terceiro botão para gravar o valor de CONTADOR em outra posição da EEPROM. Implemente o quarto botão para poder ler este valor. Assim, serão salvos dois valores em duas posições diferentes: o primeiro toda vez que contador é alterado e o segundo quando o botão é pressionado.

ACESSO INDIRETO À MEMÓRIA

Até aqui acessamos várias vezes a memória do PIC, mas todas elas foram de forma direta, isto é, passando diretamente o endereço desejado como argumento de alguma instrução. Mas existe uma outra maneira de acessarmos a memória, por meio de registradores auxiliares, tornando o acesso indireto. O conceito é um pouco estranho, mas com o tempo você descobrirá que esse sistema é muito poderoso.

TRABALHANDO COM FSR E INDF

O acesso indireto utiliza dois SFRs que são denominados FSR e INDF. O primeiro refere-se ao endereço que desejamos acessar e o segundo ao valor propriamente dito, que pode ser lido ou escrito. Se você já trabalhou com alguma linguagem de programação que possui ponteiros, como o C++, vai reconhecer logo as semelhanças. Mas vejamos um exemplo prático: a função do código seguinte será iniciar um bloco de X bytes da RAM com o valor 0 (zero). O bloco terá seu endereço inicial definido por INICIO_BL e o final por FIM_BL.

INICO_BL	EQU	0X20	; FÍNICO DO BLOCO NA RAM
FIM_BL	EQU	0X2F	; FIM DO BLOCO NA RAM
LIMPA_BLOCO			
MOVLW	INICIO_BL		; ROTINA DE LIMPEZA DA RAM
MOVWF	FSR		; INICIA O PONTEIRO PARA O COMEÇO DO BLOCO
LB_1	CLRF	INDF	; NA VERDADE, ESTAMOS LIMPANDO O ENDEREÇO EXISTENTE ; NO REGISTRADOR FSR
	MOVLW	FIM_BL	
	XORWF	FSR,W	; COMPARA O PONTEIRO COM O FIM DO BLOCO
	BTFSC	STATUS,Z	; JÁ LIMPOU O ÚLTIMO ENDEREÇO?
	RETURN		; SIM, FINALIZA A ROTINA
	INCFL	FSR,F	; NÃO, INCREMENTA O PONTEIRO PARA ACESSAR O ; PRÓXIMO BYTE
	GOTO	LB_1	; VOLTA AO LOOPING

Existem muitas outras aplicações para o acesso indireto, mas nada melhor que o tempo e o uso para que você realmente as aprenda.

TRABALHANDO COM WATCHDOG (WDT) (CRLWDT)

O Watchdog Timer (WDT), popularmente conhecido como "cão de guarda", também é um contador incrementado automaticamente, só que com um clock independente. O PIC possui um RC interno só para a operação do WDT. Isso significa que seu tempo de incremento é constante, independentemente do oscilador utilizado para gerar o clock da máquina. O tempo normal de estouro do WDT é cerca de 18 ms, mas pode variar de acordo com a tensão de alimentação e a temperatura. Outra característica muito importante desse contador é que ele não é acessível ao programador, nem para a escrita e nem para a leitura. O programador só pode utilizar um comando (CLRWDT) para zerá-lo. E isto é muito importante, pois se o WDT estourar (passar de 0xFF para 0x00), o sistema será automaticamente resetado, tornando a utilização do WDT muito importante para evitarmos que o sistema travem em determinadas situações. É importante lembrar também que o WDT pode ser desligado, como uma escolha de configuração na hora de gravar o microcontrolador. Isso não permite, porém, que o WDT seja ligado ou desligado durante a execução do programa.

O WDT é um recurso poderosíssimo que deve ser sempre utilizado. Por isso, é bom você se acostumar a utilizá-lo em todos os seus sistemas desde o começo. Já falamos sobre o WDT diversas vezes até agora, mas é muito importante nos aprofundarmos em seus conceitos e funções.

Trata-se, então, de um contador automático incrementado por meio de um oscilador próprio, independente do oscilador principal do sistema. Com o prescaler de 1:1, seu tempo de estouro, ou seja, 256 incrementos, é equivalente a 18 ms. O prescaler é utilizado para aumentarmos esse tempo. Caso o WDT estoure, um reset do sistema irá ocorrer imediatamente. Essa situação pode ser checada, analisando o bit /TO do registrador STATUS.

UTILIZANDO O WDT PARA EVITAR TRAVAMENTOS

A principal função do WDT em sistemas profissionais é evitar que eles travem por um problema no software. Desta forma, nosso sistema deve executar normalmente a instrução CLRWDT, numa freqüência mais rápida que a do estouro do contador. Como essa instrução limpa o contador, este jamais estourará. Caso o programa se perca, ou trave em um ponto estranho, a instrução CLRWDT não será mais executada e o WDT estourará, gerando um reset forçado e saindo do travamento. Simples, não é? Nem tanto.

Acontece que, para esse conceito realmente funcionar, o ideal é que só exista uma instrução CLRWDT em todo o programa. Por isso, é muito comum perder muito tempo escolhendo o ponto ideal em que essa instrução deva ser colocada. O normal é ela ficar dentro do looping principal do sistema. Acontece que, como o prescaler não pode ser utilizado com o WDT e o TMRO ao mesmo tempo, às vezes ficamos limitados a limpar o WDT a cada 18 ms, e nem sempre a nossa rotina principal leva

menos tempo que isso. Não é proibido introduzirmos mais instruções CLRWDT em outros locais do programa (muitas vezes esta é a única solução), mas lembre-se de que se seu sistema possuir essa instrução por todo lado, ele nunca destravará por meio do estouro do WDT.

Lembramos ainda que a limpeza do WDT não deve ser executada dentro de interrupções de timer, ou interrupções externas que aconteçam periodicamente. Normalmente, essa parece uma solução simples e rápida, mas lembre-se que mesmo com o sistema travado em alguns pontos, as interrupções podem continuar acontecendo.

UTILIZANDO O WDT PARA MELHORAR A INICIALIZAÇÃO

Uma outra função para o WDT, que nós também recomendamos, é a melhoria da inicialização do sistema após reset, por meio de um tempo de parada para estabilização. O conceito aqui é o seguinte: logo no começo do programa, checamos se o reset foi ocasionado por um estouro de WDT. Se não foi (no caso de energização do sistema), então ficamos parados em um looping infinito, esperando que o WDT estoure. Quando isso acontecer, o sistema será iniciado novamente, pulando esse looping infinito. Com isso, conseguimos que o programa demore um tempo entre a energização do PIC e o início do processamento real. Isso ajuda muito na estabilização de todo o hardware, evitando muitos problemas de inicialização. O tempo de atraso pode ser regulado por intermédio do prescaler.

```
; COLOQUE SUAS VARIÁVEIS AQUI
GOTO    $          ;LOOP INFINITO ESPERANDO ESTOURO DE WDT
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           INCIALIZAÇÃO DAS VARIÁVEIS LOCAIS           *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
; INICIALIZAÇÃO DE TODAS AS VARIÁVEIS QUE DEVEM SER ACERTADAS QUANDO
; O PIC É RESETADO PELO WDT, A FIM DE TORNAR ISTO O MAIS TRANSPARENTE
; POSSÍVEL PARA O USUÁRIO DO SISTEMA

VAR_LOCAIS
; COLOQUE SUAS VARIÁVEIS AQUI
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;*           ROTINA PRINCIPAL           *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
MAIN
; COLOQUE SEU PROGRAMA AQUI
GOTO    MAIN
```

No caso deste exemplo, podemos observar ainda um outro recurso implementado e bastante interessante. Trata-se da criação de duas seções para inicialização de variáveis. A primeira, que neste caso chamamos de VAR_GERAIS, é processada quando o PIC é alimentado ou resetado pelo hardware. Nessa seção, devem ser colocadas todas as variáveis que precisam ser acertadas quando o programa é iniciado. Já na segunda seção, que denominamos VAR_LOCAIS, podemos colocar as variáveis que devem ser modificadas, caso o sistema trave e force um estouro de WDT. Isso é utilizado para que, quando o sistema resete por WDT, o fato passe despercebido ao usuário final.

Podemos utilizar esse mesmo conceito para elaborar um sistema ainda mais eficiente em matéria de destravamento. Imaginemos que, em todas as rotinas do sistema, um registrador é atualizado com um índice que indique a região do programa que está sendo executada naquele momento. Caso o sistema trave e seja resetado por meio do WDT, basta não alterarmos o valor desse registrador, consultando-o e pulando para a mesma região do programa que estava sendo executada. Esse sistema é extremamente eficiente para evitarmos travamentos e surpresas aos usuários. No entanto, não é tão simples quanto parece, pois todas as demais variáveis do sistema devem ser acertadas para evitarmos outros problemas na lógica.

UTILIZANDO O WDT PARA AJUDAR NA SOLUÇÃO DE PROBLEMAS

O watchdog também pode ser de alguma utilidade na debugação do sistema, caso você não possua um emulador. A mesma sugestão dada anteriormente, em relação a mantermos uma variável que determina a região do programa que está sendo executada, pode ser utilizada para indicar ao programador onde o programa está travando. Basta para isso criar uma rotina que paralise o sistema quando houver um

reset devido ao WDT e mostre, de alguma forma, o valor dessa variável. Com isso o programador poderá determinar a região de travamento para tentar resolver o problema.

MODO SLEEP

Como o próprio nome diz, quando entramos neste modo, estamos colocando o microcontrolador para "dormir". Esta é uma das melhores maneiras de economizar energia em sistemas que podem ficar paralisados temporariamente. Este modo é muito utilizado em sistemas alimentados por pilhas ou baterias, pois o PIC pode passar a consumir menos que 1 µA.

Quando entramos neste modo, o oscilador é imediatamente paralisado, e com ele todo o processamento. O estado das portas é mantido, isto é, aquelas que eram entradas continuam como entradas, assim como o nível lógico das portas que eram saídas permanecem inalterados. No entanto, como o objetivo normalmente é economizar energia, o estado das portas deve ser minuciosamente checado e acertado antes de entrarmos neste modo. Configurar as portas para entrada, quando possível, resulta em maior economia de energia.

ENTRANDO NO MODO SLEEP (SLEEP)

Para entramos no modo Sleep (Power-down), devemos executar a instrução **SLEEP**. No entanto, antes de o oscilador ser desligado, o bit /PD do STATUS é limpo (0) e o bit /TO é setado (1). Esses bits serão utilizados na hora de sairmos do Sleep. Como já dissemos, o estado das portas é mantido. O pino de Master Clear (MCRL) deve permanecer o tempo todo em nível lógico alto (1). Vale observarmos que, neste modo, apesar de todo o sistema ser paralisado, o oscilador relativo ao WDT continua funcionando (caso ele esteja ligado).

SAINDO DO MODO SLEEP

Uma vez dentro do modo Sleep, existem três maneiras possíveis de sairmos dele:

1. Por meio de um reset externo, através do pino MCRL.
2. Por um estouro de WDT, no caso de ele estar habilitado.
3. Devido a uma interrupção externa ou por mudança de estado, gerada nos pinos RB0, RB4, RB5, RB6 ou RB7. A interrupção de final de escrita na EEPROM também pode fazer com que o PIC saia do modo sleep. Já a interrupção de Timer0 jamais acontecerá no modo sleep, pois o timer pára de ser incrementado, uma vez que não há mais ciclos de máquina.

No primeiro caso, um reset de hardware foi necessário, o que irá reinicializar o sistema, independentemente de estarmos no modo sleep ou não. Esse reset pode ser reconhecido, checando os estados dos bits /TO = 1 e /PD = 0.

Já no segundo caso, o sistema "acordará" por causa de um estouro do WDT. Como esse timer não é paralisado no modo Sleep, caso ele esteja ligado, acontecerá um estouro, pois não está sendo executada a instrução CLRWDT nenhuma vez. Quando isso acontecer, o programa continuará a ser executado na linha seguinte à instrução SLEEP. Podemos checar esta situação por meio do bit /TO = 0.

Atenção: alguns modelos de PIC resetam quando acontece um estouro de WDT, voltando ao início do programa, e não à linha seguinte da instrução SLEEP.

O terceiro e último caso é o mais complexo de todos. No caso da chave individual da interrupção estar ligada, o sistema sairá do modo sleep, executando a linha seguinte à instrução SLEEP. Caso contrário (chave desligada), o sistema continuará dormindo. Caso a chave geral GIE também esteja ligada, então no ciclo seguinte o programa será desviado para o endereço 0x04, a fim de tratar a interrupção ocorrida. Por isso, quando desejamos realmente tratar a interrupção que acordará o PIC, devemos colocar um NOP após a instrução SLEEP, evitando assim que qualquer ação seja tomada antes do tratamento da interrupção.

CAPÍTULO

12

RECURSOS AVANÇADOS

Este é um capítulo totalmente novo neste livro. Nossa objetivo aqui é apresentar para você os novos recursos de hardware existentes no PIC16F628A, em relação ao modelo do qual falávamos nas edições anteriores: o PIC16F84.

Deixamos claro, entretanto, que não será a nossa intenção criarmos aqui um manual para a utilização desses recursos. E fazemos isso por dois motivos: primeiro, o foco deste livro é o aprendizado da linguagem de programação do PIC; e, em segundo lugar, atualmente existe uma outra publicação, denominada "Conectando o PIC", destinada exatamente ao aprendizado dos recursos desses microcontroladores.

Bem, mas agora vamos logo ao assunto e vejamos quais os recursos adicionais do PIC16F628A:

1. Timer 1 (16 bits)
2. Timer 2 (8 bits)
3. CCP - Capture, Compare e PWM (10 bits)
4. Comparador analógico (2x)
5. Tensão de referência ajustável
6. USART - Comunicação serial

TIMER 1

O Timer 1 é outro contador automático do sistema, mas com uma enorme vantagem sobre o Timer 0: trata-se de um contador de 16 bits. Desta forma, o valor do contador é armazenado em duas posições da memória: TMR1L (parte baixa) e TMR1H (parte alta).

Para configurar esse timer, utilize o T1CON (veja o Apêndice A).

Outra grande vantagem do TMR1 é o fato de que ele pode trabalhar com um prescaler independente, configurado diretamente em T1CON< T1CKPS1:T1CKPS0>.

De forma semelhante ao TMR0, o TMR1 também pode ser incrementado pelo clock da máquina ou por um sinal externo, ligado ao pino T1CKI (borda de subida). Quem configura o tipo de incremento é o bit TMR1CS do registrador T1CON. Só que, quando configurado para trabalhar com incremento por sinal externo, os pinos T1OSO e T1OSI possuem também um circuito de oscilação, para podermos utilizar diretamente um Ressistor/Cristal externo só para o Timer 1. Incrível, não é mesmo? Com isso podemos fazer um timer com clock diferente da máquina. Além disso, através do bit T1CON<T1OSCEN> esse oscilador pode ser habilitado ou não, fazendo com que o TMR1 seja paralisado e economizando energia para o sistema.

Por último, para que o TMR1 funcione, o bit T1CON<TMR1ON> deve ser setado (1).

Vale comentarmos também que existe uma interrupção específica relacionada ao estouro do Timer 1. Para utilizar essa interrupção, as seguintes chaves devem ser ligadas:

- PIE1<TMR1IE>: chave individual da interrupção de TMR1.
- INTCON<PIE>: chave do grupo de interrupções de periféricos.
- INTCON<GIE>: chave geral das interrupções.

Quando a interrupção acontecer, será setado o bit PIR1<TMR1IF>. Após o tratamento, este bit não será limpo automaticamente, ficando esta tarefa para o programador.

TIMER 2

Este timer, por sua vez, volta a ser de 8 bits e só pode ser incrementado internamente pelo clock da máquina. Por outro lado, ele possui duas vantagens sobre os demais.

Primeiro, além de possuir um Prescaler próprio, possui também um Postscaler. A diferença é que o prescaler é utilizado para incrementar o timer (TMR2) propriamente dito, enquanto o postscaler conta a quantidade de estouros desse timer para poder gerar uma interrupção. Por exemplo, caso o prescaler seja de 1:4 e o postscaler seja de 1:14, o sistema funcionará da seguinte maneira: a cada quatro ciclos de máquina o TMR2 será incrementado. Depois que esse timer estourar 14 vezes, uma interrupção será gerada.

Em segundo lugar, existe um segundo registrador (PR2) utilizado para controlar o estouro de TMR2. Como assim? É muito simples. No caso do Timer 0, que também é um timer de 8 bits, o estouro acontece sempre que o valor é incrementado de 0xFF para 0x00, pois o limite máximo de bits foi atingido. No Timer 2 esse limite não precisa ser atingido, pois definimos o número máximo permitido para TMR2 através de PR2. A comparação entre esses dois registradores é feita sempre, e de forma automática, pelo sistema. Caso TMR2 = PR2, no próximo incremento será conside-

rado um estouro e TMR2 voltará a zero. Com isso não é mais necessário ficarmos inicializando o valor do contador para que a quantidade de incrementos seja a desejada. Basta especificar esse valor em PR2.

Para utilizar a interrupção desse timer, as seguintes chaves devem ser ligadas:

- PIE1<TMR2IE>: chave individual da interrupção de TMR2.
- INTCON<PIE>: chave do grupo de interrupções de periféricos.
- INTCON<GIE>: chave geral das interrupções.

Quando a interrupção acontecer, será setado o bit PIR1<TMR2IF>. Após o tratamento, este bit não será limpo automaticamente, ficando esta tarefa para o programador.

CCP - CAPTURE, COMPARE E PWM

Esses três recursos estão relacionados ao mesmo módulo dentro do PIC, denominado CCP. Isso significa que só podemos utilizar um dos sistemas de cada vez. Através do registrador CCP1CON poderemos configurar qual recurso será habilitado, alterando-se os valores dos bits CCP1M3:CCP1M0 (veja o Apêndice A).

CAPTURE

O modo Capture serve para contar o tempo, através de TMR1, entre pulsos externos através do pino CCP1 (RB3). Com isso podemos implementar facilmente um timer para períodos de ondas.

A primeira coisa a fazer é escolher de que forma o pulso será considerado em CCP1. Existem quatro opções disponíveis através do registrador CCP1CON:

- Contagem em cada borda de descida;
- Contagem em cada borda de subida;
- Contagem a cada 4 bordas de subida;
- Contagem a cada 16 bordas de subida.

Depois disso, toda vez que o evento correto acontecer no pino CCP1, os valores de TMR1L e TMR1H (Timer 1) serão transferidos para os registradores CCPRL e CCPRH, respectivamente. Basta então configurar o TMR1 corretamente, conforme as necessidades da sua forma de onda. Observe que, como esse recurso utiliza diretamente o Timer 1, seu uso em outras tarefas pode ficar comprometido.

Para utilizar a interrupção que acontece junto com esse evento, as seguintes chaves devem ser ligadas:

- PIE1<CCP1IE>: chave individual da interrupção de CCP1.
- INTCON<PIE>: chave do grupo de interrupções de periféricos.
- INTCON<GIE>: chave geral das interrupções.

Quando a interrupção acontecer, será setado o bit PIR1<CCP1IF>. Após o tratamento, esse bit não será limpo automaticamente, ficando esta tarefa para o programador.

COMPARE

Nesse modo também utilizamos o TMR1 como base de tempo. No entanto, aqui, o valor do contador é constantemente comparado ao valor escrito nos registradores CCP1L e CCP1H. Quando uma igualdade acontecer, uma das seguintes ações irá acontecer:

- O pino CCP1 (se configurado como saída) será colocado em nível lógico alto (1);
- O pino CCP1 (se configurado como saída) será colocado em nível lógico baixo (0);
- O estado do pino CCP1 não será afetado, mas uma interrupção acontecerá.

Essas ações são configuradas ao setar o modo Compare dentro do registrador CCP1CON.

Para utilizar a interrupção descrita na última ação, às seguintes chaves devem ser ligadas:

- PIE1<CCP1IE>: chave individual da interrupção de CCP1.
- INTCON<PIE>: chave do grupo de interrupções de periféricos.
- INTCON<GIE>: chave geral das interrupções.

Quando a interrupção acontecer, será setado o bit PIR1<CCP1IF>. Após o tratamento, esse bit não será limpo automaticamente, ficando esta tarefa para o programador.

PWM

O PWM é provavelmente o recurso mais utilizado dentro do CCP, já que possibilita criarmos uma saída analógica, porque quando uma onda PWM passa por um filtro externo, pode ser convertida em um sinal variável de 0 a 5V_{DC}.

Este modo também é o mais complexo de trabalharmos. Devemos começar configurando-se sua escolha em CCP1CON.

A base de tempo para nosso PWM será o Timer 2. A primeira dúvida que pode surgir aqui é em relação à precisão do sistema, já que dissemos que o PWM do PIC16F628A é de 10 bits, mas a nossa base de tempo (TMR2) é de 8 bits. Como isso é possível? A questão é que, para o PWM, não serão considerados somente os 8 bits físicos do TMR2. Os 2 bits faltantes serão adquiridos através do clock interno da máquina. Lembra quando falamos sobre Q1, Q2, Q3 e Q4? Isso mesmo, a divisão do clock por 4 gera esses tempos internos, que são utilizados para incrementar o TMR2 e também para controlar o PWM. Esses quatro tempos geram os 2 bits faltantes.

Assim sendo, o valor do período do nosso PWM está vinculado a um estouro do TMR2, isto é, sempre que $TMR2 = PR2$. Portanto, para definirmos o tempo desse período, podemos utilizar a seguinte fórmula:

$$PWM_{PERÍODO} = (PR2 + 1) * 4 * T_{osc} * TMR2_{PRESCALER}$$

Por exemplo, para um sistema rodando a 4MHz poderíamos ter a seguinte configuração:

$$PWM_{PERÍODO} = (249 + 1) * 4 * 250\text{ns} * 4$$

$$PWM_{PERÍODO} = 1\text{ms}$$

Quando ao Duty Cycle (DC), isto é, o tempo que a onda fica ativa (nível lógico alto), seu valor será considerado pelo registrador CCPR1L. Como este também é um registrador de 8 bits e precisamos de 10 para a nossa precisão, os 2 bits menos significativos encontram-se em CCP1CON<CCP1X:CCP1Y>. Assim poderemos calcular:

$$PWM_{DUTY CYCLE} = (CCP1L:CCP1CON<CCP1X:CCP1Y>) * T_{osc} * TMR2_{PRESCALER}$$

Este valor encontra-se então entre 0 e 1024, e não em uma relação direta de porcentagem do período total do PWM.

A forma de onda final, com $PWM_{PERÍODO}$ e $PWM_{DUTY CYCLE}$, será gerada contínua e automaticamente no pino CCP1 (RB3).

O modo de PWM não possui uma interrupção específica relacionada a ele.

COMPARADORES

O PIC 16F628A possui internamente dois comparadores analógicos. O interessante é que esses comparadores podem ser configurados eletronicamente em oito opções. Cada uma dessas opções ligará as entradas e saídas dos comparadores de uma forma diferente. Além disso, o resultado da comparação analógica poderá ser enviado como um sinal digital a um dos pinos do PIC e/ou tratado internamente pela lógica do sistema.

Para configurar os comparadores, altere o registrador CMCON<5:0>. Consulte o Apêndice A para obter mais detalhes.

Depois de configurados, o resultado da comparação poderá ser acessado em:

- CMCON<C1OUT>: resultado do comparador 1 (pode estar ligado eletronicamente ao pino CMP1);
- CMCON<C2OUT>: resultado do comparador 2 (pode estar ligado eletronicamente ao pino CMP1);

As entradas analógicas possuem limitações quanto à tensão, que deve variar de 0 ao máximo de V_{DD} .

Para utilizar a interrupção gerada quando o resultado de um dos comparadores é alterado, as seguintes chaves devem ser ligadas:

- PIE1<CMIE>: chave individual da interrupção dos comparadores.
- INTCON<PIE>: chave do grupo de interrupções de periféricos.
- INTCON<GIE>: chave geral das interrupções.

Quando a interrupção acontecer, será setado o bit PIR1<CMIF>. Após o tratamento, esse bit não será limpo automaticamente, ficando essa tarefa para o programador.

TENSÃO DE REFERÊNCIA AJUSTÁVEL

Esse recurso é uma inovação muito interessante na família PIC, pois se trata de uma saída analógica. É claro que é uma saída de baixa resolução, uma vez que só possui duas escalas com 16 níveis em cada uma, mas, mesmo assim, não deixa de ser uma saída ($V_{REF}/RA2$) analógica. Além disso, em uma das possíveis configurações dos comparadores, essa tensão de referência é ligada a uma das entradas dos dois comparadores.

Internamente essa tensão é feita através de um divisor resistivo com 16 resistores (~2k para cada resistor). Em cada extremidade existem mais dois resistores de ~16K cada um. Um MUX pega os níveis de tensão intermediários entre os resistores. Desta forma, a impedância de saída é relativamente alta e a corrente disponível é baixa, mas a saída pode ser buferizada externamente.

Para utilizar V_{REF} , defina se a saída estará ligada ao pino RA2/ V_{REF} ou se o uso será somente interno através do bit VRCON< V_{ROE} >.

Depois, escolha a escala de operação. Ao alterar a escala, um dos resistores de 16K da extremidade (para GND) é cortado. A alteração de escala é feita por intermédio do bit VRCON< V_{RR} >.

Escolha agora o valor da tensão desejada, configurando o MUX interno através dos bits VRCON< $V_{R3:R0}$ >.

O cálculo da tensão pode ser feito através das formulas:

- Escala baixa: $V_{REF} = (\text{MUX} / 24) * V_{DD}$
- Escala alta: $V_{REF} = (\text{MUX} / 32) * V_{DD} + (V_{DD} / 4)$

Por último, basta habilitar o circuito através do bit VRCON< V_{REN} >. Quando não estiver utilizando o V_{REF} , mantenha o sistema desativado para diminuir o consumo.

Este recurso não possui interrupção associada.

USART

A USART é, sem sobra de dúvidas, o recurso de hardware mais poderoso dentro do PIC16F628A, pois possibilita que o microcontrolador possa se comunicar com o mundo real de uma forma simples e eficiente. Com este sistema é possível implementarmos uma linha de comunicação com um PC, através do padrão RS-232, sem nos preocuparmos muito com a programação em relação às rotinas de recepção e transmissão.

Existem dois modos de comunicação serial dentro da USART do PIC: síncrono e assíncrono, ambos implementados por duas vias de comunicação (fora o GND). No modo síncrono, uma das vias é utilizada como clock (via de sincronismo, por isso o nome) e a outra via é utilizada para o tráfego de dados, em ambos os sentidos. Já no modo assíncrono, não existe uma via de sincronismo para o sistema. Assim sendo, uma das vias é utilizada para a transmissão (TX) e a outra para a recepção (RX) dos dados. Este modo será mais comentado aqui, pois é o sistema utilizado no padrão RS-232.

MODO ASSÍNCRONO

A grande dúvida sobre a operação assíncrona diz respeito ao sincronismo que deve haver entre os dois lado da comunicação para que um dados seja entendido corretamente.

Nesta comunicação, o sincronismo é feito pelo próprio dado, isto é, os sistema conseguem saber quando cada dado começa. Mas como isso é possível? Através de três conceitos muito simples: marcas para o início (start bit) e fim do dado (stop bit) e precisão no tamanho de cada bit transmitido (baud rate).

Comecemos esclarecendo a respeito do Baud Rate, isto é, a velocidade com que os dados trafegam pela comunicação. Essa velocidade irá definir o tamanho de cada bit, e por isso foi padronizada a utilização da unidade BPS (bits por segundo). Os valores de Baud Rate também foram padronizados, por exemplo, em 2400, 4800, 9600, etc. Temos então que:

$$T_{\text{BIT}} = 1 / \text{Baud Rate}$$

Bem, como somos obrigados a estabelecer uma velocidade de comunicação para o sistema (por exemplo 9600 bps), e os dois lados utilizarão esta mesma velocidade, sabemos então qual o tempo de cada bit trafegando nas linhas TX e RX. Precisamos saber agora quando começa o primeiro bit. Isso será feito através do Start Bit.

O padrão para ambas as linhas de comunicação é o nível lógico alto (5V_{CC}). Quando a comunicação de um byte começa, a linha vai para nível lógico baixo (GND), permanecendo desta forma pelo tempo de 1 bit (T_{BIT}). Esse intervalo específico em zero (0) é chamado de Start Bit e serve para sincronizar a recepção do dado. Depois disso, 8 bits serão transmitidos, respeitando-se a relação lógica de tensão (0 = GND e 1 =

5V). No final, obrigatoriamente deve voltar ao nível alto, para que o próximo Start Bit seja reconhecido.

Ao final da transmissão, pode ser definido também que o sistema opte por um intervalo (T_{BIT}) em nível alto, para ajudar a separar um dado do outro. Este intervalo é chamado de Stop Bit.

Por último, existe ainda a possibilidade da transmissão de um 9º bit (entre o final do dado e o Stop Bit), que serve para checar a integridade da informação recebida. Esse bit é chamado de Paridade, e pode ser configurado como ímpar ou par. A paridade em uma comunicação é opcional.

Assim, o primeiro passo a tomarmos em nosso projeto é a definição dos parâmetros a serem utilizados em ambos os lados da comunicação, como por exemplo 9600-8N1. Isso significa que a comunicação será efetuada a uma taxa de 9600 bits por segundo, com 8 bits de dados, nenhuma paridade e 1 Stop Bit.

Muito bem, vejamos agora como configurar o PIC para trabalhar com esses parâmetros.

Primeiramente informe ao PIC para trabalhar no modo assíncrono. Isso é feito limpando-se (0) o bit TXSTA<SYNC>.

Depois, ajuste o Baud Rate desejado através do bit TXSTA<BRGH> e do registrador SPBRG. Calcule o valor de SPBRG através da fórmula:

Para TXSTA<BRGH> = 0 (velocidades baixas):

$$BR = F_{osc} / (64 \times (SPBRG + 1)) \text{ ou}$$

$$SPBRG = F_{osc} / (64 \times BR) - 1$$

Para TXSTA<BRGH> = 1 (velocidades altas):

$$BR = F_{osc} / (16 \times (SPBRG + 1)) \text{ ou}$$

$$SPBRG = F_{osc} / (16 \times BR) - 1$$

Como SPBRG deve ser arredondado para um valor inteiro, existirá com erro no Baud Rate. Faça de tudo para que esse erro seja o menor possível. Se necessário, altere o valor de TXSTA<BRGH> e recalcule para ver se melhora. Em último caso, altere o valor de F_{osc} (freqüência do oscilador). Para comunicações mais críticas, utilize um cristal externo.

Depois, ajuste o uso ou não da paridade, através dos bits TXSTA<TX9> (transmissão) e RCSTA<RX9> (recepção). Quando esses bits estão em 1, a comunicação é configurada para 9 bits (com paridade).

Depois, ative o sistema da USART através do bit RCSTA<SPEN>. Após essa ação, os pinos serão chaveados internamente para trabalharem como TX (saída) e RX (entrada).

Agora existem dois outros bits para habilitar (ou desabilitar) a transmissão e a recepção de dados.

Enquanto a recepção estiver habilitada através do RCSTA<RXEN>, cada byte recebido por RX será armazenado em RCREG. Após o término de uma recepção, uma interrupção será gerada, avisando que existe um byte recebido no buffer de entrada. Sua única função é tratá-lo. Quando um segundo byte for completamente recebido antes que o dado anterior tenha sido lido pelo programa, uma sobreescrita irá acontecer, perdendo-se o dado anterior. Quando isso acontecer, o bit RCSTA<OERR> será setado.

Por outro lado, enquanto a transmissão (sistemas independentes) estiver habilitada através do TXSTA<TXEN>, cada dado movido para TXREG será automaticamente enviado através do pino TX. Após o término de uma transmissão, uma interrupção será gerada, avisando que o buffer de saída está vazio e pronto para o próximo dado.

Para utilizar a interrupção de recepção, as seguintes chaves devem ser ligadas:

- PIE1<RCIE>: chave individual da interrupção de recepção.
- INTCON<PIE>: chave do grupo de interrupções de periféricos.
- INTCON<GIE>: chave geral das interrupções.

Quando a interrupção acontecer, será setado o bit PIR1<RCIF>. Após o tratamento, esse bit não será limpo automaticamente, ficando esta tarefa para o programador.

Para utilizar a interrupção de transmissão, as seguintes chaves devem ser ligadas:

- PIE1<TXIE>: chave individual da interrupção de transmissão.
- INTCON<PIE>: chave do grupo de interrupções de periféricos.
- INTCON<GIE>: chave geral das interrupções.

Quando a interrupção acontecer, será setado o bit PIR1<TXIF>. Após o tratamento, este bit não será limpo automaticamente, ficando essa tarefa para o programador.

CAPÍTULO

13

NOVAS CONSIDERAÇÕES SOBRE O HARDWARE

Puxa, você já chegou bem longe em relação à programação do PIC. Na verdade, você já aprendeu basicamente tudo relacionado à programação e já deve ser capaz de criar seus próprios sistemas. Daqui para a frente serão vistas algumas dicas para ajudá-lo a resolver problemas e melhorar seu rendimento como programador.

EVITANDO PROBLEMAS COM O PIC E SUAS PORTAS

O PIC é um componente extremamente resistente. Difícilmente ele queima por motivos de curto ou estática, nem mesmo no caso de ser montado ao contrário (em relação ao soquete). No entanto, ele é um componente eletrônico e relativamente caro para ficarmos desperdiçando. Por isso, um pouco de cuidado e precaução não farão mal a ninguém:

- Evite contato direto de sua mão com os pinos do componente, devido à eletricidade estática.
- Na hora de retirá-lo do soquete, muito cuidado para não entortar os terminais. Como normalmente o PIC deve ser retirado, gravado e remontado por várias vezes até que o sistema funcione corretamente, as pernas podem quebrar, inutilizando o componente. O mais indicado é a utilização de uma pinça de retirada de Cis e utilizar mais um soquete.
- Cuidado com a configuração das portas, para evitar um curto em uma porta selecionada como saída.
- Checar a capacidade máxima de saída das portas utilizadas para evitar uma sobrecarga. O PIC possui uma capacidade relativamente alta em suas portas, mas, se for necessário, utilize drives externos.
- PIC é um microcontrolador bastante imune a ruídos, mas é previsível que ele não seja 100% eficiente nesta função. Por isso, quanto mais se planejar a imunidade a ruídos do seu circuito (e também da placa), menos problemas você terá.

POWER-ON RESET (POR) MELHORADO

O objetivo do POR melhorado, em relação ao sistema básico já apresentado, é a possibilidade de alterarmos a constante de tempo da estabilização do circuito e da fonte antes de o PIC começar a funcionar. Um circuito baseado em um RC ligado ao MCLR é capaz de criar esse retardo. Esse sistema é recomendado em casos que possuem uma fonte que demora muito para atingir a tensão estabilizada.

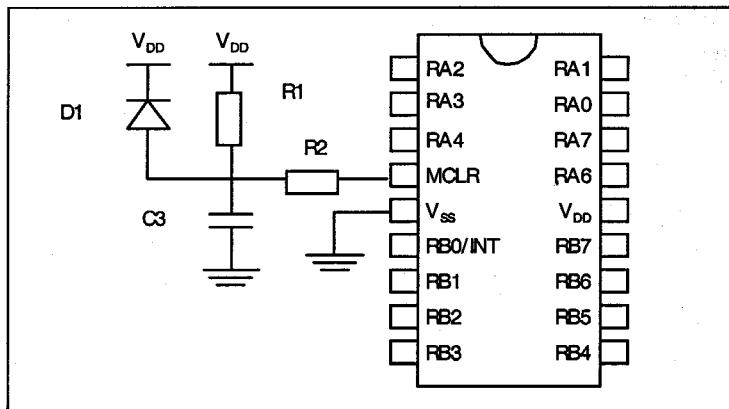


Figura 13.1 - POR Externo Melhorado.

No circuito apresentado, o diodo D1 é utilizado para ajudar a descarregar o capacitor C3 quando a fonte é desligada, resetando o sistema mais rapidamente. O tempo de retardo pode ser regulado, variando os valores de R1 e C3. O valor recomendado de R1 deve estar abaixo de 40k. Quanto ao R2, que é utilizado para limitar a corrente máxima no MCRL, seu valor deve estar entre 100R e 1k.

O PIC16F628A possui um sistema interno (Power-UP Timer) para criar esse mesmo retardo após a alimentação do sistema, que pode ser ligado ou não, conforme a configuração na hora de gravar o PIC. No entanto, esse sistema interno não pode ter seus valores alterados, por isso, em certas situações, o circuito externo pode ser mais recomendável.

BROWN-OUT

O circuito de Brown-Out é utilizado para forçar um reset quando a tensão de alimentação sofre uma pequena queda. Ele é extremamente recomendado em projetos que possibilitam ao usuário desligar e religar rapidamente a alimentação. Nestes casos, o brown-out pode evitar problemas de reinicialização.

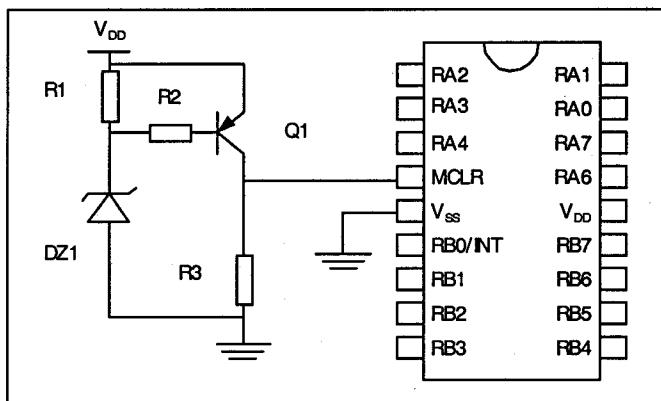


Figura 13.2 - Brown-Out Externo.

Este circuito fará com que o sistema reset quando $V_{DD} < V_Z$ (DZ1) + 0.7 Volts.

A regulação da tensão de referência injetada na base do transistor pode ser conseguida por meio de um divisor resistivo no lugar do zener DZ1 e do resistor R1. Esse sistema pode ser mais barato, no entanto é também muito menos preciso.

O PIC16F628A também possui um sistema interno de BOR para resetar automaticamente quando a tensão for menor que 4V. No entanto, esse sistema interno não pode ter seus valores alterados, e, por isso, em certas situações, o circuito externo é mais recomendável.

CAPÍTULO

14

SIMULANDO E “DEBUGANDO” O SISTEMA

INTRODUÇÃO

O MPLab é uma ferramenta extremamente poderosa, pois possui diversos recursos para rodar um programa em modo de simulação, possibilitando o monitoramento de todo o sistema para identificar e solucionar problemas. O que nós veremos a partir de agora são os principais recursos disponíveis para que você possa ganhar tempo na solução de problemas do seu sistema.

UMA LISTAGEM COMPLETA (ABSOLUT LIST)

A absolut list é uma listagem detalhada do programa após a compilação. Além de mostrar o código no assembler original, ela introduz uma série de outras informações capazes de nos ajudar a entender o que está acontecendo. Essa listagem pode ser acessada abrindo-se o arquivo com o mesmo nome do código-fonte, só que com a extensão LST. Utilize o comando *File > Open...*

As principais informações disponíveis são:

- Código-fonte original;
- Mensagens de erros e alertas nas posições corretas;
- Número de linha de texto;
- Posição na memória de programação;
- Definição de todos os símbolos (defines, variáveis, constantes, etc.);
- Mapa da memória de programação;
- Relatório de memória ocupada e livre;
- Relatório de compilação;
- Formatação apropriada para impressão.

Muitas opções de geração dessa listagem podem ser alteradas por meio da diretriz LIST (veja o Apêndice C).

ACERTANDO AS CONDIÇÕES DO HARDWARE

A simulação dentro do MPLab é muito útil, pois ela pode ser realizada sem a montagem de nenhum hardware, economizando tempo e recursos. É lógico que, em certas situações, a simulação torna-se complicada, demorada e às vezes até impossível, obrigando-nos a partir para um protótipo real ou a utilização de um emulador. Mas, na maioria dos casos, o simulador será capaz de resolver uma grande quantidade de problemas.

Antes de começar uma simulação, certifique-se que o sistema de simulação está ligado. Através do comando *Debugger > Select Tool* você deve marcar a opção **MPLAB SIM**. Na barra de Status, do lado esquerdo, esta opção irá aparecer como a ativa no momento. Na parte superior da tela aparecerá uma nova barra de botões, os quais serão mostrados aqui ao lado de suas funções.

Depois, você deve informar ao MPLab quais as reais condições de funcionamento da sua máquina, isto é, como será o hardware externo ao PIC.

Por meio do comando *Debugger > Setings...* você terá acesso a algumas pastas de configuração das condições de simulação. A mais importante no momento é a chamada *Clock*, onde poderemos escolher a freqüência de trabalho do oscilador. O acerto dessa freqüência é primordial para o correto funcionamento da ferramenta de medição de tempo real.

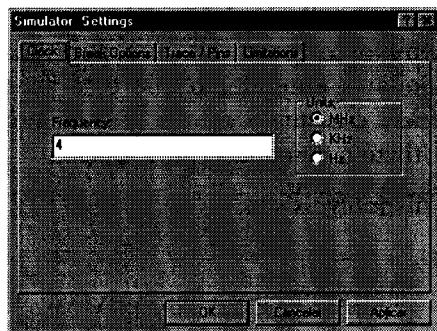


Figura 14.1 - Alteração do Clock.

Na pasta **Break Option**, você terá acesso às opções em relação as ações que podem fazer com o programa pare. A primeira opção (*Global Break Enable*) diz respeito a habilitação dos Break Points, que serão vistos mais adiante.

A segunda opção diz respeito à operação da Pilha (Stack). Como já dissemos anteriormente, quando a Pilha do PIC estourar, isto é, passar dos oito níveis de desvios susseguivos, nada acontecerá, exceto a perda dos primeiros dados. Entretanto, durante a simulação, esse acontecimento poderá ser sinalizado, através de uma parada do programa ou de um reset do sistema.

Por último, você terá acesso às opções relacionadas ao WDT, podendo fazer com que o sistema reset caso ele estoure (operação normal), ou ainda, que simplesmente paralize a execução do programa. Essas opções só estarão disponíveis caso o WDT esteja habilitado em *Configure > Configuration Bits*.

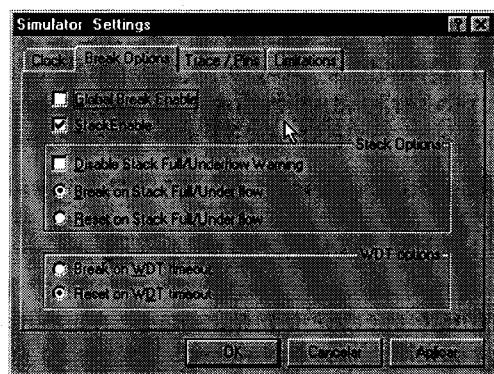


Figura 14.2 - Configurações de Paradas para o Simulador.

Na pasta *Trace / Pins* é possível simular a ligação de um resistor de pull-up (ligação ao V_{DD}) no pino MCLR. Isso é necessário para garantir a execução do programa quando estamos utilizando o Master Clear externo. O quadro *Trace Option* diz respeito às funções de Trace, que serão comentadas adiante.

A última pasta traz comentários sobre as limitações do simulador para o PIC em questão.

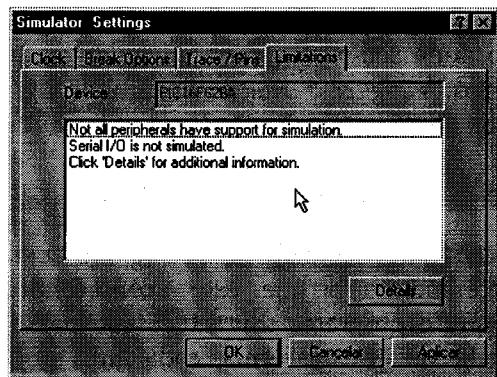


Figura 14.3 - Observações sobre Limitações.

EXECUTANDO O PROGRAMA

Uma vez acertadas as condições da máquina para o correto funcionamento do sistema, então podemos começar a rodar o programa para verificarmos se a lógica implementada está realmente correta. Existem várias maneiras de executarmos o programa. Vejamos então cada uma delas.

RODANDO DIRETO (RUN)

 No modo Run, o programa será executado da maneira mais rápida possível (a velocidade real depende do computador utilizado) e só parará quando receber uma instrução para tal. Desta maneira, enquanto o programa está sendo executado, você não visualiza nada do que está acontecendo. Quando o programa for paralisado, todos os dados de monitoramento serão atualizados. Para você saber que o modo Run está em execução, a barra de Status irá apresentar uma indicação apropriada.

Este modo é ligado por meio do comando *Debugger > Run* ou pela tecla F9.

RODANDO EM MODO DE ANIMAÇÃO (ANIMATE)

 O modo Animate executa o programa de maneira similar ao modo Run; entretanto, o MPLab mostra qual linha do programa está sendo executada no momento e atualiza todos os dados de monitoramento. Por isso, o modo Animate é muito mais lento que o Run. A velocidade depende tanto do computador quanto da quantidade de dados que estão sendo monitorados. A linha executada no momento será indicada no código-fonte através de uma seta verde no lado esquerdo.

O acesso ao modo Animate é feito por intermédio do comando *Debugger > Animate*.

PARALISANDO A EXECUÇÃO (HALT)

 A execução pode ser paralisada a qualquer momento pela tecla F5 ou pelo comando *Debugger > Halt*. A linha onde o programa foi paralisado será indicada no código-fonte através da seta verde.

RODANDO PASSO A PASSO (STEPS)

 Muitas vezes o modo Animate ainda é rápido demais para que possamos avaliar corretamente o que está acontecendo. Por isso o MPLab possibilita que o programa seja executado passo a passo, isso é, uma linha de cada vez. Desta forma, cada linha pode ser executada pelo comando *Debugger > Step Into* ou por intermédio da tecla F7.

 Existem situações ainda em que não nos interessa a visualização de todas as instruções de sub-rotinas, pois elas já foram testadas e isto só resultaria em perda de tempo. Por isso foi criada uma segunda maneira de execução passo a passo, que executa diretamente as sub-rotinas de forma contínua, sem perder tempo com elas. Para isso podemos utilizar o comando *Debugger > Step Over* ou a tecla F8.

RESETANDO O PROGRAMA



O programa também pode ser resetado a qualquer momento, como se houvesse um reset por meio do pino de Master Clear. Isso fará com que a execução recomece no endereço do vetor de reset (0x00). A RAM, entretanto, não é afetada. Esse reset pode ser executado pela tecla F6 ou pelo comando *Debugger > Reset > Processor Reset*.

PARANDO NOS PONTOS CERTOS (BREAKPOINTS E RUN TO CURSOR)

Com o passar do tempo e a aquisição de experiência na utilização do simulador, você verá que somente certas partes do código devem ser checadas e testadas, e que muitas outras podem ser desprezadas pela certeza de estarem corretas. A inicialização do PIC pode ser um exemplo disso. O simulador possui então ferramentas para que o programa seja executado diretamente até um determinado ponto e depois paralisado. A maneira mais rápida de conseguir isso é por meio do comando *Run to Cursor*, acessível pelo botão direito do mouse. Ao entrar nesse comando, o programa será executado até a linha em que se encontra o cursor.

Outra maneira de conseguir o mesmo resultado é por intermédio da inserção de um *Breakpoint*. Ele também é acessível pelo botão direito do mouse, através do comando *Set Breakpoint*. Quando é introduzido, do lado direito da linha em questão, irá aparecer um círculo vermelho com um B dentro. Sempre que um break point é encontrado, a execução é paralisada (tanto em modo Run quanto em Animate). A grande vantagem do breakpoint é que podemos introduzir vários no mesmo código. Para eliminar o breakpoint, basta acessar o comando *Remove Breakpoint* na mesma linha. Um duplo clique sobre a linha também controla o breakpoint da mesma.

Existem ainda comandos (também no menu acessado com o botão direito do mouse) para habilitar/desabilitar os breakpoints existentes, individualmente ou de forma geral.

Os breakpoints podem ser controlados também por uma tela acessada pelo comando *Debugger > Breakpoints...* ou pela tecla F2.

Com esse recurso podemos, por exemplo, parar a execução quando um registrador atingir um determinado valor (podem ser feitas ainda comparações como maior, menor, maior e igual, etc.).

CONTROLANDO AS PASSAGENS (TRACE)

Outro recurso muito poderoso para a debugação é o *Trace*. Quando esse recurso está ligado (vide comando *Debugger > Settings...*), todas as linhas executadas serão gravadas em um arquivo de LOG, que poderá ser acessado através do comando *Debugger > Simulator Trace*. Nesta tela, para cada linha, serão mostrados dados em diversas colunas:

Coluna	Descrição
Line	Ordem de execução.
Addr	Endereço na memória de programa.
Op	Código em linguagem de máquina.
Label	Nome relacionado à posição da memória de programa.
Instruction	Comando em assembler.
SA	Endereço da origem, se aplicável.
SD	Valor da origem, se aplicável.
DA	Endereço do destino, se aplicável.
DD	Valor do destino, se aplicável.
Cycles	Contador de tempo. Pode ser mostrado em diversas unidades mediante o menu do botão da direita.

Essas colunas podem ser configuradas (visíveis ou não) através de um clique com o botão da direita sobre seus títulos. Suas larguras também podem ser facilmente alteradas arrastando-se as linhas entre elas.

Com o botão da direita do mouse sobre a área central dessa janela é possível ter acesso a diversos outros comandos que operam com ela, desde uma impressão ou gravação em arquivo até a alteração de parâmetros. Esta tela não é automaticamente atualizada quando mantida aberta. Para isso, utilize os comandos *Reload* ou *Refresh*.

O contador (cycles) é resetado junto com o microcontrolador (comando *Debugger > Reset*) e a tela Trace aceita até 32.767 instruções. Sempre que ela é fechada, os registros anteriores são perdidos.

OUTROS RECURSOS

É possível ainda alterarmos diretamente o número da linha de programa (PC) que será executada em seguida. Isso é muito útil quando queremos pular um pedaço do código ou testar uma rotina específica. Para alterarmos o program counter, basta colocarmos o cursor na linha desejada e acessarmos o comando *Set PC at Cursor* através do menu do botão direito (janela do código-fonte).

VISUALIZANDO A MEMÓRIA

Até agora você aprendeu os recursos para executar o programa implementado e também paralisá-lo quando necessário, mas e quanto ao monitoramento dos registradores? Para podermos realmente saber se a lógica de operação está funcionando corretamente, é preciso verificarmos o estado de diversos registradores (SFRs ou variáveis) durante a execução.

MAPA DA MEMÓRIA

Podemos visualizar, por exemplo, um mapa total da memória, isto é, o valor armazenado em cada byte da RAM. Esta tela mostra a RAM total e pode ser utilizada para monitorar tanto os SFRs quanto as variáveis do sistema. A maneira como os dados são mostrados pode ser alterada pelos dois botões existentes na parte inferior da mesma. Esta tela pode ser acessada pelo comando *View > File Registers*.

Address	00	01	02	03	04	05	06	07	08	09	A	B	C	D	E	F	ASCI
0000	00	00	08	1B	00	20	00	--	--	--	00	00	00	--	00	00	.. - - - -
0010	00	00	00	--	00	00	00	00	00	00	--	--	--	00	--	--	- - - -
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0080	--	FF	08	1B	00	DF	FF	--	--	--	00	00	00	--	00	--	- - - -
0090																	

Figura 14.4 - Visualização da RAM.

Clicando-se duas vezes sobre um dos valores apresentados, pode-se alterar o mesmo, de forma simples e rápida. Com o botão da direita do mouse também é possível ter acesso a um menu com diversos comandos.

REGISTROS ESPECIAIS (SFRs)

Nem sempre precisamos visualizar a memória inteira, certo? O MPLab possui então uma tela para visualização somente dos SFRs, o que, em muitos casos, é mais que suficiente. Os SFRs podem ser monitorados pelo comando *View > Special Function Registers*.

Address	SFR Name	Hex	Decimal	Binary	Char
0000	UREG	--	--	11100110	- - - -
0001	THRO	00	0	00000000	.
0002	PCL	08	8	00001000	.
0003	STATUS	1B	27	00011011	.
0004	FSR	00	0	00000000	.
0005	PORTA	20	32	00100000	.
0006	PORTB	00	0	00000000	.
000A	PCLATH	00	0	00000000	.

Figura 14.5 - Visualização dos SFRs.

A ordenação dos dados pode ser alterada clicando-se sobre os títulos das colunas. Também valem os comentários anteriores sobre alteração de valores e menu de comandos extras.

PILHA (STACK)

A situação atual da pilha pode ser visualizada e alterada por intermédio do comando *View > Hardware Stack*.

EEPROM

A memória EEPROM também pode ser visualizada e alterada por intermédio do comando *View > EEPROM*.

MEMÓRIA DE PROGRAMA

A memória de programa (código compilado que será gravado no PIC) pode ser visualizada por intermédio do comando *View > Program Memory*. O incrível é que esta janela, assim como as outras, também permite que sejam alterados os valores apresentados com apenas um duplo clique.

Line	Address	Opcode	Description
10	0009	3FFF	ADDLW 0xff
11	000A	3FFF	ADDLW 0xff
12	000B	3FFF	ADDLW 0xff
13	000C	3FFF	ADDLW 0xff
14	000D	3FFF	ADDLW 0xff
15	000E	3FFF	ADDLW 0xff
16	000F	3FFF	ADDLW 0xff
17	0010	3FFF	ADDLW 0xff
18	0011	3FFF	ADDLW 0xff

Figura 14.6 - Visualização da Memória de Programa.

Nesta tela também é possível ver a execução do programa e existem três modos de visualização, acessados pelos botões da parte inferior. O menu do botão direito do mouse traz uma série de outros recursos interessantes.

REGISTROS DIVERSOS (WATCHS)

A maneira mais poderosa de monitorarmos a RAM é por meio de uma janela de Watchs. Nessa janela, podemos escolher e configurar exatamente o que queremos visualizar, descartando os registradores desnecessários. Outra vantagem da janela Watch é que ela pode ser gravada na forma de um arquivo.

Desta maneira, nosso projeto pode possuir várias janelas Watchs diferentes, sendo uma para cada tipo de simulação. Essas janelas podem ser criadas pelo comando *View > Watch*. Durante a criação de uma janela, o botão “Props” pode alterar as propriedades de visualização do registrador, como, por exemplo, a formatação (Hex, decimal, binário, ASCII, etc.).

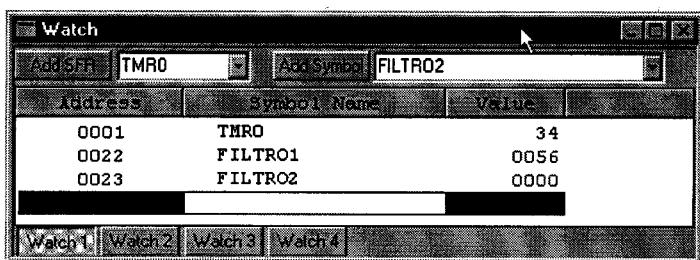


Figura 14.7 - Visualização Específica da RAM.

Uma vez dentro da janela, inserções e eliminações de registradores podem ser efetuadas através dos botões superiores e pela tecla DEL ou pelo menu do botão direito do mouse. Nesse menu, existem também comandos para salvar e recuperar a configuração atual em um arquivo (*Save* e *Load Watch Tab*).

Observe também que os seus registradores podem ser agrupados em pastas diferentes, acessadas pelos botões existentes na parte inferior da tela. Essas pastas podem ser renomeadas e até criadas outras, se necessário. Tudo isso é feito pelo menu auxiliar (botão da direita do mouse), através dos comandos *Add*, *Rename* e *Remove Watch Tab*.

Para alterarmos as propriedades de visualização de um registrador, utilize o comando *Properties...* do menu auxiliar.

Para alterar o valor de um registrador, basta clicar duas vezes sobre o valor atual do mesmo.

CONTROLANDO AS ENTRADAS

A janela de modificação de registradores é muito útil, mas ela possui uma limitação: não podemos alterar os pinos configurados como entradas. Mas como podemos então tornar nossa simulação real sem afetarmos as entradas do sistema? Na verdade, essas entradas terão de ser alteradas, mas por intermédio de um outro recurso: os estímulos.

ESTÍMULOS DIRETOS (ASSÍNCRONOS)

No simulador do MPLab, podemos associar botões aos pinos de entrada do PIC, podendo afetar diretamente o estado desses pinos com um simples “clique” do mouse. Para isso, execute o comando *Debugger > Stimulus* e visualize a pasta *Pin Stimulus*. Esta pasta é muito parecida com uma planilha. Para controlar um determinado pino, será necessário acrescentarmos uma linha relacionada a ele. Faça isso através do botão *Add Row*. Agora, na parte inferior da tela, configure corretamente as colunas relacionadas à nova linha que você acabou de inserir.

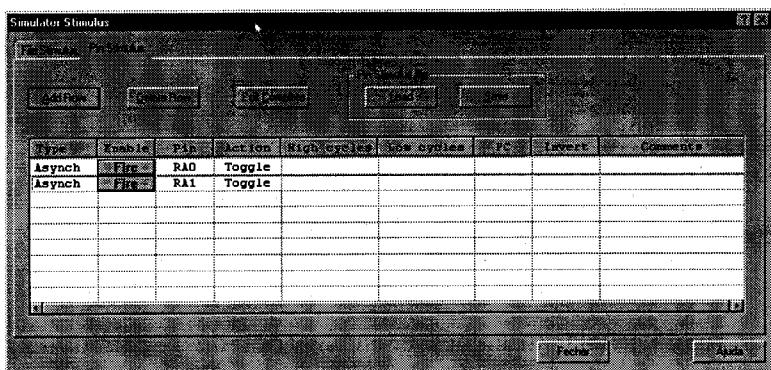


Figura 14.8 - Estímulos Manuais para as Entradas.

A coluna **Type** deve permanecer com o valor **Asynch** (assíncrono). Na coluna **Pin**, basta selecionar o pino do PIC (entrada) que receberá o estímulo. Observe que, além das portas, estão disponíveis também os pinos de Master Clear e T0CKI. Na coluna **Action**, defina como este estímulo será executado:

- **Pulse:** Estímulo de somente um pulso, invertendo momentaneamente o estado atual do pino. A duração do pulso é de somente um ciclo de máquina, por isso nem sempre é fácil utilizar esse tipo de estímulo.
- **High:** Força o estado do pino para nível alto (1).
- **Low:** Força o estado do pino para nível baixo (0).
- **Toggle:** Inverte o estado do pino a cada toque. Esse tipo de estímulo é o mais utilizado.

Exceto a coluna **Comments** (comentários), as demais não estão disponíveis para este tipo de estímulo.

Muito bem, agora, toda vez que você desejar criar um estímulo em uma das entradas definidas, basta pressionar o botão **Fire** existente na coluna **Enable**.

Valem algumas observações muito importantes sobre a tela de estímulos:

- Quando ela está aberta, a execução do programa fica muito mais lenta. Por isso, se não estiver utilizando os estímulos, feche-a.
- Somente um botão pode ser pressionado de cada vez, isto é, entre um estímulo e outro, você deve executar pelo menos uma instrução do seu código.
- Não se esqueça que, ao clicar em um dos botões de estímulos, essa tela passa a ser a ativa. Para poder voltar a executar o programa, pela tecla F7, por exemplo, será necessário antes clicar na tela do código-fonte para torná-la ativa novamente. No modo Run ou Animate, você pode clicar diretamente nos botões de estímulos e ver o que acontece com seu programa.
- Antes de fechar esta tela, salve a configuração dos estímulos através do botão **Save**.

ESTÍMULOS PERIÓDICOS (SÍNCRONOS)

São os que podem ser programados para acontecerem periodicamente de acordo com o clock da máquina. Na mesma janela estudada anteriormente nós poderemos configurar também este tipo de estímulo automático e repetitivo. Basta escolher o pino e informar o tempo de duração do estado alto (1) e do estado baixo (0), e você estará criando uma freqüência no pino escolhido. Lembre-se de que os valores introduzidos são relativos a ciclos de máquina, e não a tempo propriamente dito. Colocando somente o valor para o campo “High” e clicando no quadro “Invert”, você criará uma onda quadrada.

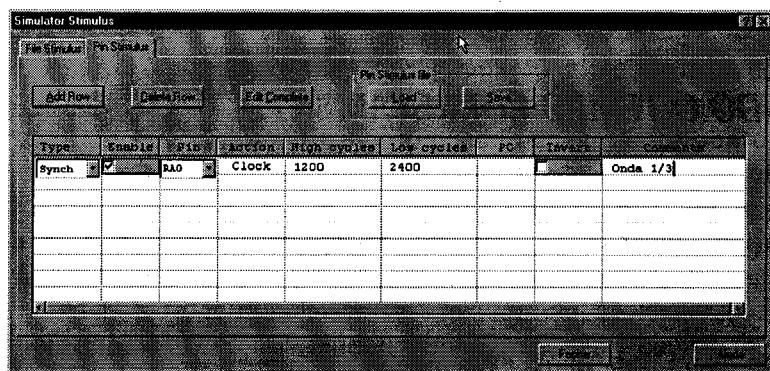


Figura 14.9 - Estímulos Automáticos para as Entradas.

ARQUIVO DE ESTÍMULOS

Você pode criar um arquivo de texto para gerar estímulos automáticos nos pinos do microcontrolador. Trata-se de um arquivo especial que será gravado com a extensão “.SSTI”. O próprio MPLab irá gerenciar esse arquivo através da pasta *File Stimulus*, acessada pelo comando *Debugger > Stimulus*. Para começar, utilize o botão *Add* (Input Files) e especifique o nome do arquivo que armazenará seus dados. Depois de criado, utilize o botão *Edit* para poder alterar os dados dos estímulos. Os botões do grupo *Edit Controls* servirão para inserir e deletar linhas na parte inferior da tela (edição do evento). Cada linha se refere a um evento de tempo. Acontece que mais de um estímulo, em pinos diferentes, podem acontecer ao mesmo tempo. Por isso, é possível inserirmos outras colunas na nossa tabela, através do botão *Add Column*.

As colunas de Trigger servem para especificar quando o evento deverá ocorrer, em ciclos de máquina ou especificando-se uma posição de programa (PC) específica. As colunas *Pin/Register* e *Value* referem-se ao local relacionado ao destino: pode ser um pino ou um registrador qualquer.

Uma vez alterados os valores, salve o arquivo. O grupo *File Stimulus* serve para armazenar a lista de todos os arquivos de estímulos relacionados. Observe que os estímulos podem ser gravados em grupos, através de arquivos diferentes. Na hora de rodar, todos os arquivos relacionados serão considerados.

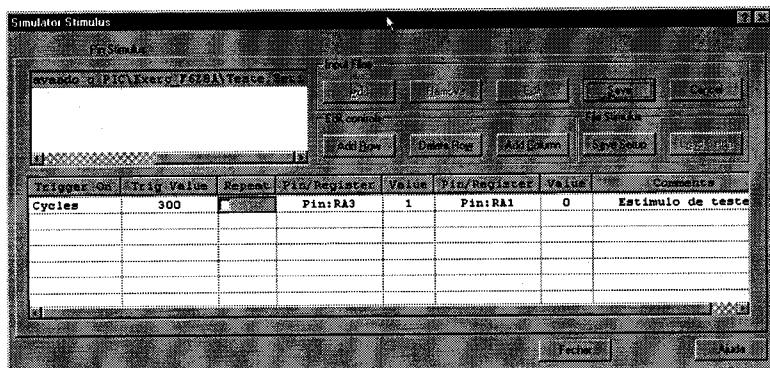


Figura 14.10 - Arquivo de Estímulos para as Entradas e/ou Registradores.

CONTANDO O TEMPO CORRETAMENTE

A calibração de tempo dentro das rotinas do seu sistema pode ser muito importante, e também muito trabalhosa. É por isso que o MPLab possui uma ferramenta para auxiliá-lo nesta tarefa. O Stopwatch é um contador de ciclos de máquina que calcula o tempo baseado na freqüência do oscilador que você configurou.

Desta forma, para sabermos exatamente quanto tempo está sendo perdido em uma rotina de delay, por exemplo, podemos executar o programa até a linha que chama a rotina, zeramos o cronômetro (botão **Zero**) e depois pressionamos F8 para que a rotina seja executada de uma só vez. Quando o delay acabar, a execução será paralisada e a tela *Stopwatch* mostrará exatamente quanto tempo durou esta tarefa. Simples, não?

Para acessar esse relógio, utilize o comando *Debugger > Stopwatch*.

Utilize o botão **Synch** para sincronizar esse recurso com as demais tarefas que utilizam o contador, tal como os estímulos ou a tela de *Trace*.

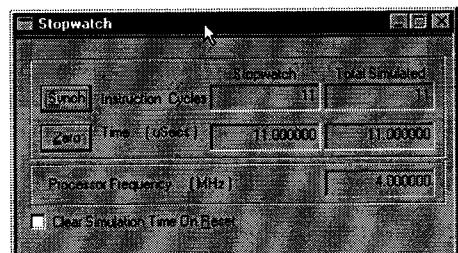


Figura 14.11 - Cronômetro.

PROJETOS PROPOSTOS

Pronto, agora que você já conhece toda a programação do PIC, está na hora de fazer seu próprio projeto. O mais indicado é elaborar um sistema e executá-lo do início até o fim. Acontece que, a grande maioria, quando chega neste ponto, ainda não tem uma idéia bem-definida do que poderia fazer como um primeiro projeto completo. Por isso, tomamos a liberdade de sugerir dois sistemas que podem ser implementados no hardware proposto no Apêndice F. Os projetos seguintes serão somente especificados, sendo que as soluções devem ser totalmente implementadas por você. Os exemplos dados no decorrer do Capítulo 11 ajudarão muito nesta tarefa. As informações dos apêndices também podem contribuir muito no desenvolvimento dos seus conhecimentos e na solução de dúvidas, e por isso devem ser estudados paralelamente à implementação do seu primeiro projeto com PIC.

De agora em diante é uma questão de criatividade e muito treino. Mão à obra e boa sorte!

TIMER

Depois do exemplo voltado ao timer, é lógico que não poderíamos deixar de sugerir a implementação de um timer completo. O mais indicado é você criar um timer para controlar uma saída, que no Exemplo 5 era um LED. Essa saída pode ser, por exemplo, um relé que controlará um sistema externo qualquer. Programe dois botões para incrementar e decrementar o tempo inicial do timer, como no caso do contador do Exemplo 4. Você pode ainda utilizar os recursos de memória do Exemplo 6 para não perder o valor ajustado. Continue utilizando dois botões para o controle, só que com as seguintes alterações: o botão 1 deve disparar e paralisar o timer; e o botão 2 deve resetá-lo, desligando a saída e voltando ao tempo inicial.

Futuramente, você pode ainda implementar uma outra saída para acionar um buzzer ao término do tempo.

DIMMER

Um dimmer também é um projeto bem interessante. Você pode, por exemplo, utilizar o sistema do contador para manipular uma variável interna que servirá de referência para o seu ajuste de intensidade. Inicialmente comece com um dimmer para tensões DC, que pode ser aplicado a uma lâmpada no hardware proposto pelo Apêndice F (saída PWM). Com a interrupção de TMRO, você pode criar uma saída PWM, controlando o tamanho do pulso com base no valor ajustado na variável interna. Continue utilizando dois botões para incrementar e decrementar o ajuste, que pode ir de 0 a 100%, com 16 steps. Como só existe um display, represente o número de forma hexadecimal, conforme o Exemplo 4. Um terceiro botão pode ser utilizado para ligar e desligar a lâmpada, no ajuste atual. Quando a lâmpada estiver desligada, o display deve ser apagado, bloqueando o ajuste de intensidade.

O projeto pode ser depois aperfeiçoado para controlar uma tensão AC. Neste caso, a saída deve disparar um TRIAC, ajustando o ângulo de disparo em relação à senóide. Para tal, será necessário um sincronismo com a rede elétrica, que pode ser conseguido injetando uma amostra da senóide no pino RA4 (já existe um conector para isso no hardware). Por meio das interrupções de mudança de estado, será possível sincronizar o início de cada um dos semiciclos. Basta então contar um tempo proporcional ao ajuste da intensidade e disparar o TRIAC. Recomendamos que esse disparo seja feito, pelo menos inicialmente, com o auxílio de um MOC, isolando assim a parte de potência da parte de controle.

REGISTRADORES ESPECIAIS (SFR)**INTRODUÇÃO**

Este apêndice destina-se a apresentar um detalhamento de todos os registradores especiais (SFRs) disponíveis no PIC16F628A, para auxiliá-lo na programação dos seus sistemas.

Para efeito de padronização, cada bit dentro desses registradores receberá um nome, sendo também especificado se este bit pode ser lido (R-Read) e/ou escrito (W-Write). Caso o bit não esteja implementado (U-unimplemented) será lido com zero.

STATUS

Registrador: STATUS				Endereços: 03h, 83h, 103h e 183h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R	R	R/W	R/W	R/W
IRP	RP1	RPO	/TO	/PD	Z	DC	C
Condição em Power-On Reset (POR)							
0	0	0	1	1	X	X	X

IRP: Seletor de banco de memória de dados usado para endereçamento indireto:

0 = Banco 0 e 1 (00h - FFh).

1 = Banco 2 e 3 (100h - 1FFh).

RP1 e

RPO: Seletor de banco de memória de dados usado para endereçamento direto:

00 = Banco 0 (00h - 7Fh).

01 = Banco 1 (80h - FFh).

10 = Banco 2 (100h - 17Fh).

11 = Banco 3 (180h - 1FFh).

- /TO:** Indicação de *Time-out*:
 0 = Indica que ocorreu um estouro do *WatchDog Timer* (WDT).
 1 = Indica que ocorreu um *power-up* ou foram executadas as instruções CLRWDT ou SLEEP.
- /PD:** Indicação *Power-down*:
 0 = Indica que a instrução SLEEP foi executada.
 1 = Indica que ocorreu um *power-up* ou foi executada a instrução CLRWDT.
- Z:** Indicação de Zero:
 0 = Indica que o resultado da última operação (lógica ou aritmética) não resultou em zero.
 1 = Indica que o resultado da última operação (lógica ou aritmética) resultou em zero.
- DC:** *Digit Carry/borrow*:
 0 = A última operação da ULA não ocasionou um estouro de dígito.
 1 = A última operação da ULA ocasionou um estouro (*carry*) entre o bit 3 e 4, isto é, o resultado ultrapassou os 4 bits menos significativos. Utilizado quando se trabalha com números de 4 bits.
- C:** *Carry/borrow*:
 0 = A última operação da ULA não ocasionou um estouro (*carry*).
 1 = A última operação da ULA ocasionou um estouro (*carry*) no bit mais significativo, isto é, o resultado ultrapassou os 8 bits disponíveis.

OPTION

Registrador: OPTION REG								Endereços: 81h e 181h							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
/RBU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	Condição em Power-On Reset (POR)							
1	1	1	1	1	1	1	1								

- /RBU:** Habilita *pull-ups* internos para o PORTB:
 0 = *Pull-ups* habilitados para todos os pinos do PORTB configurados como saída.
 1 = *Pull-ups* desabilitados.
- INTEDG:** Configuração da borda que gerará a interrupção externa no RBO:
 0 = A interrupção ocorrerá na borda de descida.
 1 = A interrupção ocorrerá na borda de subida.

- T0CS:** Configuração do incremento para o TMRO:
 0 = TMRO será incrementado internamente pelo *clock* da máquina.
 1 = TMRO será incrementado externamente pela mudança no pino RA4/T0CKI.
- T0SE:** Configuração da borda que incrementará o TMRO no pino RA4/T0CKI, quando T0CS=1:
 0 = O incremento ocorrerá na borda de subida de RA4/T0CKI.
 1 = O incremento ocorrerá na borda de descida de RA4/T0CKI.
- PSA:** Configuração de aplicação do *prescalerr*:
 0 = O *prescalerr* será aplicado ao TMRO.
 1 = O *prescalerr* será aplicado ao WDT.

PS2, PS1

- e **PS0:** Configuração de valor de *prescalerr*:

PS 2/1/0	TMRO	WDT
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

PCON

Registrador: PCOM				Endereços: 8Eh				
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
U	U	U	U	R/W	U	R/W	R/W	
-	-	-	-	OSCF		-	/POR	/BOR
Condição em Power-On Reset (POR)								
0	0	0	0	1	0	0	X	

- OSCF:** Freqüência do oscilador interno:

0 = 4 MHz.

1 = 37 KHz.

- /POR:** Indicação de *Power-On Reset* (energização):

0 = Ocorreu um *Power-On Reset*.

1 = Não ocorreu um *Power-On Reset*.

- /BOR:** Indicação de *Brown-Out Reset* (queda de energia):

0 = Não ocorreu um *Brown-Out Reset*.

1 = Ocorreu um *Brown-Out Reset*.

PCL e PCLATH

Registrador: PCL				Endereços: 02h, 82h, 102h e 182h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Parte baixa do PC							
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

Registrador: PCLATCH				Endereços: 0Ah, 8Ah, 10Ah e 18Ah			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
U	U	U	R/W	R/W	R/W	R/W	R/W
-	-	-	-	-	-	-	-
Parte alta do PC							
Condição em Power-On Reset (POR)							
-	-	-	0	0	0	0	0

FSR e o INDF

Registrador: FSR				Endereços: 04h, 84h, 104h e 184h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Ponteiro para endereçamento indireto							
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

Registrador: INDF				Endereços: 00h, 80h, 100h e 180h							
Valor apontado pelo FSR (endereçamento indireto - não é um registrador implementado)											
Condição em Power-On Reset (POR)											
-	-	-	0	0	0	0	0				

PORTA e TRISA

Registrador: PORTA				Endereços: 05h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R	R	R/W	R/W	R/W
RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
Condição em Power-On Reset (POR)							
X	X	X	X	0	0	0	0

Registrador: TRISA				Endereços: 85h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R	R	R/W	R/W	R/W
Ref. RA7	Ref. RA6	Ref. RA5	Ref. RA4	Ref. RA3	Ref. RA2	Ref. RA1	Ref. RA0
Condição em Power-On Reset (POR)							
1	1	1	1	1	1	1	1

PORTB e TRISB

Registrador: PORTB				Endereços: 06h e 106h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R	R	R/W	R/W	R/W
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

Registrador: TRISB				Endereços: 86h e 186h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R	R	R/W	R/W	R/W
Ref. RB7	Ref. RB6	Ref. RB5	Ref. RB4	Ref. RB3	Ref. RB2	Ref. RB1	Ref. RB0
Condição em Power-On Reset (POR)							
1	1	1	1	1	1	1	1

TMRO

Registrador: TMRO				Endereços: 01h e 101h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Contador Timer 0 de 8 bits							
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

T1CON, TMR1L e TMR1H

Registrador: T1CON				Endereços: 10h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
U	U	R/W	R/W	R/W	R/W	R/W	R/W
-	-	T1CKPS1	T1CKPS0	T1OSCEN	/T1SYNC	TMR1CS	TMR1ON
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

T1CKPS1

T1CKPS0: Ajuste do prescaler do Timer 1:

00 = prescaler de 1:1.

01 = prescaler de 1:2.

10 = prescaler de 1:4.

11 = prescaler de 1:8.

T1OSCEN: Habilitação do sistema de oscilação externa para os pinos T1OSO e T1OSI:

0 = Oscilador desabilitado. Caso exista um cristal externo, o sistema é desligado.

1 = Habilita o oscilador externo.

/T1SYNC: Controle do sincronismo interno. Quando TMR1CS=0 este bit é ignorado:
 0 = Sistema de sincronismo ligado.
 1 = Sistema de sincronismo desligado (modo assíncrono).

TMR1CS: Seleção da origem do *clock* para Timer 1:
 0 = *Clock* interno ($F_{osc}/4$)
 1 = *Clock* externo no pino T1OSO/T1CKI.

TMR1ON: Habilitação do Timer 1:
 0 = Timer 1 desabilitado. Paralisa o contador do Timer 1.
 1 = Timer 1 habilitado.

Registrador: TMR1L				Endereços: 0Eh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Contador Timer 1 de 16 bits - Parte baixa							
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

Registrador: TMR1H				Endereços: 0Fh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Contador Timer 1 de 16 bits - Parte alta							
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

T2CON, TMR2 e PR2

Registrador: T2CON				Endereços: 12h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
U	R/W	R/W	R/W	R/W	R/W	R/W	R/W
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

TOUTPS3

TOUTPS2

TOUTPS1

TOUTPS0: Ajuste do *postscale*:

- | | |
|--------------------------|---------------------------|
| 0000 = postscaler de 1:1 | 1000 = postscaler de 1:9 |
| 0001 = postscaler de 1:2 | 1001 = postscaler de 1:10 |
| 0010 = postscaler de 1:3 | 1010 = postscaler de 1:11 |
| 0011 = postscaler de 1:4 | 1011 = postscaler de 1:12 |
| 0100 = postscaler de 1:5 | 1100 = postscaler de 1:13 |
| 0101 = postscaler de 1:6 | 1101 = postscaler de 1:14 |
| 0110 = postscaler de 1:7 | 1110 = postscaler de 1:15 |
| 0111 = postscaler de 1:8 | 1111 = postscaler de 1:16 |

TMR2ON: Habilitação do Timer 2:

0 = Timer 2 desabilitado. Paralisa o contador do Timer 2.

1 = Timer 2 habilitado.

T2CKPS1

T2CKPS0: Ajuste do *prescaler*:

00 = *prescaler* de 1:1

01 = *prescaler* de 1:4

10 = *prescaler* de 1:16

11 = *prescaler* de 1:16

Registrador: TMR2				Endereços: 11h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Contador Timer 2 de 8 bits							
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

Registrador: PR2				Endereços: 92h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Limite superior para a contagem do Timer 2							
Condição em Power-On Reset (POR)							
1	1	1	1	1	1	1	1

INTCON, PIE1 e PIR1

Registrador: INTCON				Endereços: 0Bh, 8Bh, 10Bh e 18Bh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	X

GIE: Habilitação geral das interrupções (chave geral):

0 = Nenhuma interrupção será tratada.

1 = As interrupções habilitadas individualmente serão tratadas.

PEIE: Habilitação das interrupções de periféricos (chave de grupo para periféricos):

0 = As interrupções de periféricos não serão tratadas.

1 = As interrupções de periféricos habilitadas individualmente serão tratadas.

TOIE: Habilitação da interrupção de estouro de TMRO (chave individual):

0 = Interrupção de TMRO desabilitada.

1 = Interrupção de TMRO habilitada.

- INTE:** Habilitação da interrupção externa no pino RB0 (chave individual):
 0 = Interrupção externa desabilitada.
 1 = Interrupção externa habilitada.
- RBIE:** Habilitação da interrupção por mudança de estado nos pinos RB4 a RB7 (chave individual):
 0 = Interrupção por mudança de estado desabilitada.
 1 = Interrupção por mudança de estado habilitada.
- TOIF:** Identificação de estouro do TMR0:
 0 = Não ocorreu estouro do TMR0.
 1 = Ocorreu estouro do TMR0 (este bit deve ser limpo por software).
- INTF:** Identificação da interrupção externa no pino RB0:
 0 = Não ocorreu evento da interrupção.
 1 = Ocorreu evento da interrupção (este bit deve ser limpo por software).
- RBIF:** Identificação da interrupção por mudança de estado nos pinos RB4 a RB7:
 0 = Não ocorreu evento da interrupção.
 1 = Ocorreu evento da interrupção (este bit deve ser limpo por software).

Registrador: PIE1				Endereços: 8Ch			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	U	R/W	R/W	R/W
EEIE	CMIE	RCIE	TXIE	-	CCP1IE	TMR2IE	TMR1IE
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

- EEIE:** Habilitação da interrupção de final de escrita na EEPROM (chave individual):
 0 = Interrupção da EEPROM desabilitada.
 1 = Interrupção da EEPROM habilitada.
- CMIE:** Habilitação da interrupção do comparador (chave individual):
 0 = Interrupção do comparador desabilitada.
 1 = Interrupção do comparador habilitada.
- RCIE:** Habilitação da interrupção de recepção da USART (chave individual):
 0 = Interrupção de recepção da USART desabilitada.
 1 = Interrupção de recepção da USART habilitada.
- TXIE:** Habilitação da interrupção de transmissão da USART (chave individual):
 0 = Interrupção de transmissão da USART desabilitada.
 1 = Interrupção de transmissão da USART habilitada.

CCP1IE: Habilitação da interrupção do módulo CCP1 (chave individual):
 0 = Interrupção de CCP1 desabilitada.
 1 = Interrupção de CCP1 habilitada.

TMR2IE: Habilitação da interrupção do Timer 2 (chave individual):
 0 = Interrupção de Timer 2 desabilitada.
 1 = Interrupção de Timer 2 habilitada.

TMR1IE: Habilitação da interrupção de estouro do Timer 1 (chave individual):
 0 = Interrupção de Timer 1 desabilitada.
 1 = Interrupção de Timer 1 habilitada.

Registrador: PIR1								Endereços: 0Ch							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
EEIF	CMIF	RCIF	TXIF	-	CCP1IF	TMR2IF	TMR1IF								
<i>Condicão em Power-On Reset (POR)</i>															
0	0	0	0	0	0	0	0								

EEIF: Identificação da interrupção da EEPROM:
 0 = Não ocorreu evento de final de escrita na EEPROM.
 1 = Ocorreu evento de final de escrita na EEPROM.

CMIF: Identificação da interrupção do comparador:
 0 = Comparador não identificou alteração nas entradas.
 1 = Comparador identificou alguma alteração nas entradas.

RCIF: Identificação da interrupção de recepção da USART:
 0 = Buffer de recepção da USART está vazio.
 1 = Buffer de recepção da USART está cheio.

TXIF: Identificação da interrupção de transmissão da USART:
 0 = Buffer de transmissão da USART está cheio.
 1 = Buffer de transmissão da USART está vazio.

CCP1IF: Identificação da interrupção do módulo CCP1:
 0 = Não ocorreu condição de interrupção no módulo CCP1.
 1 = Ocorreu condição de interrupção no módulo CCP1.

TMR2IF: Identificação da interrupção de Timer 2:
 0 = Não ocorreu evento da interrupção de Timer 2.
 1 = Ocorreu evento da interrupção de Timer 2.

TMR1IF: Identificação da interrupção de estouro do Timer 1:

0 = Não ocorreu estouro do Timer 1.

1 = Ocorreu estouro do Timer 1.

CMCON

Registrador: CMCON				Endereços: 01Fh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R	R	R/W	R/W	R/W	R/W	R/W	R/W
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
<i>Condição em Power-On Reset (POR)</i>							
0	0	0	0	0	0	0	0

C2OUT: Valor da saída do comparador 2:

Normal (C2INV=0):

0 = C2 V_{IN+} < C2 V_{IN-}

1 = C2 V_{IN+} > C2 V_{IN-}

Inversa (C2INV=1):

0 = C2 V_{IN+} > C2 V_{IN-}

1 = C2 V_{IN+} < C2 V_{IN-}

C1OUT: Valor da saída do comparador 1:

Normal (C1INV=0):

0 = C1 V_{IN+} < C1 V_{IN-}

1 = C1 V_{IN+} > C1 V_{IN-}

Inversa (C1INV=1):

0 = C1 V_{IN+} > C1 V_{IN-}

1 = C1 V_{IN+} < C1 V_{IN-}

C2INV: Tipo de saída do comparador 2:

0 = Normal.

1 = Inversa.

C1INV: Tipo de saída do comparador 1:

0 = Normal.

1 = Inversa.

CIS: Chave seletora de entrada do comparador:

Quando CM2:CM0 = 001

0 = RA0 conectado a C1 V_{IN-}

1 = RA3 conectado a C1 V_{IN-}

Quando CM2:CM0 = 010

0 = RA0 conectado a C1 V_{IN-}

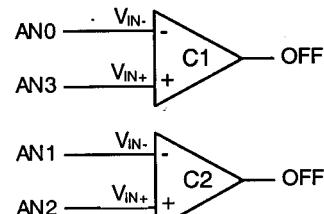
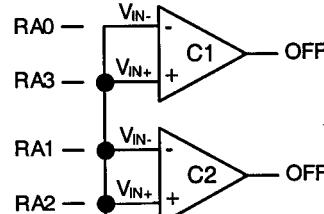
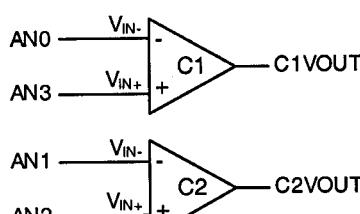
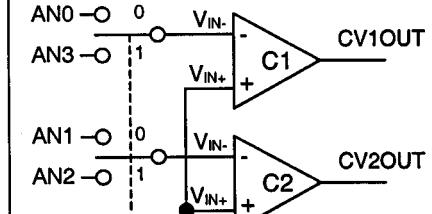
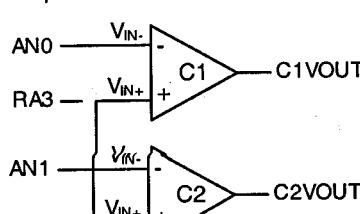
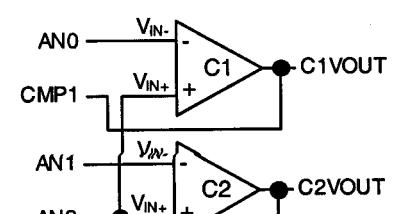
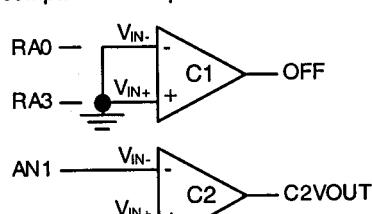
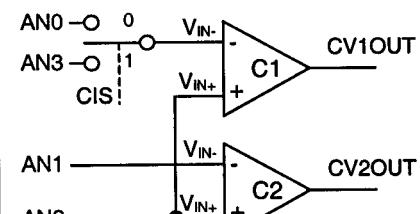
RA1 conectado a C2 V_{IN-}

1 = RA3 conectado a C1 V_{IN} .

RA2 conectado a C2 V_{IN} .

CM2, CM1

CM0: Configura a pinagem dos comparadores (modo de operação):

CM	Esquema	CM	Esquema
000	Reset dos comparadores 	111	Comparadores desligados 
100	2 comparadores independentes 	010	4 entradas multiplexadas 
011	2 comparadores com ref. comum 	110	2 comparadores com ref. comum e saídas 
101	1 comparador independente 	001	3 entradas multiplexadas 

VRCOM

Registrador: VRCOM				Endereços: 9Fh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	U	R/W	R/W	R/W	R/W
VREN	VRON	VRR	-	VR3	VR2	VR1	VR0
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

VREN: Energização do sistema de tensão de referência:

0 = Circuito de V_{REF} desenergizado.

1 = Circuito de V_{REF} energizado.

VRON: Habilitação da saída de V_{REF} :

0 = Tensão de referência desligada.

1 = Tensão de referência ligada ao pino RA2.

VRR: Seleção do range de operação do sistema de V_{REF} :

0 = Range baixo.

1 = Range alto.

VR3...VR0: Seleção do valor da tensão de V_{REF} :

Se VRR = 1:

$$V_{REF} = (VR/24) * V_{DD}$$

Se VRR = 0:

$$V_{REF} = \frac{1}{4} * V_{DD} + (VR/32) * V_{DD}$$

CCP1CON, CCPR1L e CCPR1H

Registrador: CCP1COM				Endereços: 17h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
U	U	R/W	R/W	R/W	R/W	R/W	R/W
-	-	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

CCP1X

CCP1Y: Parte baixa do PWM de 10 bits. A parte alta fica em CCPR1L. Válido somente quando em PWM.

CCP1M3

CCP1M2

CCP1M1

- CCP1M0:** Seleção do modo CCP1 - Compare/Capture/PWM:
 0000 = Modo desligado.
 0100 = Capture ligado para borda de descida com *prescaler* de 1:1.
 0101 = Capture ligado para borda de subida com *prescaler* de 1:1.
 0110 = Capture ligado para borda de subida com *prescaler* de 1:4.
 0111 = Capture ligado para borda de subida com *prescaler* de 1:16.
 1000 = Compare ligado. Pino de saída (RB3) será setado (1) quando o compare ocorrer.
 1001 = Compare ligado. Pino de saída (RB3) será zerado (0) quando o compare ocorrer.
 1010 = Compare ligado. Pino de saída (RB3) não será afetado.
 1011 = Compare ligado. Pino de saída (RB3) não será afetado. TMR1 será resetado.
 1100 = PWM ligado.
 1101 = PWM ligado.
 1110 = PWM ligado.
 1111 = PWM ligado.

Registrador: CCPR1L				Endereços: 15h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Registrador de controle do CCP1 - Parte baixa							
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

Registrador: CCPR1H				Endereços: 16h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Registrador de controle do CCP1 - Parte alta							
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

EECON1, EECON2, EEADR e EEDATA

Registrador: EECON1				Endereços: 9Ch			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
U	U	U	U	R/W	R/W	R/S	R/S
Condição em Power-On Reset (POR)							
-	-	-	-	WRERR	WREN	WR	RD
-	-	-	-	X	0	0	0

- WRERR:** Identificação de erro durante a escrita na EEPROM:
 0 = Não ocorreu erro, a escrita foi completada.
 1 = Em erro ocorreu por uma escrita não terminada (um reset pode ter ocorrido).

- WREN:** Habilitação de escrita na EEPROM (bit de segurança):
 0 = Não habilita a escrita na EEPROM.
 1 = Habilita a escrita na EEPROM.

WR:

Ciclo de escrita na EEPROM:

0 = Este bit será zerado pelo hardware quando o ciclo de escrita terminar (não pode ser zerado por software).

1 = Inicia o ciclo de escrita (deve ser setado por software).

RD:

Ciclo de leitura da EEPROM:

0 = Este bit será zerado pelo hardware quando o ciclo de leitura terminar (não pode ser zerado por software).

1 = Inicia o ciclo de leitura (deve ser setado por software).

Registrador: EECON2								Endereços: 9Dh
Registrador de proteção para escrita na EEPROM (não é implementado de fato)								
Condição em Power-On Reset (POR)								
-	-	-	-	-	-	-	-	-

Registrador: EEADR								Endereços: 9Bh
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Endereço para escrita/leitura na EEPROM								
Condição em Power-On Reset (POR)								
X	X	X	X	X	X	X	X	X

Registrador: EEDATA								Endereços: 9Ah
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Registrador de dado para escrita/leitura na EEPROM								
Condição em Power-On Reset (POR)								
X	X	X	X	X	X	X	X	X

TXSTA e RCSTA

Registrador: TXSTA								Endereços: 98h
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
R/W	R/W	R/W	R/W	U	R/W	R	R/W	
CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D	
Condição em Power-On Reset (POR)								
0	0	0	0	0	0	1	0	

CSRC: Seleção entre Master/Slave (somente modo Síncrono):

0 = Slave.

1 = Master.

TX9: Habilitação da comunicação em 9 bits para a transmissão:

0 = Transmissão em 8 bits.

1 = Transmissão em 9 bits.

- TXEN:** Habilitação da transmissão:
 0 = Transmissão desabilitada.
 1 = Transmissão habilitada. No modo síncrono, a recepção tem prioridade sobre este bit.
- SYNC:** Seleção entre modo Assíncrono/Síncrono:
 0 = Assíncrono.
 1 = Síncrono.
- BRGH:** Seleção para *Baud Rate* (somente modo Assíncrono):
 0 = *Baud Rate* baixo.
 1 = *Baud Rate* alto.
- TRMT:** Situação do registrador interno de transmissão (TSR):
 0 = TSR cheio.
 1 = TSR vazio.
- TX9D:** Valor a ser transmitido como 9º bit. Pode ser usado como paridade ou endereçamento.

Registrador: RCSTA								Endereço: 18h														
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	R/W	R/W	R/W	R/W	R	R	R								
SPEN	RX9	SREN	CREN	ADEN	FERR	OERR	RX9D	Condição em Power-On Reset (POR)														
0	0	0	0	0	0	0	X															

- SPEN:** Habilitação da USART:
 0 = USART desabilitada.
 1 = USART habilitada.
- RX9:** Habilitação da comunicação em 9 bits para a recepção:
 0 = Recepção em 8 bits.
 1 = Recepção em 9 bits.
- SREN:** Habilitação da recepção unitária (somente para modo Síncrono em Master):
 0 = Recepção unitária desabilitada.
 1 = Recepção unitária habilitada. Depois de receber um dado, desliga-se automaticamente.
- CREN:** Habilitação da recepção contínua:
 0 = Recepção contínua desabilitada.
 1 = Recepção contínua habilitada.
- ADDEN:** Habilitação do sistema de endereçamento (somente modo Assíncrono de 9 bits):
 0 = Desabilita sistema de endereçamento.
 1 = Habilita sistema de endereçamento.

FERR: Erro de *Stop bit* (somente modo Assíncrono):

0 = Não ocorreu erro. *Stop bit* = 1.

1 = Ocorreu um erro. *Stop bit* = 0 (deve ser atualizado lendo o registrador RCREG e recebendo o próximo dado válido).

OERR: Erro de muitos bytes recebidos sem nenhuma leitura:

0 = Não houve problemas de estouro do limite.

1 = Estouro do limite de 3 bytes recebidos antes da leitura de RCREG (para limpar deve-se zerar o bit CREN).

RX9D: Valor recebido no 9º bit. Pode ser usado como paridade ou endereçamento.

TXREG e RCREG

Registrador: TXREG				Endereços: 19h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Buffer para a transmissão de dados pela USART							
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

Registrador: RCREG				Endereços: 1Ah			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Buffer para a recepção de dados pela USART							
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

SPBRG

Registrador: SPBRG				Endereços: 99h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Ajuste do Baund Rate							
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

SET DE INSTRUÇÕES COMPLETO***ADDLW SOMA UMA LITERAL AO W***

Sintaxe:	ADDLW k				
Descrição:	O valor da literal passada no argumento k é somado ao valor de W e o resultado é armazenado no próprio W.				
Limites:	$0 \leq k \leq 255$				
Operação:	$(W) + (k) \rightarrow (W)$				
Status afet.:	C,DC,Z				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>11</td><td>111</td><td>kkk</td><td>kkk</td></tr></table>	11	111	kkk	kkk
11	111	kkk	kkk		
Palavras:	1				
Ciclos:	1				
Exemplo:	ADDLW 0x15 Antes da instrução: W = 0x10 Após a instrução: W = 0x25				

ADDWF SOMA ENTRE W E O REGISTRADOR F

Sintaxe:	ADDWF f,d				
Descrição:	O valor de W é somado ao valor do registrador F e o resultado é armazenado no lugar definido por d.				
Limites:	$0 \leq f \leq 127$ $d=0$ (W) ou $d=1$ (F)				
Operação:	$(W) + (f) \rightarrow (d)$				
Status afet.:	C,DC,Z				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>00</td><td>0111</td><td>Dfff</td><td>ffff</td></tr></table>	00	0111	Dfff	ffff
00	0111	Dfff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo:	ADDWF FSR,W Antes da instrução: W = 0x17 FSR = 0xC2 Após a instrução: W = 0xD9 FSR = 0xC2				

ANDLW OPERAÇÃO “E” ENTRE UMA LITERAL E W

Sintaxe: ANDLW k

Descrição: Executa um “E” lógico entre o valor da literal passado no argumento k e o valor de W. O resultado é armazenado no próprio W.

Limites: $0 \leq k \leq 255$

Operação: (W) .AND. (k) \rightarrow (W)

Status afet.: Z

Encoding:

11	1001	kkkk	kkkk
----	------	------	------

Palavras: 1

Ciclos: 1

Exemplo: ANDLW 0x5F

Antes da instrução: W = 0xA3

Após a instrução: W = 0x03

ANDWF OPERAÇÃO “E” ENTRE W E F

Sintaxe: ANDWF f,d

Descrição: Executa um “E” lógico entre o valor de W e o valor do registrador F. O resultado é armazenado no lugar definido por d.

Limites: $0 \leq f \leq 127$

d=0 (W) ou d=1 (F)

Operação: (W) .AND. (f) \rightarrow (d)

Status afet.: Z

Encoding:

00	0101	dfff	ffff
----	------	------	------

Palavras: 1

Ciclos: 1

Exemplo: ANDWF FSR,F

Antes da instrução: W = 0x17

FSR = 0xC2

Após a instrução: W = 0x17

FSR = 0x02

BCF LIMPA UM BIT DO REGISTRADOR F

Sintaxe:	BCF f,b				
Descrição:	O bit de número b do registrador f será zerado (clear).				
Limites:	$0 \leq f \leq 127$ $0 \leq b \leq 7$				
Operação:	$0 \rightarrow (f,b)$				
Status afet.:	Nenhum				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>01</td> <td>00bb</td> <td>bfff</td> <td>ffff</td> </tr> </table>	01	00bb	bfff	ffff
01	00bb	bfff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo:	BCF FLAG,5 Antes da instrução: FLAG = B'11111111' Após a instrução: FLAG = B'11011111'				

BTFSC TESTA O BIT DE F, PULA SE ZERO

Sintaxe:	BTFS C f,b				
Descrição:	Se o bit b do registrador f for 1, então a próxima linha será executada. Caso ele seja 0, a próxima linha será pulada. Neste caso, a instrução leva dois ciclos.				
Limites:	$0 \leq f \leq 127$ $0 \leq b \leq 7$				
Operação:	Pula se $(f,b) = 0$				
Status afet.:	Nenhum				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>01</td> <td>10bb</td> <td>bfff</td> <td>ffff</td> </tr> </table>	01	10bb	bfff	ffff
01	10bb	bfff	ffff		
Palavras:	1				
Ciclos:	1 ou 2				
Exemplo:	BTFSC LED GOTO APAGA_LED GOTO ACENDE_LED Antes da instrução: PC = Linha 1 Após a instrução: Se LED = 1 PC = Linha 2 Se LED = 0 PC = Linha 3				

BSF **SETA UM BIT DO REGISTRADOR F**

Sintaxe:	BSF f,b				
Descrição:	O bit de número b do registrador f será setado (set).				
Limites:	$0 \leq f \leq 127$				
	$0 \leq b \leq 7$				
Operação:	$1 \rightarrow (f,b)$				
Status afet.:	Nenhum				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>01</td> <td>01bb</td> <td>bfff</td> <td>ffff</td> </tr> </table>	01	01bb	bfff	ffff
01	01bb	bfff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo:	BSF FLAG,5 Antes da instrução: FLAG = B'00000000' Após a instrução: FLAG = B'00100000'				

BTFSS **TESTA O BIT DE F, PULA SE UM**

Sintaxe:	BTFSS f,b				
Descrição:	Se o bit b do registrador f for 0, então a próxima linha será executada. Caso ele seja 1, a próxima linha será pulada. Neste caso, a instrução leva dois ciclos.				
Limites:	$0 \leq f \leq 127$				
	$0 \leq b \leq 7$				
Operação:	Pula se $(f,b) = 1$				
Status afet.:	Nenhum				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>01</td> <td>11bb</td> <td>bfff</td> <td>ffff</td> </tr> </table>	01	11bb	bfff	ffff
01	11bb	bfff	ffff		
Palavras:	1				
Ciclos:	1 ou 2				
Exemplo:	BTFSS LED GOTO ACENDE_LED GOTO APAGA_LED Antes da instrução: PC = Linha 1 Após a instrução: Se LED = 0 PC = Linha 2 Se LED = 1 PC = Linha 3				

CALL	CHAMA UMA SUB-ROTINA				
Sintaxe:	CALL k				
Descrição:	Chama a sub-rotina representada por k, que é um endereço da memória de programação, mas normalmente é representado por um nome (label). Antes do desvio, o endereço de retorno (PC+1) é armazenado na pilha.				
Limites:	$0 \leq k \leq 2047$				
Operação:	(PC) + 1 → TOS (k) → (PC,10:0) (PCLATH,4:3) → (PC,12:11)				
Status afet.:	Nenhum				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>10</td> <td>0kkk</td> <td>kkkk</td> <td>kkkk</td> </tr> </table>	10	0kkk	kkkk	kkkk
10	0kkk	kkkk	kkkk		
Palavras:	1				
Ciclos:	2				
Exemplo:	CALL DELAY				
	Antes da instrução: PC = Linha 1				
	Após a instrução: PC = DELAY				

CLRF	LIMPA O REGISTRADOR F				
Sintaxe:	CLRF f				
Descrição:	Limpa o valor do registrador f e o bit Z é setado.				
Limites:	$0 \leq f \leq 127$				
Operação:	$0 \rightarrow (f)$ $1 \rightarrow Z$				
Status afet.:	Z				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>00</td> <td>0001</td> <td>1fff</td> <td>ffff</td> </tr> </table>	00	0001	1fff	ffff
00	0001	1fff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo:	CLRF FLAG				
	Antes da instrução: FLAG = B'11111111'				
	Após a instrução: FLAG = B'00000000'				
	Z= 1				

CLRW **LIMPA O REGISTRADOR W**

Sintaxe: CLRW
Descrição: Limpa o valor do registrador W e o bit Z é setado.
Limites: Nenhum
Operação: 0 → (W)
 1 → Z
Status afet.: Z
Encoding:

00	0001	0000	0011
----	------	------	------

Palavras: 1
Ciclos: 1
Exemplo: CLRW
Antes da instrução: W = B'11111111'
Após a instrução: W = B'00000000'
Z = 1

CLRWDT **LIMPA O WATCHDOG TIMER**

Sintaxe: CLRWDT
Descrição: Limpa o valor do contador WDT, resetando o valor do contador de prescaler. Os bits /TO e /PD são setados.
Limites: Nenhum
Operação: 0 → (WDT)
 0 → WDT prescaler
 1 → /TO
 1 → /PD
Status afet.: /TO, /PD
Encoding:

00	0000	0110	0100
----	------	------	------

Palavras: 1
Ciclos: 1
Exemplo: CLRWDT
Antes da instrução: WDT cont. = ?
 WDT ps = ?
Após a instrução: WDT cont. = 0
 WDT ps = 0
 TO = 1
 PD = 1

COMF	COMPLEMENTO DE F				
Sintaxe:	COMF f,d				
Descrição:	Calcula o complemento do registrador f. O resultado é armazenado no lugar definido por d.				
Limites:	$0 \leq f \leq 127$				
	d=0 (W) ou d=1 (F)				
Operação:	(f) \rightarrow (d)				
Status afet.:	Z				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>00</td> <td>1001</td> <td>Dfff</td> <td>ffff</td> </tr> </table>	00	1001	Dfff	ffff
00	1001	Dfff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo:	COMF REG1,F Antes da instrução: REG1 = 0x13 Após a instrução: REG1 = 0xEC				
DECF	DECREMENTA O REGISTRADOR F				
Sintaxe:	DECF f,d				
Descrição:	Decrementa uma unidade do valor do registrador F. O resultado é armazenado no lugar definido por d.				
Limites:	$0 \leq f \leq 127$				
	d=0 (W) ou d=1 (F)				
Operação:	(f) - 1 \rightarrow (d)				
Status afet.:	Z				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>00</td> <td>0011</td> <td>dfff</td> <td>ffff</td> </tr> </table>	00	0011	dfff	ffff
00	0011	dfff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo:	DECF CNT,F Antes da instrução: CNT = 0x01 Z = 0 Após a instrução: CNT = 0x00 Z = 1				

DECFSZ DECREMENTA F, PULANDO SE ZERO

Sintaxe:	DECFSZ f,d				
Descrição:	Decrementa uma unidade do valor do registrador F. O resultado é armazenado no lugar definido por d. Se o resultado da operação for zero, a linha seguinte é pulada. Quando isso acontece, a instrução ocupa dois ciclos.				
Limites:	$0 \leq f \leq 127$ $d=0$ (W) ou $d=1$ (F)				
Operação:	$(f) - 1 \rightarrow (d)$ Pula a próxima linha se resultar em 0.				
Status afet.:	Nenhum				
Encoding:	<table border="1"><tr><td>00</td><td>1011</td><td>dfff</td><td>ffff</td></tr></table>	00	1011	dfff	ffff
00	1011	dfff	ffff		
Palavras:	1				
Ciclos:	1 ou 2				
Exemplo:	DECFSZ CNT,F GOTO CONTINUA GOTO ACABOU Antes da instrução: PC = Linha 1 Após a instrução: Se $CNT-1 \neq 0$ PC = Linha 2 Se $CNT-1 = 0$ PC = Linha 3 CNT = CNT-1				

GOTO DESVIA PARA UM OUTRO ENDEREÇO

Sintaxe:	GOTO k				
Descrição:	Desvia para um outro ponto representado por k, que é um endereço da memória de programação, mas normalmente é representado por um nome (label).				
Limites:	$0 \leq k \leq 2047$				
Operação:	$(k) \rightarrow (PC,10:0)$ $(PCLATH,4:3) \rightarrow (PC,12:11)$				
Status afet.:	Nenhum				
Encoding:	<table border="1"><tr><td>10</td><td>1kkk</td><td>kkkk</td><td>kkkk</td></tr></table>	10	1kkk	kkkk	kkkk
10	1kkk	kkkk	kkkk		
Palavras:	1				
Ciclos:	2				
Exemplo:	GOTO CONTINUA Antes da instrução: PC = Linha 1 Após a instrução: PC = CONTINUA				

INCF

INCREMENTA O REGISTRADOR F

Sintaxe:	INCF f,d				
Descrição:	Incrementa uma unidade no valor do registrador F. O resultado é armazenado no lugar definido por d.				
Limites:	$0 \leq f \leq 127$				
	d=0 (W) ou d=1 (F)				
Operação:	(f) + 1 → (d)				
Status afet.:	Z				
Encoding:	<table border="1"><tr><td>00</td><td>1010</td><td>dfff</td><td>ffff</td></tr></table>	00	1010	dfff	ffff
00	1010	dfff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo:	INCF CNT,F Antes da instrução: CNT = 0xFF Z = 0 Após a instrução: CNT = 0x00 Z = 1				

INCFSZ

INCREMENTA F, PULANDO SE ZERO

Sintaxe:	INCFSZ f,d				
Descrição:	Incrementa uma unidade no valor do registrador F. O resultado é armazenado no lugar definido por d. Se o resultado da operação for zero, a linha seguinte é pulada. Quando isso acontece, a instrução ocupa dois ciclos.				
Limites:	$0 \leq f \leq 127$				
	d=0 (W) ou d=1 (F)				
Operação:	(f) + 1 → (d) Pula a próxima linha se resultar em 0.				
Status afet.:	Nenhum				
Encoding:	<table border="1"><tr><td>00</td><td>1111</td><td>Dfff</td><td>ffff</td></tr></table>	00	1111	Dfff	ffff
00	1111	Dfff	ffff		
Palavras:	1				
Ciclos:	1 ou 2				
Exemplo:	INCFSZ CNT,F GOTO CONTINUA GOTO ACABOU Antes da instrução: PC = Linha 1 Após a instrução: Se CNT+1 ≠ 0 PC = Linha 2 Se CNT+1 = 0 PC = Linha 3 CNT = CNT+1				

IORLW **OPERAÇÃO “OU” ENTRE UMA LITERAL E W**

Sintaxe: IORLW k

Descrição: Executa um “OU” lógico entre o valor da literal passado no argumento k e o valor de W. O resultado é armazenado no próprio W.

Limites: $0 \leq k \leq 255$

Operação: $(W) .OR. (k) \rightarrow (W)$

Status afet.: Z

Encoding:	11	1000	kkkk	kkkk
-----------	----	------	------	------

Palavras: 1

Ciclos: 1

Exemplo: IORLW 0x35

Antes da instrução: W = 0x9A

Após a instrução: W = 0xBF

IORWF **OPERAÇÃO “OU” ENTRE W E F**

Sintaxe: IORWF f,d

Descrição: Executa um “OU” lógico entre o valor de W e o valor do registrador F. O resultado é armazenado no lugar definido por d.

Limites: $0 \leq f \leq 127$

$d=0$ (W) ou $d=1$ (F)

Operação: $(W) .OR. (f) \rightarrow (d)$

Status afet.: Z

Encoding:	00	0100	dfff	ffff
-----------	----	------	------	------

Palavras: 1

Ciclos: 1

Exemplo: IORWF REG,W

Antes da instrução: REG = 0x13

W = 0x91

Após a instrução: REG = 0x13

W = 0x93

MOVLW **MOVE UMA LITERAL PARA W**

Sintaxe: MOVLW k

Descrição: Move o valor de uma literal para o registrador W.

Limites: $0 \leq k \leq 255$

Operação: $(k) \rightarrow (W)$

Status afet.: Nenhum

Encoding:

11	00xx	kkkk	kkkk
----	------	------	------

Palavras: 1

Ciclos: 1

Exemplo: MOVLW 0x5A
Antes da instrução: W = ?
Após a instrução: W = 0x5A

MOVF **MOVE O VALOR DE F PARA O DESTINO D**

Sintaxe: MOVF f,d

Descrição: Move (copia) o valor do registrador F para o local determinado pelo destino d.

Limites: $0 \leq f \leq 127$
d=0 (W) ou d=1 (F)

Operação: $(f) \rightarrow (d)$

Status afet.: Z

Encoding:

00	1000	Dfff	ffff
----	------	------	------

Palavras: 1

Ciclos: 1

Exemplo: MOVF FSR,W
Antes da instrução: W = ?
 FSR = 0 x 23
Após a instrução: W = 0 x 23
 FSR = 0 x 23

MOVWF *MOVE O VALOR DE W PARA F*

Sintaxe:	MOVWF f				
Descrição:	Move (copia) o valor do registrador W para o registrador f.				
Limites:	$0 \leq f \leq 127$				
Operação:	(W) \rightarrow (f)				
Status afet.:	Nenhum				
Encoding:	<table border="1"><tr><td>00</td><td>0000</td><td>1fff</td><td>ffff</td></tr></table>	00	0000	1fff	ffff
00	0000	1fff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo:	MOVWF OPTION Antes da instrução: OPTION = 0xFF W = 0x4F Após a instrução: OPTION = 0x4F W = 0x4F				

NOP *NÃO EXECUTA NADA*

Sintaxe:	NOP				
Descrição:	Esta instrução é usada somente para perder tempo, pois ela não executa operação nenhuma.				
Limites:	Nenhum				
Operação:	Nenhuma				
Status afet.:	Nenhum				
Encoding:	<table border="1"><tr><td>00</td><td>0000</td><td>0xx0</td><td>0000</td></tr></table>	00	0000	0xx0	0000
00	0000	0xx0	0000		
Palavras:	1				
Ciclos:	1				
Exemplo:	NOP				

RETFIE	RETORNA DA INTERRUPÇÃO				
Sintaxe:	RETFIE				
Descrição:	Retorna da interrupção, recuperando o último endereço da pilha e setando o bit GIE.				
Limites:	Nenhum				
Operação:	TOS → (PC) 1 → GIE				
Status afet.:	Nenhum				
Encoding:	<table border="1"><tr><td>00</td><td>0000</td><td>0000</td><td>1001</td></tr></table>	00	0000	0000	1001
00	0000	0000	1001		
Palavras:	1				
Ciclos:	2				
Exemplo:	RETFIE Após a instrução: PC = TOS GIE = 1				
RETLW	RETORNA COM UMA LITERAL EM W				
Sintaxe:	RETLW k				
Descrição:	Retorna de uma sub-rotina, recuperando o último endereço da pilha e colocando o valor passado por k em W.				
Limites:	$0 \leq k \leq 255$				
Operação:	(k) → (W) TOS → (PC)				
Status afet.:	Nenhum				
Encoding:	<table border="1"><tr><td>11</td><td>01xx</td><td>kkkk</td><td>kkkk</td></tr></table>	11	01xx	kkkk	kkkk
11	01xx	kkkk	kkkk		
Palavras:	1				
Ciclos:	2				
Exemplo:	RETLW 0x50 Após a instrução: PC = TOS W = 0x50				

RETURN RETORNA DE UMA SUB-ROTINA

Sintaxe:	RETURN				
Descrição:	Retorna de uma sub-rotina, recuperando o último endereço da pilha.				
Limites:	Nenhum				
Operação:	TOS → (PC)				
Status afet.:	Nenhum				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>00</td> <td>0000</td> <td>0000</td> <td>1000</td> </tr> </table>	00	0000	0000	1000
00	0000	0000	1000		
Palavras:	1				
Ciclos:	2				
Exemplo:	RETURN Após a instrução: PC = TOS				

RLF ROTACIONA F UM BIT PARA A ESQUERDA

Sintaxe:	RLF f,d				
Descrição:	Rotaciona o registrador F um bit para a esquerda. O valor de carry é colocado no bit 0, e depois o valor do bit 7 é colocado em carry. O resultado é armazenado no lugar definido por d.				
Limites:	$0 \leq f \leq 127$ $d=0$ (W) ou $d=1$ (F)				
Operação:	Conforme descrição (vide Figura 11.3).				
Status afet.:	C				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>00</td> <td>1101</td> <td>dfff</td> <td>ffff</td> </tr> </table>	00	1101	dfff	ffff
00	1101	dfff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo:	RLF REG1,F Antes da instrução: C = 0 REG1 = B'11100110' Após a instrução: C = 1 REG1 = B'11001100'				

RRF

ROTACIONA F UM BIT PARA A DIREITA

Sintaxe:	RRF f,d				
Descrição:	Rotaciona o registrador F um bit para a direita. O valor de carry é colocado no bit 7, e depois o valor do bit 0 é colocado em carry. O resultado é armazenado no lugar definido por d.				
Limites:	$0 \leq f \leq 127$				
	$d=0$ (W) ou $d=1$ (F)				
Operação:	Conforme descrição (vide figura 11.4).				
Status afet.:	C				
Encoding:	<table border="1"><tr><td>00</td><td>1100</td><td>dfff</td><td>ffff</td></tr></table>	00	1100	dfff	ffff
00	1100	dfff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo:	RRF REG1,F Antes da instrução: C = 0 REG1 = B'11100110' Após a instrução: C = 0 REG1 = B'01110011'				

SLEEP

ENTRA EM MODO SLEEP

Sintaxe:	SLEEP				
Descrição:	Coloca o microcontrolador em modo sleep.				
Limites:	Nenhum				
Operação:	$0 \rightarrow$ WDT $0 \rightarrow$ WDT prescaler $1 \rightarrow$ /TO $0 \rightarrow$ /PD /TO, /PD				
Status afet.:					
Encoding:	<table border="1"><tr><td>00</td><td>0000</td><td>0110</td><td>0011</td></tr></table>	00	0000	0110	0011
00	0000	0110	0011		
Palavras:	1				
Ciclos:	1				
Exemplo:	SLEEP				

SUBLW **SUBTRAI O VALOR DE W DE UMA LITERAL**

Sintaxe: SUBLW k

Descrição: Subtrai o valor de W da constante passada por k. O resultado é armazenado no próprio W.

Limites: $0 \leq k \leq 255$

Operação: $(k) - (W) \rightarrow (W)$

Status afet.: C, DC, Z

Encoding:	11	110x	kkkk	kkkk
-----------	----	------	------	------

Palavras: 1

Ciclos: 1

Exemplo1: SUBLW 0x02

Antes da instrução: W = 1

C = ?

Após a instrução: W = 1

C = 1 (positivo)

Exemplo2: SUBLW 0x02

Antes da instrução: W = 2

C = ?

Após a instrução: W = 0

C = 1 (zero)

Exemplo3: SUBLW 0x02

Antes da instrução: W = 3

C = ?

Após a instrução: W = 255

C = 0 (negativo)

SUBWF

SUBTRAI O VALOR DE W DO REGISTRADOR F

Sintaxe:	SUBWF f,d				
Descrição:	Subtrai do registrado F o valor de W. O resultado é armazenado no lugar definido por d.				
Limites:	$0 \leq f \leq 127$				
	d=0 (W) ou d=1 (F)				
Operação:	$(f) - (W) \rightarrow (d)$				
Status afet.:	C, DC, Z				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>00</td><td>0010</td><td>ffff</td><td>ffff</td></tr></table>	00	0010	ffff	ffff
00	0010	ffff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo1:	SUBWF REG,F Antes da instrução: REG = 3 W = 2 C = ? Após a instrução: REG = 1 W = 2 C = 1 (positivo)				
Exemplo2:	SUBLW REG,F Antes da instrução: REG = 2 W = 2 C = ? Após a instrução: REG = 0 W = 2 C = 1 (zero)				
Exemplo1:	SUBLW REG,F Antes da instrução: REG = 1 W = 2 C = ? Após a instrução: REG = 255 W = 2 C = 0 (negativo)				

SWAPF INVERSÃO DO REGISTRADOR F

Sintaxe:	SWAPF f,d				
Descrição:	Inverte a parte alta (bits de 4 a 7) com a parte baixa (bits de 0 a 3) do registrador F. O resultado é armazenado no lugar definido por d.				
Limites:	$0 \leq f \leq 127$ d=0 (W) ou d=1 (F)				
Operação:	(f,3:0) \rightarrow (d,7:4) (f,7:4) \rightarrow (d,3:0)				
Status afet.:	Nenhum				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>00</td> <td>1110</td> <td>dfff</td> <td>ffff</td> </tr> </table>	00	1110	dfff	ffff
00	1110	dfff	ffff		
Palavras:	1				
Ciclos:	1				
Exemplo1:	SWAPF REG,F Antes da instrução: REG = 0xA5 Após a instrução: REG = 0x5A				

XORLW OPERAÇÃO “OU EXCLUSIVO” ENTRE UMA LITERAL E W

Sintaxe:	XORLW k				
Descrição:	Executa um “OU” lógico exclusivo entre o valor da literal passado no argumento k e o valor de W. O resultado é armazenado no próprio W.				
Limites:	$0 \leq k \leq 255$				
Operação:	(W) .XOR. (k) \rightarrow (W)				
Status afet.:	Z				
Encoding:	<table border="1" style="display: inline-table; vertical-align: middle;"> <tr> <td>11</td> <td>1010</td> <td>kkkk</td> <td>kkkk</td> </tr> </table>	11	1010	kkkk	kkkk
11	1010	kkkk	kkkk		
Palavras:	1				
Ciclos:	1				
Exemplo:	XORLW 0xAF Antes da instrução: W = 0xB5 Após a instrução: W = 0x1A				

XORWF OPERAÇÃO “OU EXCLUSIVO” ENTRE W E F

Sintaxe: XORWF f,d

Descrição: Executa um “OU” lógico exclusivo entre o valor de W e o valor do registrador F. O resultado é armazenado no lugar definido por d.

Limites: $0 \leq f \leq 127$

$d=0$ (W) ou $d=1$ (F)

Operação: (W) .XOR. (f) \rightarrow (d)

Status afet.: Z

Encoding:

00	0110	dfff	ffff
----	------	------	------

Palavras: 1

Ciclos: 1

Exemplo: XORWF REG,F

Antes da instrução: W = 0xB5

REG = 0xAF

Após a instrução: W = 0xB5

REG = 0x1A

DIRETRIZES DA LINGUAGEM MPASM

BADRAM - CONFIGURA REGIÕES DA RAM NÃO DISPONÍVEIS

Sintaxe

_BADRAM <expr>[-<expr>[, <expr>-[<expr>]]]

Descrição

Determina blocos de memória RAM que não podem ser utilizados pelo microcontrolador. Os valores de <expr> devem estar sempre dentro do limite imposto pela diretriz _MAXRAM. Caso o programa tente utilizar um endereço englobado por esses limites, uma mensagem de erro será gerada. O programador também não precisa se preocupar com essa diretriz, pois ela está definida nos arquivos de INCLUDE fornecidos pela Microchip.

Exemplo

Veja exemplo de _MAXRAM

Veja também

_MAXRAM

BANKISEL - GERA CÓDIGO PARA ACERTAR BANCO DE MEMÓRIA (ACESSO INDIRETO)

Sintaxe

BANKISEL <nome>

Descrição

Essa diretriz é utilizada para acertar automaticamente o banco de memória para acesso indireto. Na verdade, o compilador irá gerar o código necessário para acertar o banco. Para PICs de 14 bits, serão utilizadas instruções para setar ou limpar o bit IRP do registrador de STATUS. Já para os PICs de 16 bits, as instruções MOVLB e

MOVLR serão implementadas. O argumento <nome> representa a variável com a qual se trabalhará e que deve servir de referência para a seleção do banco.

Exemplo

```
MOVLW VAR1
MOVWF FSR
BANKISEL    VAR1
...
MOVWF INDF
```

Veja também

PAGESEL, BANKSEL

BANKSEL - GERA CÓDIGO PARA ACERTAR BANCO DE MEMÓRIA (ACESSO DIRETO)

Sintaxe

```
BANKSEL    <nome>
```

Descrição

Esta diretriz é utilizada para acertar automaticamente o banco de memória para acesso direto. Na verdade, o compilador irá gerar o código necessário para acertar o banco. Para PICs de 12 bits, as instruções para acertar os bits do FSR serão inseridas no código. Para PICs de 14 bits, serão utilizadas instruções para setar ou limpar os bits RP0 e RP1 do registrador de STATUS. Já para os PICs de 16 bits, as instruções MOVLB e MOVLR serão implementadas. Entretanto, se o PIC em uso contém somente um banco de RAM, nenhuma instrução adicional é gerada. O argumento <nome> representa a variável com a qual se trabalhará e que deve servir de referência para a seleção do banco.

Exemplo

```
MOVLW .10
BANKSEL    VAR1 *
MOVWF VAR1
```

Veja também

PAGESEL, BANKISEL

CBLOCK - DEFINE UM BLOCO DE CONSTANTES

Sintaxe

```
CBLOCK [<expr>]
        <nome> [:<incremento>] [,<nome> [:<incremento>]]
ENDC
```

Descrição

Define uma série de constantes. O primeiro <nome> é associado ao valor de <expr>. Já o segundo é associado ao valor seguinte e assim sucessivamente. Caso não seja determinado o valor de <incremento>, a próxima constante receberá o valor imediatamente seguinte, isto é, incremento unitário. Caso contrário, a próxima constante receberá o valor da constante anterior mais o <incremento>. A definição de constantes deve ser terminada pela diretriz ENDC.

Exemplo

CBLOCK 0X20

```
    W_TEMP           ;W_TEMP=0X20
    STATUS_TEMP      ;STATUS_TEMP=0X21
    TEMP1, TEMP2    ;TEMP1=0X22, TEMP2=0X23
    END:0, END_H, END_L ;END=0X24, END_H=0X24, END_L=0X25
    CODIGO:2        ;CODIGO=0X26
    CONTA            ;CONTA=0X28
```

ENDC

Veja também

ENDC

CODE - DECLARA O INÍCIO DE UM BLOCO DE PROGRAMA

Sintaxe

[<nomel>] CODE [<ROM endereço>]

Descrição

Usado para objetos. Serve para declarar o início de um bloco de programa. Caso o argumento <nome> não seja definido, o bloco será chamado CODE. O endereço inicial é passado pelo argumento <ROM endereço>. Se este não for especificado, o compilador assume o endereço zero.

Exemplo

```
RESET CODE      H'01FF'
                GOTO      START
```

Veja também

IDATA, UDATA, UDATA_OVR, UDATA_SHR, EXTERN, GLOBAL

CONFIG - CONFIGURA OS DADOS PARA GRAVAÇÃO DO MICROCONTROLADOR

Sintaxe

CONFIG <expr>

Descrição

Utilizado para configurar previamente os dados para a gravação do PIC. Para facilitar a vida do programador, a Microchip já definiu uma série de símbolos para essas opções nos arquivos de INCLUDE de cada modelo de PIC. Basta dar uma conferida nesses arquivos para saber quais os símbolos pertinentes ao tipo de oscilador, WDT, etc. A <expr> deve ser montada pela junção destes símbolos por meio do operador &.

Exemplo

CONFIG _WDT_ON & _XT_OSC & _CP_OFF

Veja também

LIST, PROCESSOR, IDLOCS

CONSTANT - DEFINE UMA CONSTANTE

Sintaxe

CONSTANT <nome> = <expr> [, <nome> = <expr>]

Descrição

Define uma constante para ser utilizada nas expressões que serão interpretadas pelo compilador. Como o próprio nome diz, o valor determinado por <expr> é associado de maneira definitiva ao <nome>, não podendo mais ser alterado no decorrer do programa.

Exemplo

CONSTANT TAM_NOME = 10 , TAM_CODIGO = 5

.

.

.

CONSTANT TAM_TOTAL = TAM_NOME + TAM_CODIGO

Veja também

SET, VARIABLE

DATA - PREENCHE A MEMÓRIA COM NÚMEROS OU TEXTOS

Sintaxe

```
DATA <expr>[,<expr>, . . . ,<expr>]  
DATA "texto"[, "texto", . . . , "texto"]
```

Descrição

Preenche a memória com o valor determinado por `<expr>` ou "texto", começando na posição atual e avançando a quantidade de posições necessárias para caber o dado. As expressões ou caracteres unitários são colocados um em cada posição de memória. Já os textos ASCII são arquivados com dois caracteres para cada posição da Memória, sendo o primeiro no byte mais significativo. Se o texto possui um número ímpar de caracteres, um zero é escrito no último byte.

Exemplo

```
DATA 1,2,CONTA  
DATA 'N'  
DATA "TEXTO DE EXEMPLO"
```

Veja também

DW, DB, DE, DT

DB - PREENCHE A MEMÓRIA BYTE A BYTE

Sintaxe

```
DB <expr>[,<expr>, . . . ,<expr>]
```

Descrição

Preenche a memória com o valor determinado por `<expr>`, começando na posição atual e guardando os dados a cada byte, avançando a quantidade de posições necessárias para caber todas as `<expr>`.

Exemplo

```
DB .1,'T',0x0f,CONTA,'s'
```

Veja também

DATA, DW, DE, DT

DE - PREENCHE A MEMÓRIA EEPROM BYTE A BYTE

Sintaxe

```
DE      <expr>[,<expr>, ...,<expr>]  
DE      "texto" [, "texto", ..., "texto"]
```

Descrição

Preenche a memória EEPROM com o valor determinado por <expr> ou "texto", começando na posição atual e guardando os dados a cada byte, avançando a quantidade de posições necessárias para caber todas as <expr>. No caso do "texto", cada caractere será gravado em um byte. Caso essa diretriz seja utilizada para gravar dados na área de programação, os bits mais significativos (acima do 7) serão zerados.

Exemplo

```
ORG      H'2100'          ;aponta para o início da EEPROM  
DE       "Exemplo, V1.0"
```

Veja também

DATA, DW, DB, DT

#DEFINE - DEFINE UMA SUBSTITUIÇÃO DE TEXTO

Sintaxe

```
#DEFINE    <nome> [<texto>]
```

Descrição

Essa diretriz define uma substituição de texto. Sempre que o compilador encontrar <nome>, ele será substituído pelo <texto> associado a ele. Ao contrário da diretriz EQU, que só pode associar valores, essa diretriz pode associar qualquer coisa ao nome, inclusive um comando ou o endereço de um bit. Símbolos criados desta maneira não podem ser monitorados pelas ferramentas do MPLab (simulador). Definindo um nome sem um <texto> associado, ele poderá ser usado com a diretriz IFDEF.

Exemplo

```
#DEFINE    LED           PORTA, 0  
#DEFINE    LED_ON        BSF            LED  
#DEFINE    LED_OFF       BCF            LED
```

Veja também

IFDEF, IFNDEF, #UNDEFINE

Sintaxe

DT <expr> [, <expr>, . . . , <expr>]
DT "texto"

Descrição

Preenche a memória com uma série de instruções RETLW, uma para cada <expr> ou caractere do "texto", começando na posição atual e avançando a quantidade de posições necessárias para caber todas as instruções.

Exemplo

DT "MOSAICO"
DT VALOR1, VALOR2, VALOR3

Veja também

DATA, DE, DB, DW

DW - PREENCHE A MEMÓRIA PALAVRA A PALAVRA

Sintaxe

DW <expr> [, "texto", . . . , <expr>]

Descrição

Preenche a memória com o valor determinado por <expr> ou "texto", começando na posição atual e avançando a quantidade de posições necessárias para caber todos os dados. As expressões ou caracteres unitários são colocados um em cada posição de memória. Já os textos ASCII são arquivados com dois caracteres para cada posição, sendo o primeiro no byte mais significativo. Se o texto possuir um número ímpar de caracteres, um zero é escrito no último byte.

Exemplo

DW 39, "MOSAICO"

Veja também

DATA, DE, DB, DT

ELSE - BLOCO ALTERNATIVO PARA TESTE CONDICIONAL

Sintaxe

ELSE

Descrição

Usado em conjunto com as diretrizes IF, IFDEF e IFNDEF para executar um bloco específico no caso do teste condicional apresentar um resultado negativo.

Exemplo

Veja o exemplo de IF.

Veja também

IF, IFDEF, IFNDEF, ENDIF

END - FIM DO PROGRAMA

Sintaxe

END

Descrição

Usado para determinar o fim do programa. Essa diretriz é obrigatória na última linha do código-fonte.

Exemplo

INICIO

(corpo do programa)

END

ENDC - FINALIZA UM BLOCO DE CONSTANTES

Sintaxe

ENDC

Descrição

Finaliza o bloco de constantes iniciado por CBLOCK.

Exemplo

Veja o exemplo de CBLOCK.

Veja também

CBLOCK

Sintaxe

ENDIF

Descrição

Usado em conjunto com as diretrizes IF, IFDEF, IFNDEF e ELSE para finalizar os blocos de códigos que serão executados de acordo com os testes condicionais.

Exemplo

Veja o exemplo de IF.

Veja também

IF, IFDEF, IFNDEF, ELSE

ENDM - FINALIZA O BLOCO DE UMA MACRO

Sintaxe

ENDM

Descrição

Finaliza o bloco de uma macro.

Exemplo

veja o exemplo de MACRO.

Veja também

EXITM, MACRO

ENDW - FIM DO BLOCO DE LOOP

Sintaxe

ENDW

Descrição

Usado em conjunto com a diretriz WHILE para finalizar o bloco de códigos que será repetido enquanto o teste de WHILE for verdadeiro.

Exemplo

veja o exemplo de WHILE.

Veja também

WHILE

EQU - DEFINE UMA SUBSTITUIÇÃO

Sintaxe

<nome> EQU <expr>

Descrição

O <nome> será substituído pelo valor determinado por <expr> sempre que for encontrado. Este conceito é totalmente similar ao de constantes.

Exemplo

```
TEMPO EQU .4
```

Veja também

SET, CONSTANT

ERROR - GERA UMA MENSAGEM DE ERRO

Sintaxe

ERROR "<texto>"

Descrição

Gera uma mensagem de erro, que será mostrada como um erro no relatório gerado durante a compilação. O texto pode ter até 80 caracteres.

Exemplo

```
IFNDEF __PIC16F84
    ERROR "ATENÇÃO: o PIC escolhido não é o 16F84"
ENDIF
```

Veja também

MESSG

ERRORLEVEL - DETERMINA O NÍVEL DE GERAÇÃO DAS MENSAGENS

Sintaxe

ERRORLEVEL 0/1/2/+/- <msg_num>

Descrição

Determina que tipo de mensagens deve ser mostrado durante a compilação:

Nível	O que será mostrado
0	Mensagens, alertas e erros (tudo).
1	Alertas e erros.
2	Somente erros.
+<msg_num>	Habilita a mensagem de número <msg_num>.
-<msg_num>	Inibe a mensagem de número <msg_num>.

Dica: Os números das mensagens, alertas e erros podem ser encontrados no Apêndice C do manual do compilador MPASM. Nas versões mais novas do MPLab, essa diretriz pode ter seu nível acertado e gravado no arquivo de projeto.

Exemplo

ERRORLEVEL 1, -202

Veja também

LIST

EXITM - FORÇA A SAÍDA DE UMA MACRO

Sintaxe

EXITM

Descrição

Força a saída de uma macro. Seus efeitos são similares aos da diretriz ENDM.

Exemplo

veja o exemplo de MACRO.

Veja também

ENDM, MACRO

EXPAND - EXPANDE O CÓDIGO DAS MACROS

Sintaxe

EXPAND

Descrição

Determina que todas as macros encontradas após essa diretriz serão expandidas na listagem do programa (Absolut List - Arquivo .LST).

Veja também

LIST, NOEXPAND, MACRO

EXTERN - DECLARA SÍMBOLOS DEFINIDOS EXTERNAMENTE

Sintaxe

EXTERN <nome> [, <nome>]

Descrição

Usado para objetos. Declara símbolos que serão definidos no código corrente, mas que são definidos como GLOBAL em outros módulos. Essa diretriz deve ser aplicada antes dos símbolos serem utilizados no código corrente. Mais de um símbolo pode ser definido na mesma linha, por meio dos argumentos <nome>.

Exemplo

EXTERN TESTE

...

CALL TESTE

Veja também

TEXT, IDATA, UDATA, UDATA_OVR, UDATA_SHR, GLOBAL

FILL - PREENCHE A MEMÓRIA COM UM VALOR OU INSTRUÇÃO

Sintaxe

FILL <expr>, <num>

Descrição

Preenche a memória com o valor determinado por <expr>, começando na posição atual e avançando <num> posições. A <expr> pode também ser uma instrução do Assembler, mas neste caso deve ser colocada entre parênteses.

Exemplo

FILL 0X1009,5 . ;Preenche 5 posições com o valor 0x1009
FILL (NOP),CONTA ;Preenche CONTA posições com a instr. NOP

Veja também

ORG, DATA, DW

GLOBAL - DECLARA SÍMBOLOS QUE PODEM SER USADOS EXTERNAMENTE

Sintaxe

GLOBAL <nome> [, <nome>]

Descrição

Usado com objetos. Declara os símbolos que serão definidos no módulo corrente, mas que poderão ser exportados para outros módulos. Essa diretriz pode ser utilizada antes ou após a definição do símbolo. Mais de um símbolo pode ser declarado na mesma linha por meio dos argumentos <nome>.

Exemplo

```
UDATA
VAR1    RES      .1
VAR2    RES      .1
GLOBAL VAR1, VAR2
```

```
CODE
SOMATRES
GLOBAL SOMATRES
ADDLW .3
RETURN
```

Veja também

TEXT, IDATA, UDATA, UDATA_OVR, UDATA_SHR, EXTERN

IDATA - DECLARA UMA SEÇÃO DE DADOS JÁ INICIALIZADOS

Sintaxe

[<nome>] IDATA · [<RAM endereço>]

Descrição

Usado com objetos. Declara o início de uma seção de dados inicializados. Se o argumento <nome> não for especificado, a seção será chamada de IDATA. O endereço inicial para os dados é definido por <RAM endereço>, ou será adotado o valor zero caso esse argumento não seja especificado. Nenhum código será gerado nesta seção. O linker irá gerar uma tabela de entrada para cada byte especificado. O usuário deverá então definir a inicialização apropriada. Essa diretriz não está disponível para PICs de 12 bits.

Exemplo

```
IDATA
LIMITEL      DW      .0
LIMITEH      DW      .300
GANHO DW     .5
FLAGS DW     .0
TEXTO DB     'MOSAICO'
```

Veja também

TEXT, UDATA, UDATA_OVR, UDATA_SHR, EXTERN, GLOBAL

IDLOCS - ESPECIFICA OS DADOS PARA GRAVAÇÃO DO ID

Sintaxe

_IDLOCS <expr>

Descrição

Utilizado para especificar previamente os dados que serão gravados nas quatro posições de ID.

Exemplo

_IDLOCS H'1234'

Veja também

LIST, PROCESSOR, _CONFIG

IF - TESTE CONDICIONAL DE UMA EXPRESSÃO

Sintaxe

IF <expr>

Descrição

Testa se a expressão definida por <expr> é verdadeira ou falsa. Caso ela seja verdadeira, então o código existente entre essa diretriz e a ENDIF será executado. Caso contrário, ele será pulado, podendo ainda ser executado o bloco relacionado à diretriz ELSE, caso ela exista.

Exemplo

CONTA = 5

IF CONTA > 3
 (rotina específica para CONTA > 3)
ELSE
 (rotina específica para CONTA ≤ 3)

ENDIF

Veja também

ENDIF, ELSE

IFDEF - TESTE CONDICIONAL DE EXISTÊNCIA DE UM SÍMBOLO

Sintaxe

IFDEF <nome>

Descrição

Testa se o símbolo especificado por <nome> já foi definido (diretriz #DEFINE, sem o argumento <texto>). Caso o símbolo já tenha sido definido, então o código existente entre essa diretriz e a ENDIF será executado. Caso contrário, ele será pulado, podendo ser executado o bloco associado à diretriz ELSE, caso ela exista.

Exemplo

```
#DEFINE      TESTE
...
IFDEF TESTE
    (rotina que será executada para teste)
ENDIF
```

Veja também

IFNDEF, #DEFINE, #UNDEFINE, ENDIF, ELSE

IFNDEF - TESTE CONDICIONAL DE NÃO-EXISTÊNCIA DE UM SÍMBOLO

Sintaxe

IFNDEF <nome>

Descrição

Testa se o símbolo especificado por <nome> não foi definido. Caso o símbolo não tenha sido definido, então o código existente entre essa diretriz e a ENDIF será executado. Caso contrário, ele será pulado, podendo ser executado o bloco associado à diretriz ELSE, caso ela exista.

Exemplo

```
#UNDEFINE   TESTE           ;garante a não-definição de TESTE
...
IFNDEF TESTE
    (rotina que será executada fora do teste)
ENDIF
```

Veja também

IFDEF, #DEFINE, #UNDEFINE, ENDIF, ELSE

Sintaxe

```
#INCLUDE      <nome_do_arquivo>
#INCLUDE      "nome_do_arquivo"
```

Descrição

Usado geralmente no início do programa, para incluir definições especificadas em arquivos auxiliares. Esses arquivos, com mesma formatação dos códigos-fonte, possuem, no entanto, somente definições, variáveis, constantes e similares, para facilitar a vida do programador. Junto com o MPLab, a Microchip fornece vários desses arquivos com a extensão .INC, um para cada tipo de PIC. Se o path for especificado em nome_do_arquivo, então o arquivo será procurado somente nesse diretório. Caso contrário, a ordem de procura será a seguinte: diretório atual, diretório dos arquivos de código-fonte e diretório do MPASM.

Exemplo

```
#INCLUDE      "C:\MOSAICO\MOSAICO.INC"
#INCLUDE      <PIC16F84.INC>
```

LST - OPÇÕES PARA A LISTAGEM DO PROGRAMA

Sintaxe

```
LST   [<opção>, <opção>]
```

Descrição

Configura uma série de opções que será utilizada para a criação da listagem de programa (arquivo com extensão .LST). As opções possíveis encontram-se na seguinte tabela:

Opção	Padrão	Descrição
b=nnn	• 8	Espaços para tabulações.
c=nnn	132	Quantidade de colunas.
f=<formato>	INHX8M	Formato do arquivo de saída. Pode ser INHX32, INHX8M ou INHX8S.
free	fixed	Utiliza formato livre.
fixed	fixed	Utiliza formato fixo.
mm=ON/OFF	On	Mostra o mapa da memória no arquivo de listagem.
n=nnn	60	Número de linhas por página.

Opção	Padrão	Descrição
p=<tipo>	Nenhum	Modelo de PIC (por exemplo: PIC16F84).
r=<radix>	Hex	Radix utilizado: HEX, DEC ou OCT.
st=ON/OFF	On	Mostra a tabela de símbolos no arquivo de listagem.
t=ON/OFF	Off	Trunca linhas muito grandes.
w=0/1/2	0	Escolhe o nível de mensagens (vide ERRORLEVEL).
x=ON/OFF	On	Expande ou não as macros no arquivo de listagem.

Dica: Os valores destas opções são representados sempre em decimal.

Exemplo

```
LIST p=PIC16F84, r=DEC, c=80
```

Veja também

NOLIST, PROCESSOR, RADIX, ERRORLEVEL, EXPAND, NOEXPAND

LOCAL - DEFINE UMA VARIÁVEL LOCAL PARA UMA MACRO

Sintaxe

```
LOCAL <nome>, [<nome>]
```

Descrição

Declara variáveis que serão utilizadas somente dentro da MACRO. Mesmo que essa variável possua o mesmo <nome> de uma outra já definida no corpo do programa, ela será tratada como uma variável totalmente nova, não afetando o valor da outra anteriormente definida.

Exemplo

```
COMP SET .10
LARG SET .20
.

TESTE MACRO TAMANHO
LOCAL COMP, LARG ;AS VARIÁVEIS LOCAIS NÃO GERAM CONFLITOS
COMP SET LARG
ENDM ;COMP=10 E LARG=20
```

Veja também

ENDM, MACRO

MACRO - DEFINE UMA MACRO

Sintaxe

<nome> MACRO [<arg1>, ..., <argx>]

Descrição

Uma macro é uma seqüência de instruções que pode ser inserida no seu código com uma simples referência ao nome dela. Por isso, normalmente as macros são utilizadas para economizar digitação de funções muito utilizadas e podem ser definidas em arquivos do tipo INCLUDE. Uma macro pode chamar outra macro de dentro dela. Os argumentos passados à macro serão substituídos no corpo do código. Para terminar uma macro, é necessária a diretriz ENDM. No entanto, ela pode ser finalizada quando é encontrada a diretriz EXITM. Variáveis utilizadas somente dentro das macros podem ser definidas como LOCAL. Maiores informações sobre este poderoso recurso podem ser obtidas no Capítulo 4 do manual do MPASM.

Exemplo

```
; (Definição)
TESTE MACRO REG
    IF      REG == 1
        EXITM
    ELSE
        ERRO    "REGISTRADOR ERRADO
    ENDIF
ENDM
; (Utilização)
TESTE TEMP
```

Veja também

ENDM, EXITM, LOCAL, IF, ELSE, ENDIF, WHILE, ENDW

MAXRAM - CONFIGURA O TAMANHO MÁXIMO DA RAM

Sintaxe

MAXRAM <expr>

Descrição

Utilizado para configurar o tamanho máximo da memória para o PIC que está sendo utilizado. A <expr> determina o último endereço de memória disponível. O programador não precisa se preocupar com essa diretriz, pois ela está definida nos arquivos de INCLUDE fornecidos pela Microchip.

Exemplo

```
; (para PIC 16F84)
__MAXRAM H'CF'
__BADRAM H'07', H'50'-H'7F', H'87'
```

Veja também

BADRAM

MESSG - GERA UMA MENSAGEM DEFINIDA PELO USUÁRIO

Sintaxe

MESSG "texto"

Descrição

Gera uma mensagem que será mostrada no relatório gerado durante a compilação. O texto pode ter até 80 caracteres.

Exemplo

```
IFNDEF __PIC16F84
    MESSG "ATENÇÃO: o PIC escolhido não é o 16F84"
ENDIF
```

Veja também

ERROR

NOEXPAND - NÃO EXPANDE O CÓDIGO DAS MACROS

Sintaxe

NOEXPAND

Descrição

Determina que todas as macros encontradas após essa diretriz não serão expandidas na listagem do programa (.LST).

Veja também

LIST, EXPAND, MACRO

NOLIST - DESLIGA A GERAÇÃO DO ARQUIVO DE LISTAGEM

Sintaxe

NOLIST

Descrição

Inibe a geração automática do arquivo de listagem (Absolut List - Arquivo .LST).

Veja também

LIST

ORG - ACERTA PONTO DA MEMÓRIA DE PROGRAMAÇÃO

Sintaxe

[<nome>] ORG <expr>

Descrição

Usado para determinar o ponto da memória de programação em que a próxima instrução será escrita. Se nenhuma diretriz ORG for colocada no programa, então ele começará a ser escrito na posição 0x00. Caso <nome> seja especificado, então o valor de <expr> será associado a ele.

Exemplo

```
END_1 ORG 0x10  
END_2 ORG END_1 + 0x10
```

PAGE - INSERE UMA QUEBRA DE PÁGINA

Sintaxe

PAGE

Descrição

Insere uma quebra de página na geração do arquivo de listagem do programa (.LST).

Veja também

LIST, TITLE, SUBTITLE, SPACE

PAGESEL - SELEÇÃO A PÁGINA DE PROGRAMAÇÃO

Sintaxe

PAGESEL <nome>

Descrição

Essa diretriz é utilizada para acertar automaticamente a página de memória de programação. Na verdade, o compilador irá gerar o código necessário para acertar a página. Para PICs de 12 bits, as instruções para acertar os bits do STATUS serão inseridas no código. Para PICs de 14 e 16 bits, serão utilizadas as instruções MOVLW e MOVWF para acertar o valor de PCLATH. Entretanto, se o PIC em uso contém somente uma página de programação, nenhuma instrução adicional é gerada. O argumento <nome> representa a rotina com a qual se trabalhará, e deve ser definida antes do uso dessa diretriz.

Exemplo

```
PAGESEL      GOTODEST  
GOTO      GOTODEST
```

Veja também

BANKSEL, BANKISEL

PROCESSOR - DETERMINA O TIPO DE PIC UTILIZADO

Sintaxe

```
PROCESSOR      <tipo>
```

Descrição

Determina o tipo de PIC que será utilizado. Nas versões mais novas do MPLab, essa diretriz não é mais necessária, pois este valor é acertado e gravado no arquivo de projeto.

Exemplo

```
PROCESSOR      PIC16F84
```

Veja também

LIST

RADIX - DETERMINA O RADIX PADRÃO

Sintaxe

```
RADIX <tipo>
```

Descrição

Determina o tipo padrão a ser considerado a todos os números encontrados no programa, quando não devidamente especificados. O valor de <tipo> pode ser: HEX, DEC ou OCT. Lembre-se de que, independentemente do RADIX padrão, qualquer número pode ser acertado por meio das predefinições:

Radix	Formato 1	Formato 2
Decimal	D'xx'	.x
Hexadecimal	H'xx'	0Xxx
Octadecimal	O'xx'	
Binário	B'xxxxxxxx'	
ASCII	A'x'	'x'

Dica: Recomendamos que as predefinições apresentadas acima sejam sempre utilizadas, para evitar problemas caso alguém altere o valor do radix em seus programas.

Nas versões mais novas do MPLab, essa diretriz não é mais necessária, pois este valor é acertado e gravado no arquivo de projeto.

Exemplo

RADIX DEC

Veja também

LIST

RES - RESERVA MEMÓRIA

Sintaxe

[<nome>] RES <tamanho>

Descrição

Reserva memória de programação para uso posterior. O bloco irá começar na posição atual e terá o tamanho especificado pelo argumento <tamanho>. A definição de <nome> poderá ser usada para referenciar-se ao início do bloco.

Exemplo

BUFFER RES 64 ;Reserva 64 palavras

Veja também

ORG, FILL

SET - DEFINE UMA VARIÁVEL

Sintaxe

<nome> SET <expr>

Descrição

Define uma variável para ser utilizada nas expressões que serão interpretadas pelo compilador. Como o próprio nome diz, o valor determinado por <expr> será associado ao <nome>, podendo ser alterado no decorrer do programa.

Exemplo

```
COMP SET .10
LARG SET .20
ALTURA SET .15
AREA SET COMP * LARG
VOL SET AREA * ALTURA
```

Veja também

EQU, VARIABLE

SPACE - INSERE LINHAS EM BRANCO

Sintaxe

SPACE <expr>

Descrição

Insere <expr> número de linhas em branco na geração do arquivo de listagem do programa (.LST).

Exemplo

SPACE 3 ;insere 3 linhas em branco

Veja também

LIST, TITLE, SUBTITLE, SPACE, PAGE

SUBTITLE - DETERMINA O SUBTÍTULO DO PROGRAMA

Sintaxe

SUBTITLE "<texto>"

Descrição

O <texto> pode ter até 60 caracteres e será utilizado como segunda linha no cabeçalho de toda página no arquivo de listagem do programa (.LST).

Exemplo

SUBTITLE " Desenvolvido pela MOSAICO ENGENHARIA"

Veja também

LIST, TITLE

TITLE - DETERMINA O TÍTULO DO PROGRAMA

Sintaxe

TITLE "<texto>"

Descrição

O <texto> pode ter até 60 caracteres e será utilizado no cabeçalho de toda página no arquivo de listagem do programa (.LST).

Exemplo

TITLE "Programa de exemplo"

Veja também

LIST, SUBTITLE

UDATA - DECLARA O INÍCIO DE UMA SEÇÃO DE DADOS NÃO INICIALIZADOS

Sintaxe

[<nome>] UDATA [<RAM endereço>]

Descrição

Usado com objetos. Declara o início de uma seção de dados não inicializados. Se o argumento <nome> não for especificado, a seção será chamada de UDATA. O endereço inicial para os dados é definido por <RAM endereço>, ou será adotado o valor zero caso esse argumento não seja especificado. Nenhum código será gerado nesta seção. A diretriz RES deve ser usada para reservar espaço para os dados.

Exemplo

```
UDATA
VAR1      RES    .1
DOUBLE    RES    .2
```

Veja também

TEXT, IDATA, UDATA_OVR, UDATA_SHR, EXTERN, GLOBAL

UDATA_OVR - DECLARA O INÍCIO DE UMA SEÇÃO DE DADOS SOBRECARREGADOS

Sintaxe

[<nome>] UDATA_OVR [<RAM endereço>]

Descrição

Usado com objetos. Declara o início de uma seção de dados não inicializados. Se o argumento <nome> não for especificado, a seção será chamada de UDATA_OVR. O endereço inicial para os dados é definido por <RAM endereço>, ou será adotado o valor zero caso este argumento não seja especificado. O espaço declarado nesta seção pode ser sobrecrecido por outras seções desse tipo que possuam o mesmo nome. Isso é ideal para declarar variáveis temporárias que devem ocupar o mesmo espaço na memória. Nenhum código será gerado nesta seção. A diretriz RES deve ser usada para reservar espaço para os dados.

Exemplo

```
TEMPS UDATA_OVR
TEMP1 RES .1
TEMP2 RES .1
TEMP2 RES .1

TEMPS UDATA_OVR
LONGTEMP1 RES .2
LONGTEMP2 RES .2
```

Veja também

TEXT, IDATA, UDATA, EXTERN, GLOBAL, UDATA_SHR

UDATA_SHR - DECLARA O INÍCIO DE UMA SEÇÃO DE DADOS COMPARTILHADOS

Sintaxe

[<nome>] UDATA_SHR [<RAM endereço>]

Descrição

Usado com objetos. Declara o início de uma seção de dados compartilhados e não inicializados. Se o argumento <nome> não for especificado, a seção será chamada de UDATA_OVR. O endereço inicial para os dados é definido por <RAM endereço>, ou será adotado o valor zero caso este argumento não seja especificado. Os dados declarados nesta seção podem ser acessados de qualquer banco da RAM. Isso é ideal para declarar variáveis que são utilizadas em rotinas que trabalham com mais de um banco. Nenhum código será gerado nesta seção. A diretriz RES deve ser usada para reservar espaço para os dados.

Exemplo

```
TEMPS UDATA_OVR  
TEMP1 RES .1  
TEMP2 RES .1  
TEMP2 RES .1
```

Veja também

TEXT, IDATA, UDATA, EXTERN, GLOBAL, UDATA_OVR

#UNDEFINE - ELIMINA UMA SUBSTITUIÇÃO DE TEXTO

Sintaxe

#UNDEFINE <nome>

Descrição

Elimina definição anteriormente criada pela diretriz #DEFINE.

Exemplo

```
#DEFINE LED PORTA, 0
```

```
#UNDEFINE LED
```

Veja também

IFDEF, IFNDEF, #DEFINE

VARIABLE - DEFINE UMA VARIÁVEL

Sintaxe

VARIABLE <nome> [= <expr> ,<nome> = <expr>]

Descrição

Define uma variável para ser utilizada nas expressões que serão interpretadas pelo compilador. Esta diretriz é similar à diretriz SET, com a diferença de que a variável não precisa ser inicializada no momento da sua definição. Como o próprio nome diz, o valor determinado por <expr> será associado ao <nome>, podendo ser alterado no decorrer do programa.

Exemplo

```
VARIABLE      I, CONTA = 5  
. .  
I = 0  
      WHILE I < CONTA  
. .  
I += 1  
ENDW
```

Veja também

SET, CONSTANT

WHILE - LOOP ENQUANTO A EXPRESSÃO FOR VERDADEIRA

Sintaxe

```
WHILE <expr>  
. .
```

```
ENDW
```

Descrição

Enquanto o teste da <expr> resultar em verdadeiro, o bloco definido entre essa diretriz e a ENDW será executado. Caso contrário, ele será pulado. Lembre-se de que o valor 0 (zero) será considerado falso, enquanto qualquer outro número será considerado verdadeiro. O bloco poderá ter no máximo cem linhas e poderá ser repetido até 256 vezes.

Exemplo

```
VARIABLE      I  
CONSTANT     CONTA = 5  
. .  
I = 0  
      WHILE I < CONTA  
          (bloco que será repetido 5 vezes)  
I += 1  
ENDW
```

Veja também

ENDW

INSTRUÇÕES ESPECIAIS

A tabela seguinte mostra várias instruções especiais aceitas pelo compilador para facilitar a digitação do programa. Estas instruções não fazem parte do set de instruções, pois na verdade o compilador irá substituí-las pelas instruções equivalentes. Entretanto, muitos programadores utilizam-se delas na hora de escrever seus programas, e por isso é bom conhecê-las para possibilitar o entendimento e manutenção de sistemas escritos por terceiros. A tabela mostra ainda os bits de STATUS afetados pela operação.

Instrução		Descrição	Instruções Equivalentes	Status
ADDCF	f, d	Se houve carry, incrementa o registrador f, guardando o resultado em d.	BTFSC STATUS, C INCF f, d	Z
ADDDCF	f, d	Se houve digit carry, incrementa o registrador f, guardando o resultado em d.	BTFSC STATUS, DC INCF f, d	Z
B	k	Pula (branch) para a posição definida por k.	GOTO k	
BC	k	Pula (branch) para a posição definida por k, caso tenha ocorrido um carry.	BTFSC STATUS, C GOTO k	
BDC	k	Pula (branch) para a posição definida por k, caso tenha ocorrido um digit carry.	BTFSC STATUS, DC GOTO k	
BNC	k	Pula (branch) para a posição definida por k, caso não tenha ocorrido um carry.	BTFSS STATUS, C GOTO k	
BNDC	k	Pula (branch) para a posição definida por k, caso não tenha ocorrido um digit carry.	BTFSS STATUS, DC GOTO k	
BNZ	k	Pula (branch) para a posição definida por k, caso não tenha ocorrido um zero.	BTFSS STATUS, Z GOTO k	

Instrução	Descrição	Instruções Equivalentes	Status
BZ k	Pula (branch) para a posição definida por k, caso tenha ocorrido um zero.	BTFS C STATUS, Z GOTO k	
CLRC	Limpa o bit de carry.	BCF STATUS, C	C
CLRDC	Limpa o bit de digit carry.	BCF STATUS, DC	DC
CLRZ	Limpa o bit de zero.	BCF STATUS, Z	Z
LCALL k	Chamada de rotina (k) distante. Acerta o PCLATH (bits 3 e 4) automaticamente.	BCF/BSF PCLATH, 3 BCF/BSF PCLATH, 4 CALL k	
LGOTO k	Desvio de programa (k) distante. Acerta o PCLATH (bits 3 e 4) automaticamente.	BCF/BSF PCLATH, 3 BCF/BSF PCLATH, 4 GOTO k	
MOVFW f	Move o valor do registrador f para W.	MOV F, W	Z
NEGF f, d	Acerta o valor do resultado negativo de uma conta.	COMF f, F INCF f, d	Z
SETC	Seta o bit de carry.	BSF STATUS, C	C
SETDC	Seta o bit de digit carry.	BSF STATUS, DC	DC
SETZ	Seta o bit de zero.	BSF STATUS, Z	Z
SKPC	Pula a próxima linha se houve um carry.	BTFS S STATUS, C	
SKPDC	Pula a próxima linha se houve um digit carry.	BTFS S STATUS, DC	
SKPNC	Pula a próxima linha se não houve um carry.	BTFS C STATUS, C	
SKPNDC	Pula a próxima linha se não houve um digit carry.	BTFS C STATUS, DC	
SKPNZ	Pula a próxima linha se não houve um zero.	BTFS C STATUS, Z	
SKPZ	Pula a próxima linha se houve um zero.	BTFS S STATUS, Z	
SUBCF f, d	Se houve carry, decrementa o registrador f, guardando o resultado em d.	BTFS C STATUS, C DEC F f, d	Z
SUBDCF f, d	Se houve digit carry, decrementa o registrador f, guardando o resultado em d.	BTFS C STATUS, DC DEC F f, d	Z
TSTF f	Testa o registrador f para saber se é zero.	MOV F, F	Z

APÊNDICE

E

OPERADORES DO MPASM

Operador	Descrição	Exemplo
\$	Número da atual linha de programa (PC).	GOTO \$ + 3
(Abertura de parênteses.	1 + (R * 4)
)	Fechamento de parênteses.	(COMPR + 1) * 256
!	NÃO lógico.	IF ! (A - B)
-	Negativo.	-1 * COMPR
~	Complemento.	FLAGS = ~FLAGS
high	Byte alto de um símbolo de dois bytes.	MOVWL high CTR_TABELA
low	Byte baixo de um símbolo de dois bytes.	MOVWL low CTR_TABELA
upper	Byte superior de um símbolo de três bytes.	MOVWL upper CTR_TABELA
*	Multiplicação.	A = B * C
/	Divisão.	A = B / C
%	Resto da divisão.	COMPR = TOTAL % 16
+	Soma.	TOTAL = VALOR * 8 + 1
-	Subtração.	TOTAL = VALOR - 10
<<	Rotação para a esquerda.	VALOR = FLAGS << 1
>>	Rotação para a direita.	VALOR = FLAGS >> 2
>=	Maior ou igual.	IF INDICE >= NUM
>	Maior que.	IF INDICE > NUM
<	Menor que.	IF INDICE < NUM
<=	Menor ou igual.	IF INDICE <= NUM
==	Igual a.	IF INDICE == NUM
!=	Diferente de (NÃO igual).	IF INDICE != NUM
&	Operação E (AND) bit a bit.	FLAGS = FLAGS & MASCARA
^	Operação OU exclusivo (XOR) bit a bit.	FLAGS = FLAGS ^ MASCARA
	Operação OU inclusivo (IOR) bit a bit.	FLAGS = FLAGS MASCARA
&&	E lógico (AND).	IF (COMPR == 10) && (A > B)
	OU lógico (OR).	IF (COMPR == 10) (A != B)
=	Variável = valor.	INDICE = 0

Operador	Descrição	Exemplo
<code>+ =</code>	Variável = ela mesma + valor.	INDICE <code>+ = 1</code>
<code>- =</code>	Variável = ela mesma - valor.	INDICE <code>- = 1</code>
<code>* =</code>	Variável = ela mesma * valor.	INDICE <code>* = TOTAL</code>
<code>/ =</code>	Variável = ela mesma / valor.	INDICE <code>/ = TOTAL</code>
<code>% =</code>	Variável = resto da divisão dela mesma pelo valor.	INDICE <code>% = 16</code>
<code><<=</code>	Variável = ela mesma rotacionada à esquerda.	INDICE <code><<= 3</code>
<code>>>=</code>	Variável = ela mesma rotacionada à direita.	INDICE <code>>>= 2</code>
<code>& =</code>	Variável = ela mesma AND valor	INDICE <code>& = MASCARA</code>
<code> =</code>	Variável = ela mesma IOR valor	INDICE <code> = MASCARA</code>
<code>^ =</code>	Variável = ela mesma XOR valor	INDICE <code>^ = MASCARA</code>
<code>++</code>	Variável = ela mesma + 1 (incremento)	INDICE <code>++</code>
<code>--</code>	Variável = ela mesma - 1 (decremento)	INDICE <code>--</code>

APÊNDICE

F

TABELAS

		Caracteres ASCII							
		Digito mais significativo							
Valores em HEX		0	1	2	3	4	5	6	7
Dígito menos significativo	0			Espaço	0	@	P	`	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	\$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	BELL	ETB	'	7	G	W	g	w
	8	BS	CAN	(8	H	X	h	x
	9	HT	EM)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K		k	{
	C	FF	FS	,	<	L	\	l	
	D	CR	GS	-	=	M]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

Conversão Hexadecimal → Decimal							
Byte 2				Byte 1			
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15

Para usar esta tabela, some o valor decimal correspondente a cada dígito do número hexadecimal. Por exemplo, o número HEX A38F equivale a 41871 em decimal.

HEX (dígito 4)	HEX (dígito 3)	HEX (dígito 2)	HEX (dígito 1)	Resultado
A	3	8	F	A38F
40960	+ 768	+ 128	+ 15	= 41871

G

HARDWARE PROPOSTO

Sugerimos aqui um hardware para a realização dos treinamentos e projetos presentes neste livro. Com o esquema elétrico anexo, é possível a montagem de um circuito capaz de realizar um grande número de experiências com o PIC 16F628A (ou outro de 18 pinos). Para facilitar, seguem algumas dicas sobre o esquema em questão:

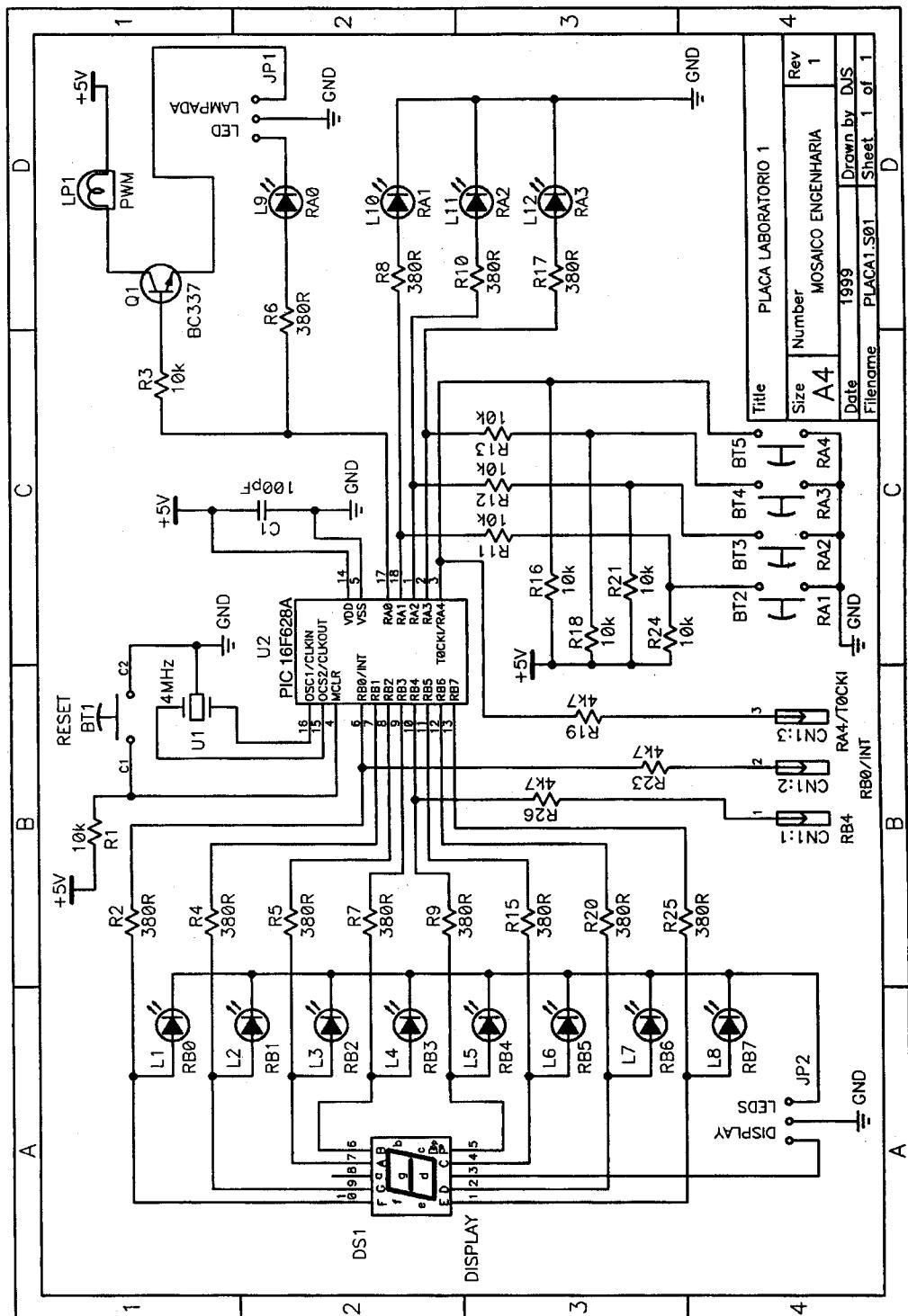
- Ao PORTB estão ligados 8 LEDs (L1 a L8) identificados com os mesmos nomes dos pinos. Eles são muito úteis para experimentos com bytes por meio da representação binária.
- Ao PORTB está ligado também um display de sete segmentos, cuja tabela de relacionamento é a seguinte:

Pino	Segmento
RB0	G
RB1	F
RB2	A
RB3	B
RB4	Ponto
RB5	C
RB6	D
RB7	E

- Esta relação foi escolhida para facilitar o layout da placa, mas pode ser alterada sem muitos problemas, desde que os exemplos sejam acertados.
- O jumper JP2 serve para selecionar com qual saída (LEDs ou display) o PORTB irá operar. Observe que a seleção é exclusiva, para evitarmos sobrecargas nos pinos do PIC.
- Para diminuir os custos, utilizamos um ressoador cerâmico de 4 MHz, que é facilmente encontrado no mercado.
- Introduzimos um botão de RESET (BT1), diretamente ligado ao /MCRL, para possibilitar resets externos.

- O pino RA0 foi deixado como uma saída de controle, que pode acionar o LED L9 ou a lâmpada LP1. A lâmpada é utilizada para testes de PWM, como no dimmer.
- O jumper JP1 seleciona entre o controle da lâmpada ou do LED.
- Três LEDs (L10 a L12) foram colocados nos pinos RA1, RA2 e RA3 para sinalizações diversas. Não colocamos LED para o pino RA4 por ele ser "open colector".
- Quatro botões (BT2 a BT5) foram ligados aos pinos RA1, RA2, RA3 e RA4 para controles diversos.
- Observe que os pinos RA1, RA2 e RA3 possuem ligações para botões e LEDs. Com o hardware proposto, é possível operarmos com o LED quando o pino é configurado para saída, e com o botão quando o pino está configurado como entrada.
- Um conector adicional (CN1) foi deixado para experiências que utilizem interrupções (RB0 e RB4) e contagem de pulsos externos (T0CKI). A esses pinos (RB0, RB4 e RA4) não está ligado nenhum pull-up ou pull-down. Por isso, quando for utilizá-los como entrada, os dois níveis devem ser garantidos.
- A placa deve ser alimentada com 5Vdc. Sugerimos a utilização de lâmpada de lanterna (6V). No caso de utilização de lâmpada automotiva, pode-se alterar o esquema para alimentá-la com 12Vdc.

Para facilitar ainda mais a sua vida, a Mosaico disponibiliza esta placa já montada em seu site, sob a denominação de McLab1. Trata-se de uma placa simples, robusta e barata, com um recurso muito interessante: a McLab1 já possui o soquete de gravação "in-circuit", tornando-a totalmente compatível com os gravadores McFlash e McPlus. Desta forma, todo o seu aprendizado e treinamento será ainda mais fácil e rápido.



REFERÊNCIAS BIBLIOGRÁFICAS

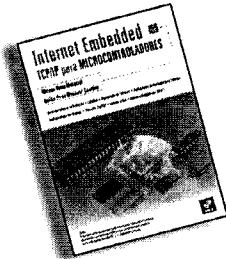
MICROCHIP Technology Inc.

- _____ . Data Sheet PIC16F628A, 2003.
- _____ . MPASM Assembler User's Guide, 2002.
- _____ . MPLAB User's Guide V6.22, 2003.
- _____ . Home page (Internet) e Technical Library CD-ROM, 2003.
- _____ . Product Line Card, 2003.

MARCAS REGISTRADAS

Os nomes PIC, PICmicro, Microchip, In-Circuit Serial Programming, Mplab e MPASM são propriedades da Microchip Technology Inc. nos Estados Unidos e em outros países.

Todos os demais nomes registrados, marcas registradas, ou direitos de uso citados neste livro, pertencem aos respectivos proprietários.

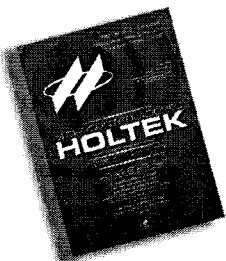


Internet Embedded - TCP/IP para Microcontroladores

Autores: Marcos P. Mokarzel e Karina P. Mokarzel Camelo • Código: 0425 • 344 páginas • Formato: 17 x 24 cm

O objetivo deste livro é estudar os conceitos para a conexão de equipamentos microcontrolados na rede Internet, para este estudo ele traz um software completo em linguagem C que serve de base para futuros projetos. Ele inicia com o estudo de baixo nível dos conceitos do protocolo IP com foco no TCP/IP, estrutura e organização do software, hardware mínimo para conexão Ethernet e desenvolve passo a passo os blocos em C que compõem uma API (Application Programming Interface) que pode ser acessada por outros programas para a comunicação TCP/IP.

No final é feita uma aplicação de um servidor HTTP dinâmico, que mostra a temperatura local e o atraso na rede, e informa por meio de uma página que pode ser aberta em qualquer navegador web.

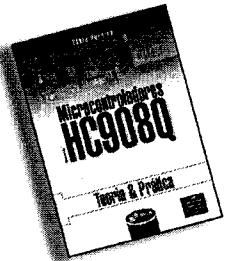


Microcontroladores Holtek - Teoria e Prática - Baseado nas Famílias I/O (HT48) e A/D (HT46)

Autores: Denys E. C. Nicolosi, Silvio Augusto Bortolim e Marcos Tadeu Scaff • Código: 0204 • 248 páginas • Formato: 17 x 24 cm

Destinado a estudantes, técnicos, engenheiros ou entusiastas da área de eletrônica ou mecatrônica, o livro apresenta em todo o seu conteúdo uma linguagem clara e acessível.

Aborda as características do microcontrolador da família Holtek, seu princípio de funcionamento, o conjunto de instruções com vários exemplos e detalhes, o funcionamento do periférico PWM, conversor A/D e barramento serial I²C. Apresenta o Ambiente de Desenvolvimento Integrado Holtek – HT-IDE; como criar o projeto passo a passo, o processo de depuração (que pode ser feito pela emulação ou simulação) e o uso do VPM, um software para simulação dos componentes externos conectados ao microcontrolador; experiências introdutórias como o uso de endereçamento indireto, interrupção, modo HALT e Watchdog Timer e experiências mais aprofundadas e úteis como varredura em matriz de teclado, LCD 16x2, comunicação serial entre o microcontrolador Holtek e o PC; conversores A/D e D/A, uso da E²PROM externa através do barramento serial I²C, etc.

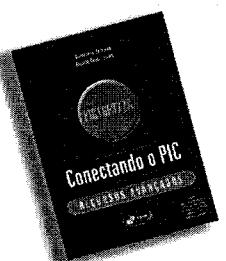


Microcontroladores HC908Q - Teoria e Prática

Autor: Fábio Pereira • Código: 0158 • 296 páginas • Formato: 17 x 24 cm

Aborda os microcontroladores Motorola HC908Q nas suas versões de 8 e 16 pinos. São estudados todos os detalhes dos chips, conjunto de instruções, arquitetura e periféricos internos, além de apresentar o ambiente de programação Codewarrior, o montador Assembly e o compilador C que compõem o ambiente de desenvolvimento.

Destaca a linguagem C e o compilador HC08 com uma revisão de ANSI C, detalhes e características especiais do compilador e técnicas de otimização do programa, além de diversos exemplos que utilizam a placa de demonstração M68EV908QC: pisca-pisca com LED, controle PWM de brilho com leitura analógica, temporizador com display LED multiplexado e um voltímetro digital com display LCD.



Conectando o PIC - Recursos Avançados

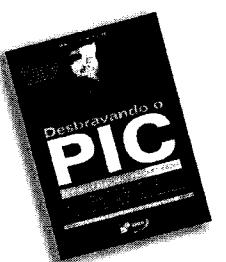
Autores: David José de Souza e Nicolás César Lavinia • Código: 7376 • 384 páginas • Formato: 17 x 24 cm

É um livro destinado aos profissionais da área de eletrônica, incluindo hobistas, técnicos e engenheiros.

É necessário que o leitor já tenha algum conhecimento da linguagem Assembly do PIC, assim como das ferramentas de trabalho: o MpLab e um sistema de gravação.

O conteúdo é altamente didático e técnico, acompanhado de teoria, exemplos, fluxogramas, código e exercícios propostos. Os principais assuntos abordados são: introdução ao PIC 16F877A, resumo do set de instruções, as primeiras explorações (I/Os e timers), varredura de display com 4 dígitos, operação com display de cristal líquido (LCD), conversor analógico-digital interno e por RC, os módulos CCP (Capture/Compare/PWM), memórias não-voláteis (E2PROM/FLASH), comunicação serial 1 - SPI e I²C, comunicação serial 2 - USART e implementação de um sistema de medição de temperatura.

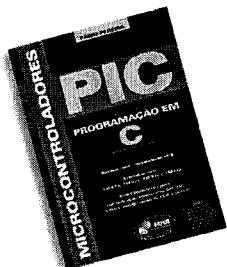
Ao final, possui apêndices com assuntos relevantes para a execução de seus projetos.



Desbravando o PIC - Ampliado e Atualizado para PIC 16F628A

Autor: David José de Souza • Código: 8674 • 272 páginas • Formato: 17 x 24 cm

Este livro é dedicado às pessoas que desejam conhecer e programar o PIC, com base no PIC16F628A. Ele aborda desde os conceitos teóricos do componente, passando pela ferramenta de trabalho (MPLab) e aprofundando-se na linguagem de programação Assembler (MPASM). O MPLab 6.22 também é estudado, com um capítulo dedicado à simulação e debugação. Quanto ao PIC, todos os seus recursos são tratados, incluindo programação, interrupções, os timers (TMR0, TMR1, TMR2 e WDT), a EEPROM interna, comparadores, o modo de tensão de referência, o modo CCP (PWM), comunicação via USART e muito mais, acompanhados de exemplos completos e projetos propostos.

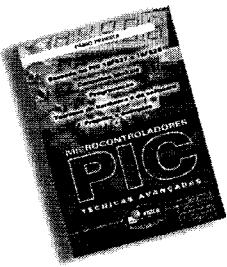


Microcontroladores PIC - Programação em C

Autor: Fábio Pereira • Código: 9352 • 360 páginas • Formato: 17 x 24 cm

A proposta desse livro é abordar a linguagem C em profundidade, mas sempre tendo como foco os microcontroladores PIC e os compiladores CCS. Num total de 12 capítulos, são apresentados os princípios básicos de programação, a linguagem C, diferenças entre C ANSI e C CCS, diretrizes e funções do compilador, técnicas de programação C para PICs, técnicas de otimização, tratamento de interrupções, manipulação de E/S, manipulação de timers internos, teclados, displays (incluindo módulos LCD), comunicação serial, conversão A/D (interna e delta-sigma), PWM, etc. Os exemplos são baseados nos principais PICs disponíveis atualmente: 12F675, 16F62x, 16F87x e 18Fxx2.

Possui diversos exercícios de fixação e exemplos, tais como: terminal RS-232 com LCD, conversores A/D com comunicação serial, comunicação com memórias I2C, controle de brilho de led com PWM, medição de temperatura com DS18S20, teclado de 12 teclas com auto-repetição, animação em LCD com caracteres



Microcontroladores PIC - Técnicas Avançadas

Autor: Fábio Pereira • Código: 7279 • 360 páginas • Formato: 17 x 24 cm

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 (baseado nas versões 16F627 e 16F628). 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 material indispensável na biblioteca de todos que pretendem aprofundar seus conhecimentos sobre os microcontroladores PIC.



Laboratório de Microcontroladores - Família 8051 - Treino de Instruções, Hardware e Software

Autor: Denys E. C. Nicolosi • Código: 8712 • 208 páginas • Formato: 17 x 24 cm

A proposta deste livro é enfocar o lado prático da aplicação do microcontrolador com detalhes, enfatizando o auto-aprendizado e autonomia. Para isto desenvolveu-se um material que apresenta: 22 programas prontos de laboratório de treino de instruções para autotreinamento por meio de um "PC"; 10 experiências de laboratório de hardware e software, também estruturadas para autotreinamento, sempre com um programa base que vem pronto para o estudante "rodar" juntamente com seu esquema eletrônico, tais como: Excitação de LEDs, Gerenciamento de Display de 7 Segmentos, Gerenciamento de Teclado simples, Alarme com Buzzer, Motor de Passo, saída PWM, etc.

Possui também listas completas das instruções, exercícios propostos, diagramas de programação, extensa bibliografia de livros e sites da área.



Aplicações Práticas do Microcontrolador 8051

Autor: Vidal Pereira da Silva Jr. • Código: 9395 • 248 páginas • Formato: 17 x 24 cm

O 8051, microcomputador de um só chip, é estudado de forma completa, desde uma introdução acessível aos iniciantes da área até exemplos completos de hardware e software com teclados, LCD, saída para impressora, conversão A/D e D/A e outros, com ênfase aos programas escritos em Assembler e em "C".

Microcontrolador 8051 - Detalhado

Autor: Denys Emilio Campion Nicolosi • Código: 721x • 256 páginas • Formato: 17 x 24 cm

Desbravando o PIC

AMPLIADO E ATUALIZADO PARA PIC16F628A

"Desbravando o PIC" é um livro dedicado às pessoas que desejam conhecer e programar o PIC, com base no PIC16F628A. Ele aborda desde os conceitos teóricos do componente, passando pela ferramenta de trabalho (MPLab) e aprofundando-se na linguagem de programação assembler (MPASM). Desta forma o MPLab 6.22 é estudado, com um capítulo dedicado à Simulação e Debugação. Quanto ao PIC, todos os seus recursos são tratados, incluindo a programação, as interrupções, os timers (TMR0, TMR1, TMR2 e WDT), a EEPROM interna, os comparadores, o modo de tensão de referência, o modo CCP (PWM), a comunicação via USART e muito mais. Outro ponto forte da obra é a estruturação do texto que foi elaborada para utilização em treinamentos ou por autodidatas, com exemplos completos e projetos propostos. O livro traz ainda vários apêndices técnicos, como o resumo do set de instrução, as diretrizes de programação e uma proposta de circuito para que você monte a sua própria placa de laboratório, tornando-o uma referência que não pode faltar em sua biblioteca.

Editorialmente mencionados neste livro são ilustrativos, podendo ser modificados ou extintos a qualquer momento.

Invista em você.

Visite uma livraria.

www.editoraerica.com.br

ISBN: 85-7194-867-4



9 788571 948679