

3^a EDIÇÃO

David José de Souza
Nicolás César Lavinia

PIC16F877A

Conectando o PIC

RECURSOS AVANÇADOS



BRINDE

Edição-Série dos exemplos,
softwares de comunicação e
kits de prova dos componentes
disponíveis na TEC
disponível no site.

Conectando o
PIC16F877A
Recursos Avançados



EDITORAS AFILIADA

Dedicatória

Mais uma vez tenho a chance de dedicar meu trabalho à pessoa que mais amo neste mundo: minha querida esposa Eliete;

Dedico também à minha amada sobrinha, Bia, que ainda é muito pequena para saber o significado de uma dedicatória, mas que me alegra com força suficiente para uma vida inteira.

David José de Souza

A paternidade é realmente uma experiência incrível. Há algum tempo eu não tinha uma idéia muito precisa sobre meu futuro, e hoje me vejo um homem muito mais completo, pois já plantei minha árvore, escrevi meu livro e acabei de ser pai. Por isso, dedico meu trabalho à minha querida esposa Andréa e à minha recém-nascida filha Nicole.

Nicolás César Lavinia

Em paz me deito e logo adormeço, porque só tu,
Senhor, me fazes viver em segurança

Agradecimentos

Uma experiência nunca é igual à outra. Escrever este livro foi muito mais difícil que o primeiro, pois a quantidade de conhecimento técnico necessário estava bem acima do meu potencial inicial. Uma vez mais o desafio prova que as pessoas são capazes de vencer seus próprios limites, principalmente quando temos o apoio das pessoas que se encontram ao nosso lado. Este é o motivo mais forte de todos para justificar esses agradecimentos.

Mais uma vez tenho a obrigação, sem o pesar da palavra, de agradecer à minha família e em especial à minha esposa. Amo todos vocês.

Agradeço a todo o pessoal da Mosaico pela participação de cada um neste livro, cada qual do seu jeito e com suas condições, mas devo um agradecimento muito especial ao meu sócio, amigo e co-autor, Nicolás, pois sem seu conhecimento técnico e capacidade de trabalho jamais conseguiria terminar esta obra. Um abraço muito especial também aos meus outros sócios da Mosaico: meu irmão de sangue, José Carlos; meu irmão de vida, Gil; e meu irmão de alma, Vanderlei.

Na verdade, concluí esta obra em junho de 2002 e, por isso, gostaria muito de aproveitar, sem demagogias, para agradecer ao meu País, cuja seleção tornou-se agora Pentacampeã Mundial de Futebol. Sou patriota e apaixonado pelo Brasil, mas gostaria de que nosso povo sentisse orgulho da sua pátria diariamente. Devemos levantar nossas cabeças e trabalhar pelo nosso futuro, recuperar nossa auto-estima e ter a certeza de que a única solução do nosso País está no próprio povo.

Por último, como em meus demais textos, não poderia deixar de agradecer a você, leitor, que está se preparando para passar horas ao lado deste livro, enfrentando os mesmos desafios que nós enfrentamos quando começamos a explorar esse vasto universo denominado PIC.

David José de Souza

Distribuidores



Mosaico Engenharia Eletrônica S/C Ltda.

Projetos eletrônicos com especialização em PIC,
consultoria e treinamento.

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

Tel:(11)4992-8775

Tel/Fax:(11)4992-8775

E-mail: comercial@labtools.com.br

Home page: www.mosaicohps.com.br

Microchip Technology Inc.

Fabricante do PIC.

2355, W. Chandler Blvd. - Chandler - Arizona -
USA

Tel: (480) 768-7200 / Fax: (480) 899-9210

Home page: www.microchip.com

Aplicações Eletrônicas Artimar Ltda.

Representante exclusivo Microchip no Brasil.

Rua Marquês de Itú, 70 - 8- andar - Conj. 82 -
São Paulo - SP

Tel: (11) 3231-0277

Fax: (11) 3255-0511

E-mail: artimar@artimar.com.br

Home page: www.artimar.com.br

Aut-Comp

Revendedor autorizado.

Rua Lord Cokrane, 616 -139 andar - Sala 1304 -
São Paulo - SP

Tel: (11) 6915-7443

Fax: (11) 6915-7443

Home page: www.autcomp.com.br

Farnell do Brasil

Revendedor autorizado.

Rua Emir Macedo Nogueira, 240 - Diadema - SP

Tel: (11) 4066-9400

Fax: (11) 4066-9410

Home page: www.farnell.com

Future Electronics

Revendedor autorizado.

Rua Lusitana, 740 - 102 andar - Conj. 103/104 -
Campinas - SP

Tel: (19) 3737-4100

E-mail: future@zaz.com.br

Home page: www.future-active.com.br

Hitech

Revendedor autorizado.
Rua Branco de Moraes, 489 - São Paulo - SP
Tel: (11) 5188-4000
Fax: (11) 5188-4191
E-mail: microchip@hitech.com.br
Home page: www.hitech.com.br

Tec Tecnologia

Revendedor autorizado.
Rua Flórida, 1737 - 29 andar - São Paulo - SP
Tel: (11) 5505-2046
Fax: (11) 5505-0017
Home page: www.tec-arrow.com.br

Sobre o Material Disponível na Internet

O material disponível no site da Editora Érica contém: códigos-fonte dos exemplos apresentados, data sheets dos componentes adicionais (LCD e memória) e o software M2Com for Windows para testes de transmissão e recepção da comunicação serial via RS232, desenvolvido pelos autores. É necessário ter instalado em sua máquina AdobeAcrobat 4,0 e MPLAB 5.7 ou superior.

Pic.exe-1.87MB

Procedimentos 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 de pesquisa** - Localize o livro desejado, digitando palavras-chave (título do livro ou nome 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-chave (título do livro ou nome do autor). Serão exibidos o nome do livro e o arquivo para download. Dê um clique sobre o arquivo executável que será transferido.

Procedimentos para Descompactação

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

C:\Conectando 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

Novamente os autores nos presenteiam com uma obra de excelente qualidade técnica. Voltada especialmente para a área de projetos com sistemas microcontrolados, com ênfase no microcontrolador PIC16F877A, ela disponibiliza ao público uma série de técnicas de "software" e "hardware" que foram desenvolvidas por eles e são utilizadas diariamente em seus laboratórios. Esta é a melhor forma de obtermos acesso a métodos realmente práticos e atualizados com as necessidades do mercado.

Totalmente transparente, esta literatura trata de forma clara e direta uma série de aspectos e problemas práticos em projetos de sistemas microcontrolados, fornecendo soluções abertas e de primorosa qualidade. Mais do que fornecer soluções prontas, o leitor vai verificar que no "Conectando o PIC 16F877A - Recursos Avançados" os assuntos são detalhadamente comentados e explicados, possibilitando que ele desenvolva projetos de maior complexidade.

Com seriedade e responsabilidade, os autores vem mostrando para o público brasileiro que o sucesso de um empresa na área de projetos não reside em deter todo conhecimento para si, mas saber difundi-lo de maneira a possibilitar a um grupo cada vez maior de pessoas o acesso a maravilhosas soluções que a eletrônica microcontrolada permite. E em poucas palavras: a competência não teme a concorrência.

Este livro combina tão bem o embasamento teórico com a exemplificação prática, que se torna um excelente material de apoio didático tanto para as pessoas que desejam aprender por seus próprios meios quanto para nós, professores, que necessitamos tanto de material complementar.

Que nesta era do conhecimento possamos, cada vez mais, contar com novas obras deste porte e qualidade.

José Carlos de Souza Jr.

Professor Universitário de diversas matérias da cadeira de Engenharia Eletrônica, incluindo a de Microcontroladores, em faculdades e universidades de São Paulo, tais como FEI, Mauá e São Judas.

Sobre os Autores

David José de Souza é formado em Engenharia Mecânica pela Universidade Santa Cecília, trabalhando há quatro anos no setor. 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.

Nicolás César Lavínia é engenheiro eletricista com especialização em eletrônica, formado pela Escola de Engenharia Mauá. Cursou mestrado no Laboratório de Automação e Controle da Escola Politécnica da Universidade de São Paulo. Possui treinamento "Master em microcontroladores PIC" promovido pela própria Microchip em Arizona, nos EUA. Trabalha na área de desenvolvimento de projetos e consultoria há mais de sete anos. Atualmente, é sócio e diretor técnico da Mosaico Engenharia, responsável pela coordenação do laboratório de desenvolvimento. Como co-autor deste livro, expõe a vasta experiência adquirida no desenvolvimento de projetos eletrônicos.

Sumário

Capítulo 1 – Introdução.....	14
Nossos objetivos.....	14
Pré-requisitos sugeridos.....	14
A didática do sistema.....	14
Introdução.....	14
Prática.....	15
Projeto final.....	15
Apêndices.....	16
Padrões textuais.....	16
Capítulo 2 - O PIC 16F877A.....	18
Pinagem.....	19
Nomenclatura dos pinos.....	19
A estruturação interna.....	22
Os ciclos de máquina.....	23
A memória de programa.....	24
Vetor de reset.....	24
Vetor de interrupção.....	24
Tamanho da memória e páginas.....	24
Pilha (Stack).....	26
Mapa da memória de programa.....	26
A memória de dados volátil (RAM).....	26
Registradores especiais.....	27
Registradores de uso geral.....	27
Tamanho da memória e bancos.....	27
Mapa da memória de dados.....	29
Memórias não-voláteis (E PROM e FLASH).....	29
As interrupções.....	30
Interrupção de Timer 0.....	30
Interrupção externa.....	30
Interrupção por mudança de estado.....	31
Interrupção da porta paralela (PSP).....	31
Interrupção dos conversores A/D.....	31
Interrupção de recepção da USART.....	31
Interrupção de transmissão da USART.....	31
Interrupção da comunicação serial (SPI e I ² C).....	31
Interrupção do CCP1 (Capture/Compare/PWM).....	31
Interrupção do Timer 2.....	32
Interrupção do Time 1.....	32
Interrupção de fim de escrita na E PROM/FLASH.....	32
Interrupção de colisão de dados (Bus Collision).....	32
Interrupção do CCP2 (Capture/Compare/PWM).....	32
Interrupção dos comparadores.....	32
Operando com as interrupções.....	32
Demais recursos e periféricos.....	35
Características elétricas.....	36
Capítulo 3 - Resumo do Set de Instruções.....	38
Os termos utilizados.....	38
A construção dos nomes das instruções.....	39
O resumo das instruções.....	39
Capítulo 4 - As Primeiras Explorações (I/Os e Timers).....	42
Introdução.....	42
Teoria e recursos do PIC.....	42
Estudo das Portas.....	42
Estudo dos Timers.....	47

Lógica do exemplo.....	55
Esquema elétrico.....	56
Fluxograma.....	57
Código.....	59
Dicas e comentários.....	65
Exercícios propostos.....	65
Capítulo 5 - Varredura de Display de Quatro Dígitos.....	67
Introdução.....	67
Teoria e recursos do PIC.....	67
Lógica do exemplo.....	69
Esquema elétrico.....	71
Fluxograma.....	72
Código.....	76
Dicas e comentários.....	95
Exercícios propostos.....	95
Capítulo 6-Operação com Display de Cristal Líquido (LCD).....	96
Introdução.....	96
Teoria e recursos do PIC.....	96
Inicialização do LCD.....	98
Comandos do LCD.....	100
Lógica do exemplo.....	104
Esquema elétrico.....	105
Fluxograma.....	106
Código.....	110
Dicas e comentários.....	127
Exercícios propostos.....	127
Capítulo 7 – Conversor Analógico-Digital Interno.....	128
Introdução.....	128
Teoria.....	128
Recursos do PIC.....	130
Lógica do exemplo.....	137
Esquema elétrico.....	137
Fluxograma.....	138
Código.....	140
Dicas e comentários.....	148
Exercícios propostos.....	148
Capítulo 8 - Conversor Analógico-Digital POR RC.....	149
Introdução.....	149
Teoria e recursos do PIC.....	149
Lógica do exemplo.....	151
Esquema elétrico.....	153
Fluxograma.....	154
Código.....	156
Dicas e comentários.....	169
Exercícios propostos.....	169
Capítulo 9 - Os Módulos CCP(Capture/Compare/PWM).....	170
Introdução.....	170
Teoria e recursos do PIC.....	170
Modo Capture.....	171
Modo Compare.....	172
Modo PWM.....	174
Lógica do exemplo.....	179
Esquema Elétrico.....	180

Fluxograma.....	181
Código.....	183
Dicas e comentários.....	199
Exercícios propostos.....	199
Capítulo 10 - Trabalhando com as Memórias Não-Voláteis.....	200
Introdução.....	200
Teoria e Recursos do PIC.....	200
Escrevendo na E ² PROM (Dados).....	202
Lendo a E ² PROM (Dados).....	203
Escrevendo na FLASH (Programa).....	203
Lendo a FLASH (Programa).....	204
Tratando a interrupção de final de escrita na E ² PROM e FLASH.....	204
Lógica do exemplo.....	205
Esquema elétrico.....	206
Fluxograma.....	207
Código.....	212
Dicas e Comentários.....	239
Exercícios propostos.....	239
Capítulo 11-Comunicação Serial 1 - SPI e I²C.....	240
Introdução.....	240
Teoria e recursos do PIC para SPI.....	240
Teoria para I ² C ²	246
Condição de Start.....	247
Condição de Stop.....	247
Condição de Re-Start.....	248
Condição de Acknowledge (ACK).....	248
Transmissão de endereço.....	249
Transmissão de dados.....	249
Pausas.....	249
Diagramas de tempo.....	249
Recursos do PIC para I ² C ²	253
Modo Slave.....	253
Modo Master.....	259
Lógica do exemplo.....	266
Esquema elétrico.....	268
Fluxograma.....	269
Código.....	274
Dicas e comentários.....	297
Exercícios propostos.....	297
Capítulo 12 - Comunicação Serial 2 – USART.....	299
Introdução.....	299
Teoria.....	299
Modo assíncrono.....	299
Modo síncrono.....	301
Recursos do PIC.....	302
Modo assíncrono.....	306
Modo síncrono.....	307
Lógica do exemplo.....	309
Esquema elétrico.....	310
Fluxograma.....	311
Código.....	313
Dicas e comentários.....	327
Exercícios propostos.....	327

Capítulo 13-Outras Características.....	328
Introdução.....	328
Comunicação paralela (PSP).....	328
Watchdog Tirner (WDT).....	330
Power-on Reset (POR).....	331
Power-up Timer(PWRT).....	331
Oscilator Start-upTimer (OST).....	331
Brown-out Reset (BOR).....	332
SLEEP (Power-downMode).....	332
Controle de Resets.....	333
Oscilador.....	333
Sistema de proteção do código (Code Protection).....	335
Registradoras de identificação (IDs).....	335
Sistema de emulação In-Circuit (Debugger Mode).....	335
Proteção de escrita interna da FLASH.....	336
Gravação In-Circuit (ICSP).....	336
Gravação em baixa tensão (Low Voltage Programiring).....	336
Capítulo 14 - Implementando um Sistema de Medição de Temperatura.....	338
Inrodução.....	338
O sistema.....	338
O sensor de temperatura.....	338
O aquecimento.....	339
O resfriamento.....	339
Comunicação serial.....	339
Considerações gerais.....	340
Esquema elétrico.....	340
Fluxograma.....	341
Código.....	346
Apêndice A-Detalhamento dos Registradores Especiais (SFRS).....	379
Introdução.....	379
Agrupamento e localização.....	379
Resumo e condições após reset.....	406
Apêndice B - Set de Instruções Completo (para 14 bits).....	409
Apêndice C - Diretrizes da Linguagem MPASWI.....	430
Apêndice D-Instruções Especiais.....	454
Apêndice E-Operadores do Compilador.....	456
Apêndice F - Esquema Elétrico da Placa Proposta (McLab2).....	458
índice Remissivo.....	462
Referências Bibliográficas.....	466

Introdução

Nossos objetivos

Quando criamos nossa primeira obra sobre este tema, o livro "Desbravando o PIC", dedicamos muitas horas de trabalho para que tudo saísse conforme nosso intento e que o sucesso alcançado por essa literatura seja o reconhecimento da qualidade do nosso trabalho. Entretanto, era nossa intenção, desde o começo, que a literatura disponível para o PIC fosse muito mais abrangente.

O objetivo deste segundo livro é exatamente este: complementar o conhecimento dos usuários do PIC. Enquanto o primeiro destina-se ao aprendizado da linguagem assembly, utilizando-se para isso dos recursos disponíveis no PIC16F628A, é nossa intenção agora aprimorar seus conhecimentos nos demais recursos da família PIC, além da integração com periféricos práticos e totalmente necessários nos dias de hoje, como varredura de displays, LCD e comunicação RS-232,

Com esses conhecimentos, seus projetos poderão dar um salto evolutivo em relação aos recursos empregados, possibilitando uma melhor interface com o usuário e o mundo exterior de uma forma geral. Com os novos recursos do PIC, seus programas ficarão menores e mais eficientes, seus circuitos mais enxutos e robustos, seus sistemas muito mais avançados e poderosos. A partir de agora, seus limites serão sua dedicação e criatividade.

Pré-requisitos sugeridos

Entretanto, tudo isso exige um conhecimento prévio. Por isso, para um bom desempenho neste treinamento, é necessário que você já tenha conhecimento da linguagem assembly do PIC, assim como as ferramentas de trabalho: o MpLab e um sistema de gravação. Como já dissemos, nossa intenção não é o aprendizado da programação básica, mas sim a exploração de recursos avançados e os sistemas complementares. Caso você ainda não domine a linguagem, recomendamos a leitura da obra "Desbravando o PIC", também publicado pela Editora Érica.

A didática do sistema

Mais uma vez houve, de nossa parte, grande empenho em relação à didática e ordem cronológica aplicada ao texto. Consideramos essa sistemática muito importante para o seu desempenho, principalmente numa aplicação autodidata.

Desta vez resolvemos dividir o conhecimento em quatro grandes partes:

Introdução

Esta parte será destinada ao esclarecimento das características do PIC utilizado como base no estudo. Antes de nos aprofundarmos sobre os recursos, devemos conhecer bem todas as facetas do PIC em questão, tais como seus registradores especiais, características elétricas, pinagem, etc.

Prática

Todas as funções e recursos estudados estão divididos em dez capítulos práticos. Cada um desses capítulos possui uma estruturação padronizada, baseada nos seguintes tópicos:

- **Introdução:** Explicações gerais sobre os assuntos estudados;
- **Teoria:** Quando necessário, detalha o embasamento teórico obrigatório para o entendimento de assunto em questão;
- **Recursos do PIC:** Detalhamento dos recursos do PIC que serão utilizados nos exemplos expostos, com a explicação do funcionamento, registradores envolvidos, etc. Muitas vezes esse tópico pode estar agrupado com o anterior;
- **Lógica do exemplo:** Explicará qual a lógica utilizada para a montagem do programa de exemplo;
- **Esquema elétrico:** Apresenta o esquema elétrico necessário para a implementação do programa. Esse esquema é totalmente compatível com o hardware apresentado no Apêndice F (McLab2), porém apresenta somente os elementos relacionados com o capítulo em questão;
- **Fluxograma:** Este tópico apresentará o fluxograma de operação para complementar a lógica apresentada anteriormente e facilitar o entendimento do exemplo;
- **Código:** O código propriamente dito, em assembly, para o exemplo do capítulo;
- **Dicas e comentários:** Dicas, comentários importantes e demais desdobramentos resultantes da utilização do exemplo apresentado;
- **Exercícios propostos:** Outros problemas e questões que podem ser discutidos e resolvidos com os recursos já aprendidos

Um fato importante é que, com essa sistemática, todos os tópicos relacionados a um determinado assunto encontram-se agrupados no mesmo capítulo, desde o embasamento teórico até o exemplo estudado e os exercícios propostos. Desta maneira, cada capítulo torna-se independente dos demais, podendo ser estudado separadamente.

Projeto final

Ao término dos capítulos práticos será apresentado um projeto real envolvendo a medição de temperatura, com atuadores para aquecimento e resfriamento. Trata-se de um exemplo capaz de utilizar os recursos adicionais da placa apresentada no Apêndice F (McLab2), baseando-se nas práticas aprendidas durante o treinamento.

Apêndices

Como já conhecemos as características marcantes de nossos leitores, os apêndices tornaram-se obrigatórios em uma boa literatura didática. Eles lhe serão de grande valia quando você se transformar em um programador dedicado, pois tornam-se fontes de consulta permanente. Desta forma, tentamos colocar nos apêndices informações úteis ao dia-a-dia do programador, seguindo, sempre que possível a formatação e os termos utilizados nas literaturas técnicas da própria Microchip

Padrões textuais

Para facilitar o entendimento deste livro, alguns padrões textuais foram utilizados:

Itálico	Termo em língua estrangeira que ainda não foi popularmente adotado no Brasil,
Negrito	Nome de registrador, bit ou pino.
Registrador<bit>	Nome do registrador e nome do(s) bit(s) interno(s).

Anotações

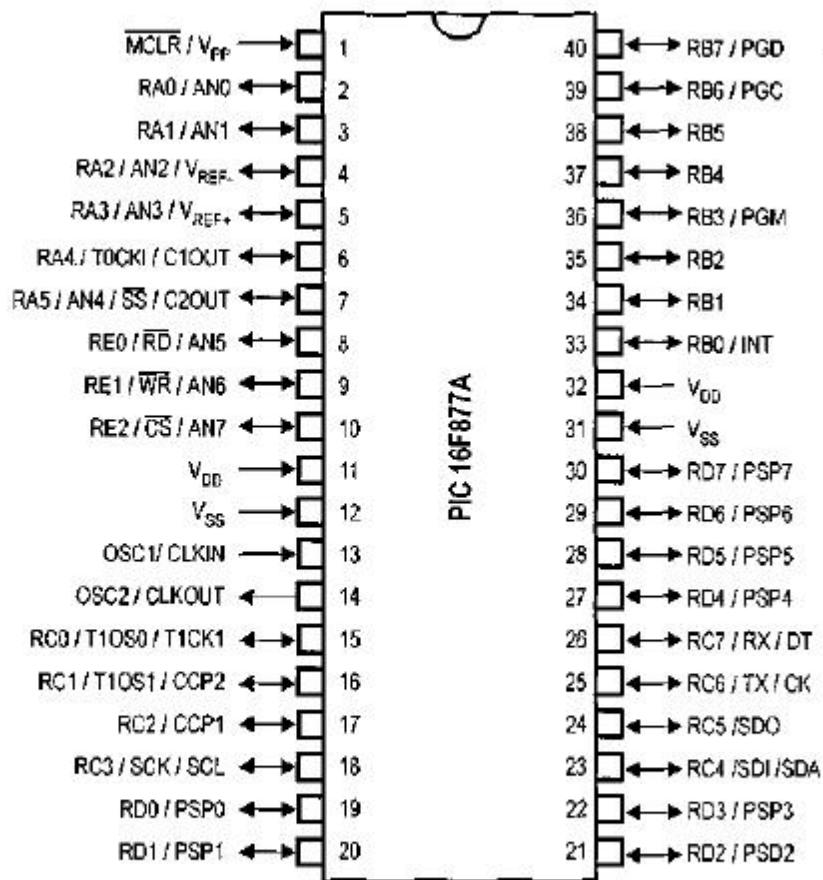
O PIC 16F877A

Para podermos conhecer os demais recursos da família PIC, devemos adotar como base de nossos estudos um modelo extremamente poderoso que agrupe de uma só vez o maior número possível de recursos disponíveis. Desta vez escolhemos trabalhar com o modelo 16F877A. Vejamos agora o porquê:

- Micro controlador de 40 pinos, o que possibilita a montagem de um hardware complexo e capaz de interagir com diversos recursos e funções ao mesmo tempo;
- Via de programação com 14 bits e 35 instruções;
- 33 portas configuráveis como entrada ou saída;
- 15 interrupções disponíveis;
- Memória de programação E PROM FLASH, que permite a gravação rápida 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 de programa com 8kwords, com capacidade de escrita e leitura pelo próprio código interno;
- Memória E PROM (não-volátil) interna com 256 bytes;
- Memória RAM com 368 bytes;
- Três Timers (2x8 bits e 1x16 bits);
- Comunicações seriais: SPI, PC e USART;
- Conversores analógicos de 10 bits (8x) e comparadores analógicos (2x);
- Dois módulos CCP: Capture, Compare e PWM;
- Programação in-circuit (alta e baixa tensão);
- Power-on Reset (POR) interno;
- Brown-out Reset (BOR) interno.

A grande vantagem da família PIC é que todos os modelos possuem um set de instruções bem parecido, assim como também mantêm muitas semelhanças entre suas características básicas. Desta maneira ao conhecermos e estudarmos o PIC 16F877A estaremos nos familiarizando com todos os Microcontroladores da Microchip, o que tornará a migração para outros modelos muito mais simples. Os recursos aqui estudados encontram-se disponíveis em diversos outros modelos.

Pinagem



Legenda: Entrada ou saída.

Somente entrada.

Somente saída.

Nomenclatura dos pinos

Para entendermos melhor o significado de cada nomenclatura utilizada para identificação dos pinos, montaremos uma tabela onde descreveremos detalhes da sua utilização.

Nome do Pino	Nº Pino	I/O/P	Tipo	Descrição
OSC1/CLKIN	13	I	ST/ CMOS ⁽⁴⁾	Entrada para cristal. Entrada para osciladores externos. (híbridos ou RC)
OSC2/CLKOUT	14	O	-	Saída para cristal. Os cristais ou ressonadores devem ser ligados ao pinos OSC1 e OSC2. Saída com onda quadrada em ¼ da freqüência imposta em OSC1 quando em modo RC. Essa freqüência equivale aos ciclos de máquina internos.

Nome do Pino	N.º Pino	I/O/P	Tipo	Descrição	
MCLR/Vpp	1	I/p	ST	Master Clear (reset) externo. O microcontrolador só funciona quando este pino encontra-se em nível alto. Entrada para tensão de programação (13V).	
Vss	12/3	p	-	GND.	
vdd	11/3 2	p	-	Alimentação positiva.	
RA0/AN0	2	I/O	TTL	RA0: I/O digital ou entrada analógica AN0.	
RA1/AN1	3	I/O	TTL	RA1 : I/O digital ou entrada analógica AN1 .	
RA2/AN2/ V _{ref} /CV _{ref}	4	I/O	TTL	RA2: I/O digital ou entrada analógica AN2 ou tensão negativa de referência analógica.	
RA3/AN3/V _{REF+}	5	I/O	TTL	RA3: I/O digital ou entrada analógica AN3 ou tensão positiva de referência analógica.	
RA4 / T0CKI / C1OUT	6	I/O	ST	RA4: I/O digital (quando saída é open drain, isto é, não consegue impor nível alto) ou entrada externa do contador TMR0 ou saída do comparador 1 .	
RA5/SS/AN4/ C2OUT	7	I/O	TTL	RA5: I/O digital ou entrada analógica AN4 ou habilitação externa (slave select) para comunicação SPI ou saída do comparador 2.	
				PORTB (I/Os digitais bidirecionais). Todos os pinos deste PORT possuem pull-up interno que podem ser ligados/desligados pelo software: RB0: I/O digital com interrupção externa. RB1: I/O digital. RB2: I/O digital. RB3: I/O digital ou entrada para programação em baixa tensão (5V). RB4: I/O digital com interrupção por mudança de estado. RB5: I/O digital com interrupção por mudança de estado. RB6/PGC	
RB6/PGC	39		TTL/ST	RB6: I/O digital com interrupção por mudança de estado ou clock da programação serial ou pino de in-circuit debugger. RB7: I/O digital com interrupção por mudança de estado ou data da programação serial ou pino de in-circuit debugger.	
RB7/PGD	40	I/O	TT17ST ⁽²⁾	PORTC (I/Os digitais bidirecionais): RC0: I/O digital ou saída do oscilador externo para TMR1 ou entrada de incremento para TMR1 . RC1: I/O digital ou entrada do oscilador externo para TMR1 ou entrada do Capture2 ou saídas para Compare2/PWM2. RC2/CCP1	
RC3/SCK/SCL	18	I/O	ST	RC2: I/O digital ou entrada do Capture1 ou saídas para Compare1/PWM1. RC3: I/O digital ou entrada/saída de clock para comunicação serial SPI / I ² C.	
RC4/SDI/SDA	23	I/O	ST	RC4: I/O digital ou entrada de dados para SPI ou via de dados (entrada/saída) para I ² C.	

Nome do Pino	Nº Pino	I/O/P	Tipo	Descrição
RC5/SDO	24	I/O	ST	RC5: I/O digital e saída de dados para SPI.
RC6/TX/CK	25	I/O	ST	RC6: I/O digital ou TX (transmissão) para comunicação USART assíncrona ou clock para comunicação síncrona.
RC7/RX/DT	26	I/O	ST	RC7: I/O digital ou RX (recepção) para comunicação USART assíncrona ou data para comunicação síncrona.
RD0/PSP0	19	I/O	TTL/ST ⁽³⁾	PORTD (I/Os digitais bidirecionais) ou porta de comunicação paralela. RD0: I/O digital ou dado 0 (comunicação paralela).
RD1/PSP1	20	I/O	TTL/ST ⁽³⁾	RD1: I/O digital ou dado 1 (comunicação paralela).
RD2/PSP2	21	I/O	TTL/ST ⁽³⁾	RD2: I/O digital ou dado 2 (comunicação paralela).
RD3/PSP3	22	I/O	TTL/ST ⁽³⁾	RD3: I/O digital ou dado 3 (comunicação paralela).
RD4/PSP4	27	I/O	TTL/ST ⁽³⁾	RD4: I/O digital ou dado 4 (comunicação paralela).
RD5/PSP5	28	I/O	TTL/ST ⁽³⁾	RD5: I/O digital ou dado 5 (comunicação paralela).
RD6/PSP6	29	I/O	TTL/ST ⁽³⁾	RD6: I/O digital ou dado 6 (comunicação paralela).
RD7/PSP7	30	I/O	TTL/ST ⁽³⁾	RD7: I/O digital ou dado 7 (comunicação paralela).
RE0/RD/AN5	8	I/O	TTL/ST ⁽³⁾	PORTE (I/Os digitais bidirecionais e sistema analógico): RE0: I/O digital ou controle de leitura da porta paralela ou entrada analógica AN5.
RE1/WR/AN6	9	I/O	TTL/ST ⁽³⁾	RE1: I/O digital ou controle de escrita da porta paralela ou entrada analógica AN6.
RE2/CS/AN7	10	I/O	TTL/ST ⁽³⁾	RE2: I/O digital ou habilitação externa da porta paralela ou entrada analógica AN7.

Legenda:

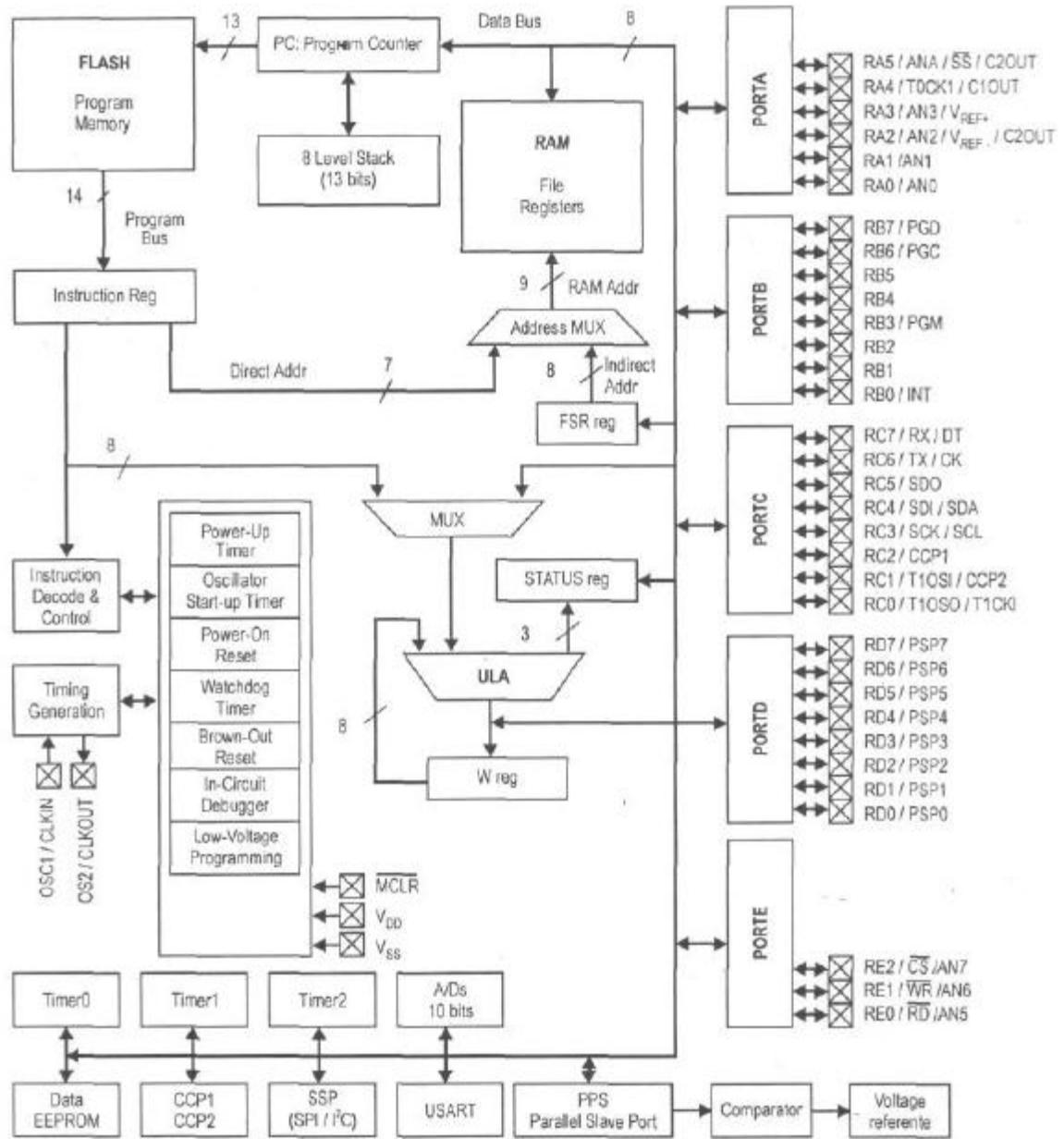
I	=	Input (entrada)
O	=	Output(saída)
I/O	=	Input/Output (entrada ou saída)
P	=	Power (alimentação)
-	=	Não-utilizado
TTL	=	Entrada tipo TTL
ST	=	Estrada tipo Schmitt Trigger

Notas:

- (1) Esta entrada é do tipo ST, somente quando configurado como interrupção externa.
- (2) Esta entrada é do tipo ST, somente durante o modo de programação serial.
- (3) Esta entrada é do tipo ST, quando configurado como I/O de uso geral e TTL quando usado em modo de porta paralela.
- (4) Esta entrada é ST quando em modo RC e CMOS nos demais casos.

A estruturação interna

Já que estamos nos aprofundando nas características do PIC 16F877A, que tal darmos uma olhadinha em seu interior? O que há por dentro deste componente? Como é sua estrutura? O diagrama de blocos, mostrado a seguir, detalhará todos os periféricos e comunicações que compõem esse tão poderoso microcontrolador:



No diagrama de blocos (retirado do data sheet original da Microchip) podem ser visualizadas as diversas partes que compõem o microcontrolador PIC16F877A. No centro encontramos a ULA (em inglês: ULA), que é a unidade de processamento e está diretamente ligada ao registrador Work (W reg). No canto superior esquerdo temos a memória de programa (FLASH) saindo desse bloco temos um barramento de 14 bits (Program Bus). Mais a direita está a memória de dados (RAM). Ela já possui um carregamento de 8 bits (Data Bus). Totalmente do lado direito encontram-se os PORTs, de PORTA a

PORTE. Na parte inferior podem ser encontrados os demais periféricos, tais como a E2PROM (memória de dados não-volátil), os timers (TMR0, TMR1 e TMR2), os A/Ds de 10 bits, os modos CCP (Compare, Capture e PWM), as comunicações seriais (SPI, I²C e USART), os comparadores e a tensão de referência. Observe que, entre todos os periféricos, a comunicação é feita através de um barramento de oito vias.

Um pouco mais ao centro, podemos encontrar ainda o registrador de status (STATUS reg), onde algumas informações importantes sobre as operações aritméticas da ULA ficam armazenadas, e os demais SFRs (registradores especiais). Na parte superior temos ainda o contador de linha de programa (Program Counter) e a pilha de oito níveis (Stack).

Temos ainda os circuitos internos de reset, POR, BOR, osciladores, Watchdog Timer (WDT) e sistema de programação.

Os ciclos de máquina

Para quem já trabalha com Microcontroladores, possivelmente este assunto já é conhecido, mas, devido à sua grande importância, achamos melhor revisá-lo e, quem sabe, aperfeiçoar seus conhecimentos.

O básico que deve ser conhecido é que, neste microcontrolador (assim como na maioria dos modelos da linha PIC), o clock interno (CK_{INT}) é equivalente ao clock externo (CK_{EXT} ou F_{osc}) dividido por 4. Com isso teremos:

$$CK_{INT} = CK_{EXT}$$

4

Desta forma, quando trabalhamos, por exemplo, com um cristal de 4 MHz, o PIC estará trabalhando internamente com uma freqüência de 1 MHz. Para nós, mais importante que o clock interno, é o período dessa freqüência (o inverso da mesma), que é equivalente ao tempo de duração de um ciclo de máquina, popularmente chamado de CM ou T_{CY} . Assim sendo teremos:

$$T_{CY} = \frac{1}{CK_{INT}}$$

CK_{INT}

Continuando com nosso exemplo de 4 MHz externo, nosso ciclo de máquina (CM) será de $1\mu s$.

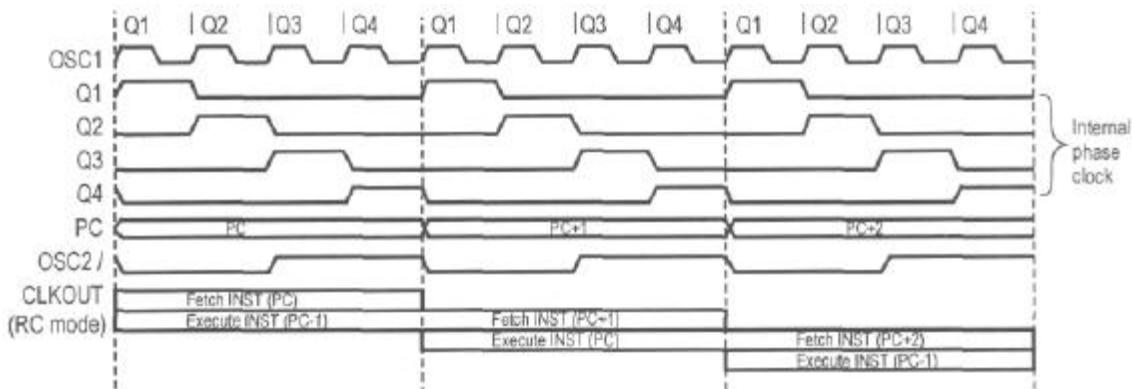
Mas, afinal de contas, qual o motivo da necessidade da divisão do clock externo por 4? Isso pode ser visualizado e explicado quando entendermos o funcionamento interno do processador.

Para a execução de uma única instrução pelo processador, várias operações precisam ser realizadas. Como não existe capacidade de processamento paralelo, essas operações são executadas em subciclos do ciclo de máquina, originados pela divisão do clock externo. Esses subciclos são chamados de Q1, Q2, Q3 e Q4.

O PC (contador de programa) é incrementado automaticamente no início de Q1. Durante o decorrer dos quatro tempos (Q1 a Q4), a instrução previamente carregada para dentro da ULA é executada, sendo trocadas informações com a memória de dados e o registrador Work sempre que necessário. Por último, ao final do tempo Q4, a próxima instrução (lembre-se de que o PC já foi incrementado) é buscada da memória de programa e armazenada na ULA.

Essa característica de buscar a informação em um 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 (continuando com nosso exemplo de 4 MHz) e tornando o sistema muito mais rápido. As únicas exceções referem-se às instruções que geram "saltos" no program counter (PC), 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 2 ciclos de máquina.

O diagrama seguinte ilustra os quatro subciclos (Q1 a Q4) e o conceito de Pipeline.



A memória de programa

A memória de programa do PIC 16F877A é de 14 bits do tipo FLASH, uma memória regravável eletronicamente com escrita rápida. Uma outra característica muito importante da memória de programa desse PIC é que ela pode ser acessada por software, possibilitando que o programa seja reescrito dinamicamente, ou, ainda, que ela seja usada como uma expansão da memória E_{PRÓM} de dados. Vamos conhecer, então, as demais características dessa memória:

Vetor de reset

Trata-se do primeiro endereço da memória de programa que será executado após um Start-up ou Reset. Neste modelo, o **Vetor de Reset** encontra-se no endereço 0x0000.

Vetor de interrupção

Este PIC possui 15 tipos de interrupções diferentes. Entretanto, quando qualquer uma delas acontece (se todas as condições necessárias forem favoráveis), o programa será desviado para um ponto específico, que é denominado **Vetor de Interrupção**. Como na maioria dos Microcontroladores PIC, para o modelo 16F877A este vetor encontra-se na posição 0x0004.

Tamanho da memória e páginas

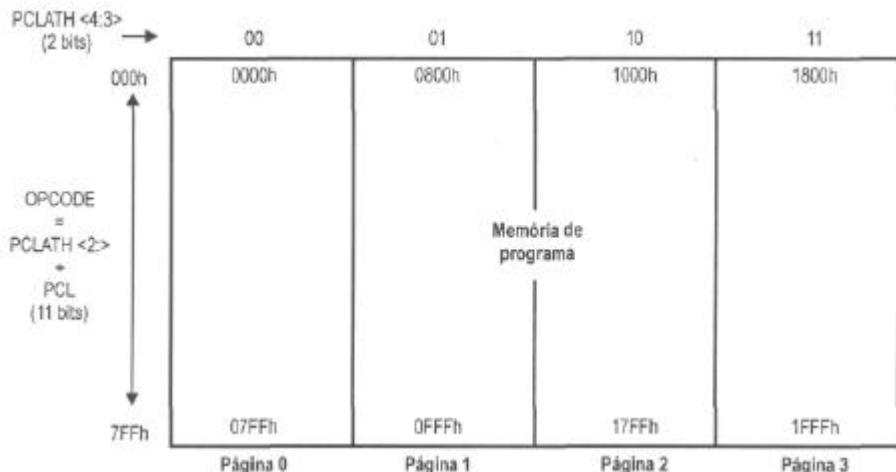
Este PIC possui uma memória de 8kwords, isto é, 8.192 endereços diferentes. Quando falamos da memória de programa sempre nos referimos ao termo WORD e nunca a BYTE, isso porque esta memória é de 14 bits, enquanto um byte possui somente 8 bits.

No entanto, o tamanho desta memória gera um grande problema de gerenciamento. Relembremos o funcionamento do processador. Cada instrução é carregada através do endereço armazenado no PC (program counter), correto? Assim sendo, para podermos acessar os 8.192 endereços diferentes, necessitaremos de um PC com 13 bits. Isso é resolvido por meio do desmembramento do PC em dois registradores (FSRs): **PCL** e **PCLATH**. Até ai tudo bem. Mas vamos checar outros casos que interferem no gerenciamento da memória de programa, como quando utilizamos as instruções CALL e GOTO. Essas instruções vinculam ao seu Opcode (código de máquina) de 14 bits e o endereço para onde o programa deve ser desviado. Vejamos esses Opcodes mais detalhadamente:

Instrução	Opcode	Observações
CALL	100 kkk kkkk kkkk	kkk kkkk kkkk equivale ao endereço de destino
GOTO	101 kkk kkkk kkkk	

Conhecendo os detalhes dos Opcodes (todos os Opcodes podem ser encontrados no apêndice B) percebemos que só existem 11 bits para o endereço de destino da instrução. Ora, mas com 11 bits só conseguimos gerenciar 2 K de memória. Então como é que podemos usar essas instruções com toda a memória de programa?

Para resolver esse problema foi criado o conceito de paginação. A memória total foi então dividida em várias páginas (no nosso caso 4) de 2 K cada uma. Dentro de cada página as instruções CALL e GOTO funcionam perfeitamente, pois possuem capacidade de gerenciamento para esse tamanho. O problema então passou a existir somente quando quisermos utilizar essas instruções com um endereço de destino localizado em outra página. A solução para isso é configurarmos manualmente os 2 bits faltantes localizado nas posições 3 e 4 do **PCLATH**. Estes bits passam, então, a operar como chaves seletoras da página ativa no momento.



Para podermos então utilizar as instruções CALL e GOTO entre páginas, devemos primeiro selecionar a página correta do endereço de destino através dos bits **PCLATH<4:3>**. O diagrama anterior demonstrou essa seleção.

Uma dica interessante é que a diretriz do compilador denominada **PAGESEL** pode ser utilizada para selecionar facilmente a página correta para um destino qualquer. Consulte o apêndice C para obter mais informações.

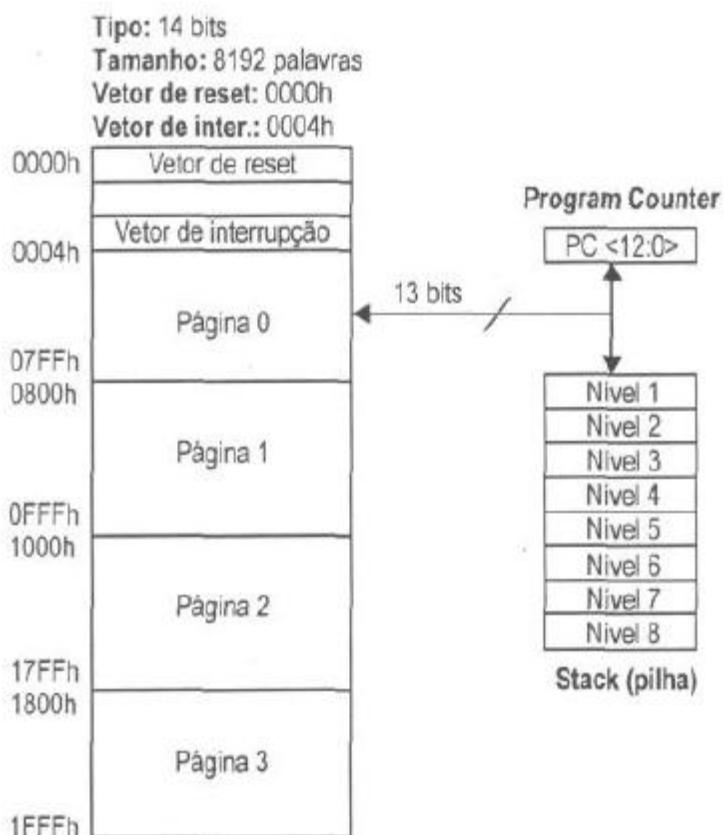
Pilha (Stack)

A pilha é um local, fisicamente separado da memória de programa, onde serão armazenados os endereços de retorno quando utilizarmos instruções de desvio para rotinas de chamada. Quando o programa é desviado para o começo de uma rotina através da instrução correta (CALL ou interrupção), o endereço seguinte ao ponto que estava sendo executado é armazenado na pilha para que, ao fim da rotina, o programa possa retornar novamente ao ponto em que estava.

Esse PIC possui uma pilha de oito níveis, isto é, é possível o armazenamento de oito endereços de retorno, possibilitando oito desvios consecutivos. Caso se tente chamar um número de rotinas maior que o tamanho da pilha, o endereço de retorno mais antigo será perdido.

Outra característica que devemos saber sobre a pilha é o seu tamanho em bits. Esse tamanho representa a quantidade de endereços que o sistema de retorno automático poderá gerenciar. No nosso caso, a pilha armazena endereços de 13 bits, o que é suficiente para o gerenciamento de até 8 K de memória de programa. Como nosso PIC possui exatamente 8 K, os retornos através da pilha não apresentam problemas com a paginação da memória de programa.

Mapa da memória de programa



A memória de dados volátil (RAM)

A memória de dados também é conhecida como RAM e serve para guardar as variáveis e os registradores utilizados pelo programa. Esta memória armazena dados de 8 bits e é volátil, ou seja, quando o PIC é desenergizado, ela é automaticamente perdida. Podemos dividi-la em dois grupos que serão estudados a seguir: **Registradores especiais** e **Registradores de uso geral**.

Registradoras especiais

Nesta região da memória encontram-se todos os registradores especiais, denominados SFRs (Special Functions Registers) e que são utilizados pelo microcontrolador para a execução do programa e processamentos da ULA. Esses registradores podem ser escritos e lidos tanto pelo usuário quanto pelo hardware e servem também para a configuração de muitas funções e para a utilização de todos os periféricos. O mapa da memória apresentará a posição de todos os registradores especiais e seus devidos nomes. Para um melhor detalhamento de suas propriedades e funções específicas, o apêndice A deve ser consultado.

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 PIC 16F877A possui 368 bytes disponíveis para uso geral.

Tamanho da memória e bancos

A arquitetura do nosso PIC está preparada para operar com uma RAM de até 512 bytes. Entretanto, verificaremos que nem toda memória está disponível ao usuário. São 368 bytes para uso geral, 77 bytes para registradores especiais (sendo alguns espelhados) e 19 indisponíveis. Os 48 endereços restantes (não fisicamente implementados) equivalem a posições espelhadas de 16 endereços válidos (Banco 0). Entenda por termo espelhado como um registrador único que pode ser acessado por mais de um endereço. Isso pode ser facilmente visualizado através do mapa da memória de dados, onde vemos um mesmo SFR aparecendo em diversos bancos (os bancos serão explicados em seguida). Esse espelhamento também acontece com a região inferior da memória de uso geral, para facilitar a vida do programador em relação a variáveis que são constantemente usadas, sem a necessidade de troca de banco.

Quanto aos endereços indisponíveis, tratam-se de posições não implementadas pela Microchip e que, no caso de uma leitura, retornarão somente zeros (0).

Muito bem, ao detalharmos o tamanho da memória de dados surgimos com o termo Banco. Mas, afinal, o que isso significa?

O mesmo tipo de problema de gerenciamento de endereços que enfrentamos ao trabalharmos com a memória de programas, surge novamente para a memória de dados. Vejamos os Opcodes de algumas instruções que trabalham com a RAM:

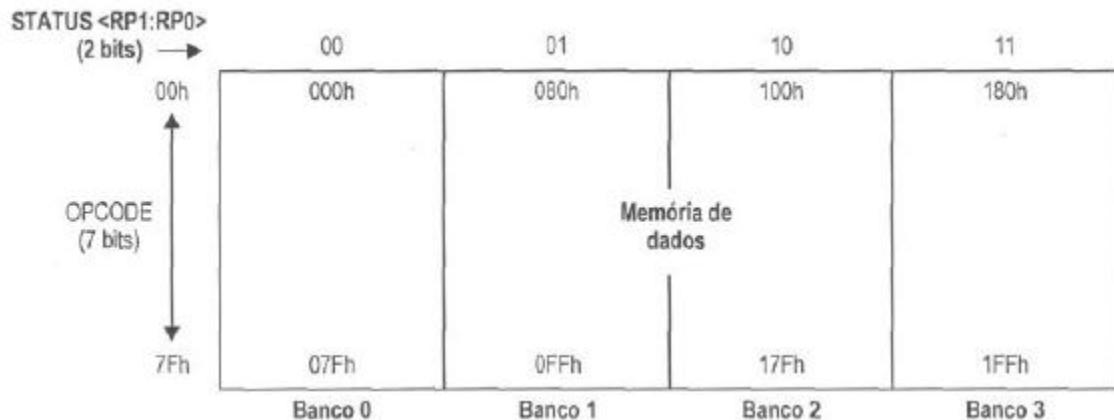
Instrução	Opcode	Observações
MOVF	00 1000 dfff ffff	d= destino
MOVWF	00 0000 1fff ffff	ffff= endereço da RAM

Fica fácil percebermos que só existem 7 bits para endereçamento de todos os registradores. Acontece que, com este número de bits, só podemos gerenciar 128 endereços diferentes. Por isso, a Microchip dividiu a memória de dados em grupos de 128 bytes, aos quais deu o nome de **Banco**, diferenciando assim da nomenclatura utilizada para a memória de programa (Página).

No caso do nosso PIC, a memória de dados é composta por quatro bancos. De maneira análoga ao sistema utilizado na memória de programa, aqui também os demais bits (2) necessários para o

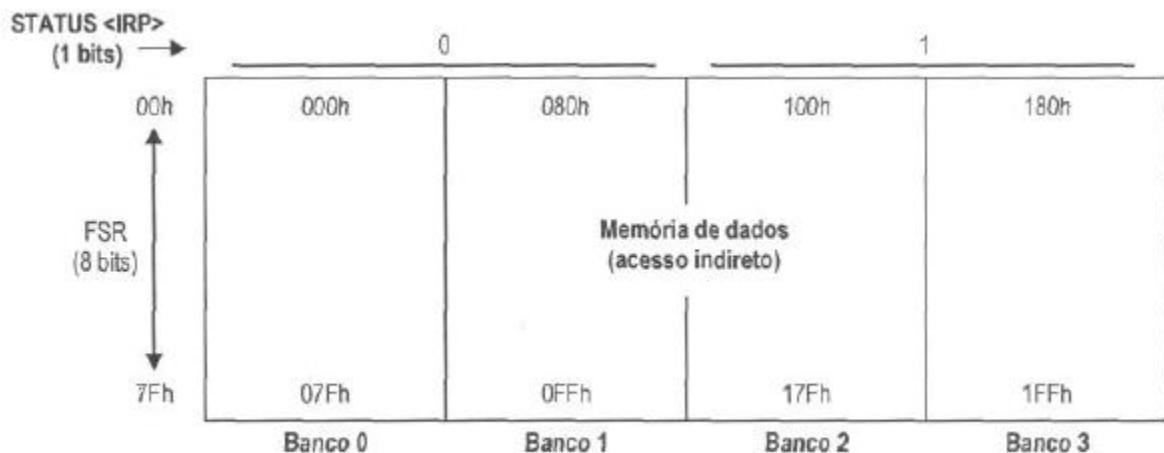
complemento do gerenciamento são utilizados como chaves seletoras para o banco atual e encontram-se no registrador **STATUS<RP1:RP0>**.

Devido a essa estruturação, antes de utilizarmos qualquer comando que trabalhe com os registradores (RAM) devemos comutar para o banco onde o mesmo se encontre. Como já foi comentado, registradores de uso muito comum podem ser armazenados nos endereços de 070h a 07Fh, para que possam ser acessados diretamente em qualquer um dos quatro bancos disponíveis (vide mapa da memória).



Para a seleção do banco de memória existe também uma diretriz do compilador que pode ser pesquisada no apêndice C: **BANKSEL**.

Já que estamos falando dos bancos de memória, existe um outro caso onde a seleção do banco é feita de maneira diferente. Trata-se do acesso indireto através dos registradores **FSR** e **INDF**. Acontece que, quando acessamos a memória desta forma, o endereço do registrador desejado será colocado em FSR, que possui 8 bits. Por isso, só falta 1 bit para complementarmos o gerenciamento. Neste caso, então, a chave seletora de banco será o bit **STATUS<IRP>**.



Mapa da memória de dados

	Banco 0	Banco 1	Banco 2	Banco 3
000h	INDF	INDF	INDF	INDF
001h	TMR0	OPTION_REG	TMR0	OPTION_REG
002h	PCL	PCL	PCL	PCL
003h	STATUS	STATUS	STATUS	STATUS
004h	FSR	FSR	FSR	FSR
005h	PORTA	TRISA	PORTB	TRISB
006h	PORTB	TRISB	PORTB	TRISB
007h	PORTC	TRISC	PORTB	
008h	PORTD	TRISO	PORTB	
009h	PORTE	TRISE	PORTB	
00Ah	PCLATH	PCLATH	PCLATH	PCLATH
00Bh	INTCON	INTCON	INTCON	INTCON
00Ch	PIR1	PIE1	EEDATA	EECON1
00Dh	PIR2	PIE2	EEADR	EECON2
00Eh	TMR1L	PCON	EEDATH	Reservado
00Fh	TMR2H		EEADRH	Reservado
010h	T1CON			
011h	TMR2	SSPCON2		
012h	T2CON	PR2		
013h	SSPBUF	SSPADD		
014h	SSPCON	SSPSTAT		
015h	CCPR1L			
016h	CCPR1H			
017h	CCP1CON			
018h	RCSTA	TXSTA		
019h	TSREG	SPBRG		
01Ah	TXREG			
01Bh	CCPR2L			
01Ch	CCPR2H			
01Dh	CCP2CON			
01Eh	ADRESH	CMCON		
01Fh	ADCON0	CRVCON		
020h				
	Uso Geral 96 bytes			
07Fh		0EFh 0F0h OFFh	11Fh 120h 16Fh 170h 17Fh	19Fh 1A0h Uso Geral 80 bytes Espelho do Banco 0
				1EFh 1F0h 1FFh Espelho do Banco 0

 Não implementado

O detalhamento de todos os SFRs podem ser encontrados no apêndice A.

Memórias não-voláteis (E²PROM e FLASH)

A memória não-volátil até poderia ser tratada diretamente dentro do tópico anterior, no qual falávamos da memória de dados. Entretanto, para o PIC 16F877A, o acesso a essa memória é mais poderoso. Em alguns modelos de PIC, quando falamos dos recursos internos para manipulação desta

memória, nos referimos somente a um bloco de bytes disponíveis para o usuário como urna memória de dados não-volátil, conhecida como E²PROM. Esse PIC possui este bloco e seu acesso é feito como nos demais modelos. O que ele tem a mais então? É que o mesmo sistema de manipulação da E²PROM também pode ser utilizado para escrever e ler na memória de programa (FLASH).

Com esse poderoso recurso, podemos utilizar a memória de programa como uma expansão da E²PROM, com uma outra vantagem: a memória de programa é de 14 bits, enquanto a memória de dados é de 8 bits. Outro benefício desse sistema é que podemos escrever na memória de programa também, possibilitando alterarmos o próprio software de controle. Isso possibilita técnicas e recursos extremamente avançados. Mas cuidado! Este benefício também pode ser uma arma letal, pois podemos alterar o programa erroneamente, travando o sistema permanentemente.

A única diferença para acessarmos uma memória ou a outra é o bit de seleção denominado **EECON1< EEPGD>**

As técnicas e os exemplos para acessarmos estas memórias serão vistas no capítulo 10. No momento, basta conhecermos as características de ambas:

Características	Dados	Programa
Tipo	8 bits	14 bits
Tamanho	256 bytes	8 K (menos o programa)
Acesso	EPPGD=0	EPPGD=1

As interrupções

Este PIC possui um total de 15 interrupções diferentes, Entretanto, como nos demais modelos, todas gerarão o desvio do programa para o mesmo vetor de interrupção (0004h), desde que devidamente configuradas para isso. Essas interrupções podem ser divididas em dois grupos: as convencionais (as mesmas existentes em modelos menores, como o 16F84), e as de periféricos, que estão diretamente relacionadas aos demais periféricos existentes neste modelo. As três interrupções descritas a seguir são as convencionais, enquanto as demais são as de periféricos.

Interrupção de Timer 0

Esta interrupção acontece sempre que o contador **TMR0**(Timer 0) estoura, ou seja, como ele é um contador de 8 bits, sempre que ele passar de 0xFF para 0x00. Esta interrupção é utilizada normalmente para a contagem de tempo. Como ela 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. Os Timers serão estudados no capítulo 4.

Interrupção externa

Esta interrupção é gerada por um sinal externo ligado ao pino **RB0**, caso ele esteja configurado como entrada. Desta maneira, podemos identificar e processar imediatamente um sinal externo. Ela é utilizada para diversas finalidades, como por exemplo, para a comunicação entre micros, garantindo o sincronismo, para o reconhecimento de botão ou outro sinal círculo sistema que necessite de uma ação imediata.

Esta interrupção pode ser configurada para a borda de subida ou para a borda de descida.

Interrupção por mudança de estado

Ao contrário da interrupção externa, a interrupção por mudança de estado acontece em ambas as bordas (subida e descida). Na verdade, esta interrupção é sensível a diferença de nível existente entre o pino e o latch interno. Esta interrupção está relacionada às portas **RB4, RB5, RB6 e RB7**, simultaneamente. Por isso, se estas portas forem configuradas como entradas, a mudança de estado em qualquer uma delas irá gerar a interrupção. Como o latch só é atualizado quando a porta é lida, a leitura é obrigatória para que o evento da interrupção pare de ocorrer. Esse tipo de interrupção pode ser utilizado, por exemplo, para criar um sincronismo com a rede de 60 Hz, para o controle de um triac ou outro sistema semelhante.

Interrupção da porta paralela (PSP)

Esta interrupção está diretamente ligada a porta paralela do tipo escravo (PSP - Parallel Slave Port) e acontece sempre que uma operação de escrita ou leitura desta porta é completada. O estudo completo deste recurso será realizado no capítulo 13.

Interrupção dos conversores A/D

Esta interrupção acontece quanto uma conversão A/D (Analógica/Digital) é completada. O estudo completo deste recurso será realizado no capítulo 7.

Interrupção de recepção da USART

Esta interrupção indica o término da recepção de um dado pela USART (Universal Synchronous Asynchronous Receiver Transmitter). O estudo completo deste recurso será realizado no capítulo 12.

Interrupção de transmissão da USART

Esta interrupção indica o esvaziamento do buffer relacionado à transmissão de um dado pela USART (Universal Synchronous Asynchronous Receiver Transmitter). O estudo completo deste recurso será realizado no capítulo 12.

Interrupção da comunicação serial (SPI e I^C)

O segundo sistema de comunicação serial deste PIC, além da USART, é denominado MSSP (Master Synchronous Serial Port), e possui dois modos de comunicação: SPI e I^C. Sempre que um dado é transmitido ou recebido por esta porta serial, esta interrupção acontecerá. Ela também acontecerá em muitas outras situações, que serão estudadas no capítulo 11.

Interrupção do CCP1 (Capture/Compare/PWM)

Esta interrupção está vinculada ao primeiro sistema CCP, que engloba os seguintes recursos: Capture, Compare! e PWM1. O estudo completo deste recurso será realizado no capítulo 9.

Interrupção do Timer 2

Esta interrupção acontecerá sempre que o Timer 2 (**TMR2**) acrescido do seu Postscaler estourar. Este é um contador de 8 bits que possui um Prescaler para incrementar o registrador e um Postscaler para gerar a interrupção. Os Timers serão estudados no capítulo 4.

Interrupção do Timer 1

Esta interrupção acontecerá sempre que o Timer 1 (**TMR1**) estourar. Este é um contador de 16 bits. Os Timers serão estudados no capítulo 4.

Interrupção de Um de escrita na E²PROM/FLASH

Como já foi visto anteriormente, alguns PICs possuem uma memória E²PROM interna. Esta interrupção serve para detectarmos o final de uma rotina de escrita nesta memória. A utilização da interrupção não é obrigatória para que a escrita funcione, mas, como a E²PROM é lenta na hora de escrever, em alguns sistemas sua utilização pode ser necessária para evitar que o programa pare durante a escrita na E²PROM. A interrupção também é válida para a memória FLASH (programa).

Interrupção de colisão de dados (Bus Collision)

Esta interrupção serve para informar o sistema sobre colisões de dados na comunicação I²C. O estudo completo deste recurso será realizado no capítulo 11.

Interrupção do CCP2 (Capture/Compare/PWM)

Esta interrupção está vinculada ao segundo sistema CCP, que engloba os seguintes recursos: Capture2, Compare2 e PWM2. O estudo completo deste recurso será realizado no capítulo 9.

Interrupção dos comparadores

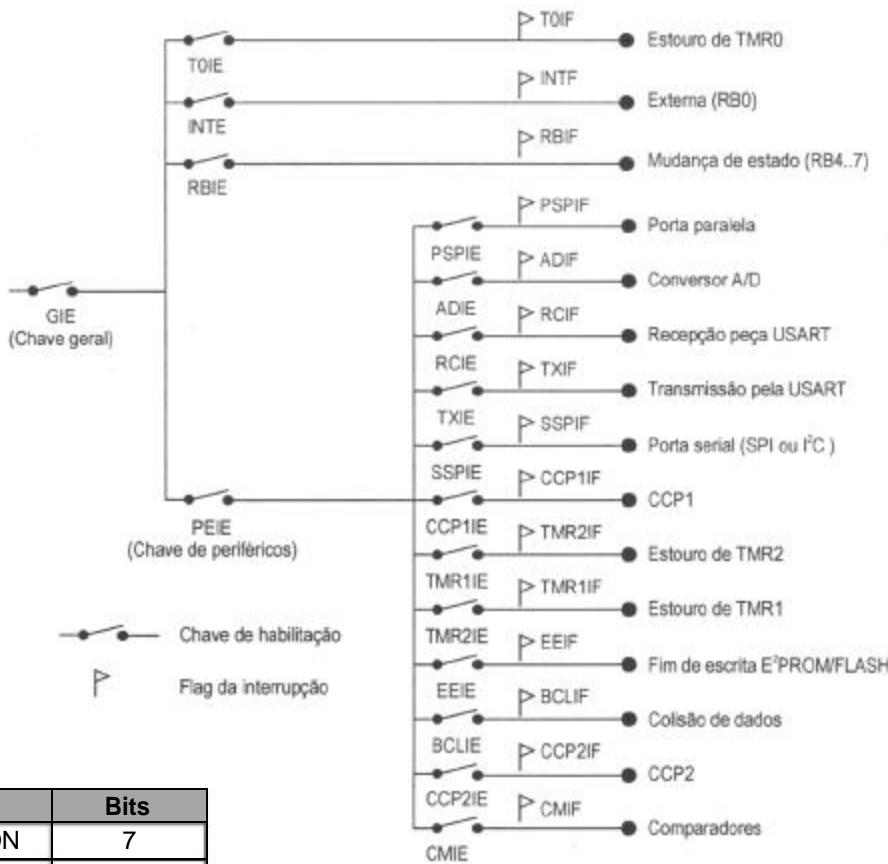
Esta interrupção está diretamente ligada a mudar de estado dos comparadores internos do PIC. Este recurso não será estudado em detalhes no decorrer deste livro.

Operando com as interrupções

O primeiro conceito que devemos entender para o uso das Interrupções é que, para elas realmente acontecerem, isto é, paralisarem o programa e desviarem para o vetor de interrupção, algumas questões devem ser verificadas:

1. Uma ação (ou evento) relacionada à interrupção deve acontecer.
2. Quando a ação acontece, o flag da interrupção é marcado.
3. Caso as chaves de habilitação da interrupção (geral e individual) estejam ligadas, o sistema será paralisado e desviado para o vetor de interrupção.

A ação propriamente dita é exatamente o que queremos monitorar. Mas e as chaves de acesso? Tratam-se de bits específicos que servem como chaves ON/OFF para ligar e desligar as interrupções, individualmente ou em grupo. O próximo diagrama ilustra todas as chaves e flags relacionados às interrupções.



Chaves	SRF	Bits
GIE	INTCON	7
PEIE	INTCON	6
TOIE	INTCON	5
INTE	INTCON	4
RBIE	INTCON	3
PSPIE	PIE1	7
ADIE	PIE1	6
RCIE	PIE1	5
TXIE	PIE1	4
SSPIE	PIE1	3
CCP1IE	PIE1	2
TMR2IE	PIE1	1

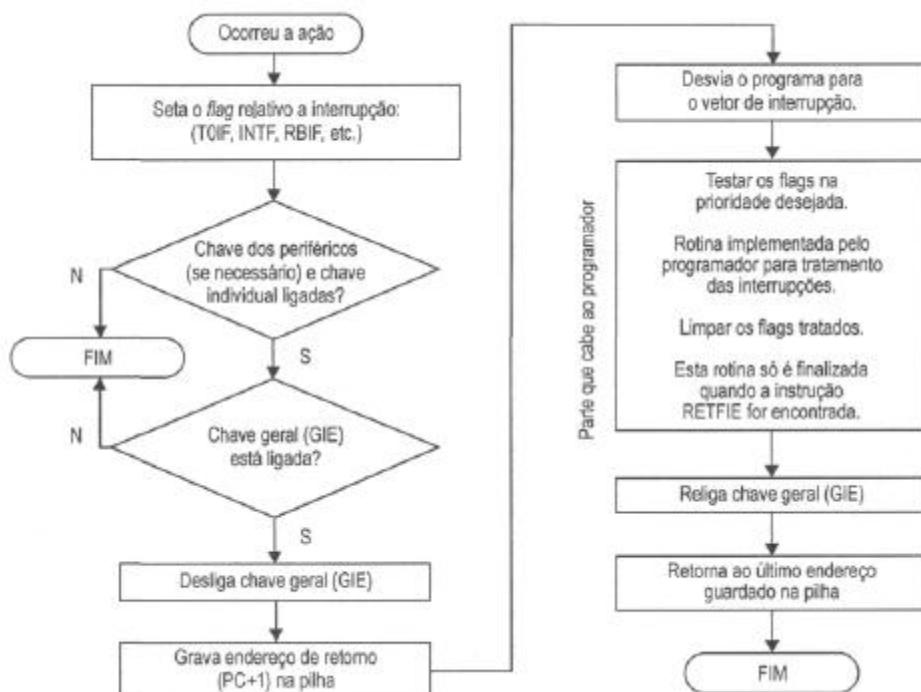
Chaves	SRF	Bits
-	-	-
-	-	-
TOIF	INTCON	2
INTF	INTCON	1
RBIF	INTCON	0
PSPIF	PIR1	7
ADIF	PIR1	6
RCIF	PIR1	5
TXIF	PIR1	4
SSPIF	PIR1	3
CCP1IF	PIR1	2
TMR2IF	PIR1	1

Chaves	SRF	Bits
TMR1IE	PIE1	0
EEIE	PIE2	4
BCLIE	PIE3	3
CCP2IE	PIE4	0

Flags	SRF	Bits
TMR1IF	PIR1	0
EEIF	PIR2	4
BCLIF	PIR3	3
CCP2IF	PIR4	0

Agora que todas as chaves e flags foram demonstradas, o fluxograma seguinte ajuda a entender a lógica de tratamento de uma interrupção. Atente-se para os seguintes pontos:

1. Como a chave **GIE** é desligada antes de desviar para o vetor de interrupção, duas interrupções não serão tratadas ao mesmo tempo, isto é, uma interrupção não gerará um desvio caso outra interrupção já esteja sendo tratada. Entretanto, o flag da segunda interrupção será marcado, e quando o tratamento da primeira terminar, o **GIE** será novamente ligado (através da instrução RETFIE) e o sistema voltará a ser desviado para o vetor de interrupção (devido ao flag).
2. Como todas as interrupções desviam para o mesmo ponto, é necessário testar os flags de todas as interrupções ligadas para saber qual realmente ocorreu. Como mais de uma ação vinculada às interrupções podem acontecer ao mesmo tempo, a ordem dos testes é que determina a prioridade de tratamento.
3. Os flags das interrupções não são limpos automaticamente pelo sistema (exceto **ADIF**, **TXIF** e **RCIF**). Cabe ao usuário efetuar esta operação na rotina de tratamento da interrupção.



Demais recursos e periféricos

Todos os demais recursos e periféricos que estão disponíveis no PIC 16F877A serão detalhados e estudados no decorrer dos capítulos (de 4 a 13). A tabela seguinte relaciona todos os itens a serem estudados e os respectivos capítulos onde os mesmos aparecem detalhadamente.

Assunto	Capítulos										
	4	5	6	7	8	9	10	11	12	13	
Portas	✓										
Porta Paralela (PSP)											✓
E2PROM (dados)								✓			
FLASH (programa)								✓			
Timer 0	✓										
Timer 1	✓										
Timer 2	✓										
CCP (Capture/Compare/PWM)							✓				
MSSP (SPI e I ² C)									✓		
USART										✓	
Conversor A/D						✓					
Interrupção de Timer 0	✓										
Interrupção de Timer 1	✓										
Interrupção de Timer 2	✓										
Interrupção Externa (RB0)											
Interrupção de mudança de estado (RB4...7)											
Interrupção da porta paralela (PSP)										✓	
Interrupção dos conversores A/D				✓							
Interrupção de recepção da USART										✓	
Interrupção de transmissão da USART										✓	
Interrupção da comunicação MSSP (SPI e I ² C)									✓		
Interrupção de CCP1							✓				
Interrupção de CCP2						✓					
Interrupção de fim de escrita na E2PROM/FLASH								✓			
Interrupção de colisão de dados (<i>Bus Collision</i>)									✓		
Osciladores											✓
Sistemas de Resets (POR, PWRT e BOR)											✓
Watchdog Timer (WDT)											✓
SLEEP											✓
Code Protection											✓
Gravação In-circuit											✓

Outros Recursos Estudados										
Assunto	Capítulos									
	4	5	6	7	8	9	10	11	12	13
Varredura de display de 7 segmentos (4 dígitos)	✓									
Operação com LCD padrão			✓							
Comunicação com memória E PROM externa								✓		
Simulação de A/D por meio de RC					✓					
Acionamento de Buzzer	✓									
Entrada e saída no mesmo pino (Botões e Leds)	✓									
Acionamento de motor DC (Ventilador)						✓				

Características elétricas

Temperatura de trabalho.....	-55°Caté+125°C
Temperatura de armazenamento.....	-65'Caté 150°C
Tensão de trabalho.....	4.0V a 5.5V
Tensão máxima no pino VDD (em relação ao V _{ss}).....	-0.3V até 7.5V
Tensão máxima no pino MCRL (em relação ao V _{ss}).....	0 até 14V
Tensão máxima no pino RA4 (em relação ao V _{ss}).....	0 até 8.5V
Tensão máxima nos demais pinos (em relação ao V _{ss}).....	-0.3V até (VDD + 0.3V)
Dissipação máxima de energia.....	1.0W
Corrente máxima de saída no pino V _{ss}	300mA
Corrente máxima de entrada no pino V _{DD}	250mA
Corrente máxima de entrada de um pino (quando em V _{ss}).....	25nA
Corrente máxima de saída de um pino (quando em V _{DD}).....	25mA
Corrente máxima de entrada do PORTA, PORTB e PORTE combinados.....	200mA
Corrente máxima de saída do PORTA, PORTB e PORTE combinados.....	200mA
Corrente máxima de entrada do PORTC e PORTD combinados	200mA
Corrente máxima de saída do PORTC e PORTD combinados.....	200mA

Anotações

Resumo do Set de Instruções

Os termos utilizados

Para facilitar o aprendizado do set de instruções do PIC é conveniente entendermos corretamente os termos utilizados na construção dos nomes das instruções e seus argumentos.

- **Work:** Trata-se de um registrador temporário para as operações da ULA. No assembly 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 argumentos das mesmas.
- **Literal:** Um número qualquer, que pode ser escrito na forma decimal, hexadecimal ou binário.
Utilizaremos a letra L para sua representação nos nomes de instruções e k nos argumentos das mesmas.
- **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 expressado 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 leira B para sua representação nos nomes das instruções e b nos argumentos das mesmas.
- **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,

- **ADD:** Somatória.
- **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 demonstrados, será muito mais fácil entender o significado de uma instrução por intermédio do seu nome, pois ele é composto pela junção destes 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**) = **DEC F**

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**)

O resumo das instruções

O Set de instrução do PIC 16F877A é exatamente o mesmo para todos os modelos de 14 bits, e é composto de 35 instruções divididas em quatro grupos: operações com registradores, operações com literais, operações com bits e controles,

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.
CRLF	f	Limpa f.
CLRW	-	Limpa W.

Operações com Registradores		
Instrução	Argumentos	Descrição
COMF	f,d	Calcula o complemento de f, guardando o resultado em d.
DECF	f,d	Decrementa 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.
MOVWF	f,d	Move W para f.
RLF	f,d	Rotaciona f 1 bit para esquerda, guardando o resultado em d.
RRF	f,d	Rotaciona f 1 bit para direita, guardando o resultado em d.
SUBWF	f,d	Subtrai W de f (f-W), guardando o resultado em d.
SWAPF	f,d	Executa uma inversão em 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 em 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) no bit do registrador f.
BSF	f,b	Impõe 1 (um) no bit do registrador f.
BTFSC	f,b	Testa o bit do registrador f, e pula a próxima linha se ele for 0 (zero).
BTFSS	f,b	Testa o bit do registrador f, e pula a próxima linha se ele for 01(um).

Controles		
Instrução	Argumentos	Descrição
NOP	-	Gasta um ciclo de máquina sem fazer absolutamente nada.
CALL	R	Executa a rotina R.
CLRWDT	-	Limpa o registrador WDT para não acontecer o <i>reset</i>
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 <i>sleep</i> (dormindo)para economia de energia.

As Primeiras Explorações (I/Os e Timers)

Introdução

O objetivo deste capítulo é pegarmos um pouco mais de intimidade com o PIC 16F877A. Quem aprendeu o assembly da Microchip com um PIC menor, provavelmente um de 18 pinos (16F84 ou "16F628, por exemplo), já deve ter percebido que, apesar da diferença de tamanho e quantidade de recursos, o set de instruções de todos esses modelos é exatamente o mesmo. Assim sendo, todo o seu conhecimento adquirido com a programação de outros PICs será de grande valia neste treinamento. Mas, por enquanto, vamos começar com o básico. E nada mais básico do que os I/Os do microcontrolador. Mas convenhamos que, apesar do 16F877A possuir 33 I/Os, um capítulo inteiro dedicado a estes é muita coisa. Para aproveitarmos melhor este estudo, vamos aprender também como maximizar estes I/Os, utilizando um mesmo pino hora como entrada, hora como saída.

Vamos aproveitar também a oportunidade para nos aprofundarmos nos 3 Timers existentes neste modelo.

Teoria e recursos do PIC

O capítulo será dividido em três tópicos principais: I/Os e Timers. Cada um desses assuntos merece considerações particulares, tanto na parte teórica quanto na parte prática, já que utilizaremos o mesmo exemplo para demonstrarmos os dois.

Estudo das PORTAS

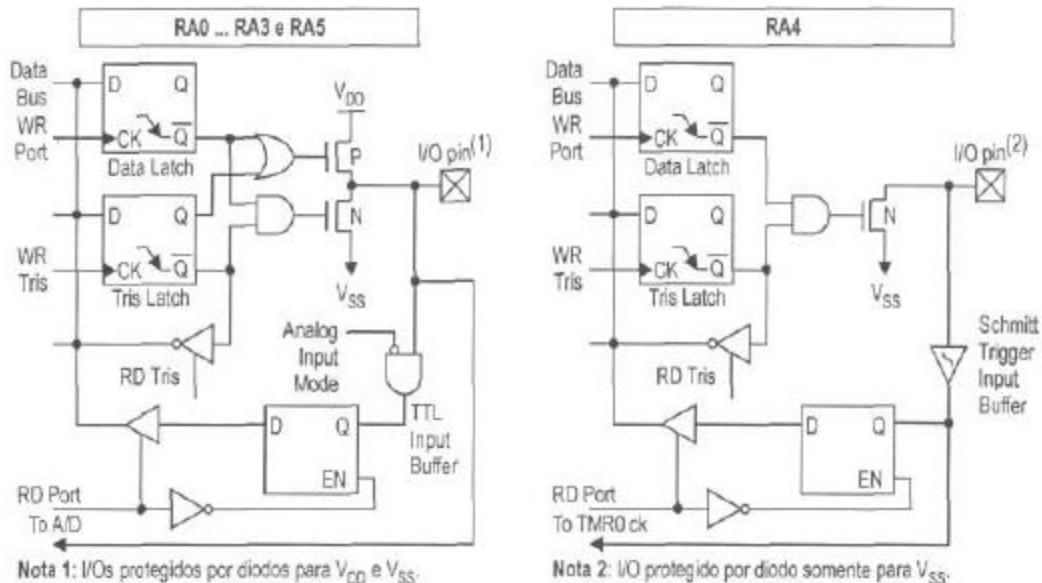
A primeira coisa que devemos saber sobre os I/Os de um microcontrolador é que eles são agrupados por PORTs. Este agrupamento se dá geralmente por características elétricas particulares e rara facilitar o gerenciamento da máquina. Por exemplo, como toda a memória de dados interna é de 8 bits, os SFRs também são. Como existem SFRs específicos para controlar os I/Os, é melhor agrupá-los em conjuntos de oito. Desta forma, o 16F877A possui seus I/Os divididos da seguinte forma:

- 1 PORT de 6 I/Os (PORTA);
- 3 PORTs de 8 I/Os (PORTB, PORTC e PORTD);
- 1 PORT de 3 I/Os (PORTE).

Ao total temos então 33 I/Os que podem ser configurados como entrada ou saída pelo programador, conforme as necessidades do projeto. Outras características importantes que devemos saber sobre essas portas dizem respeito à sua operação elétrica. Internamente esses pinos são ligados de formas diferentes, principalmente pela sobrecarga de recursos aplicados a cada um deles. Vamos conhecer melhor cada uma delas.

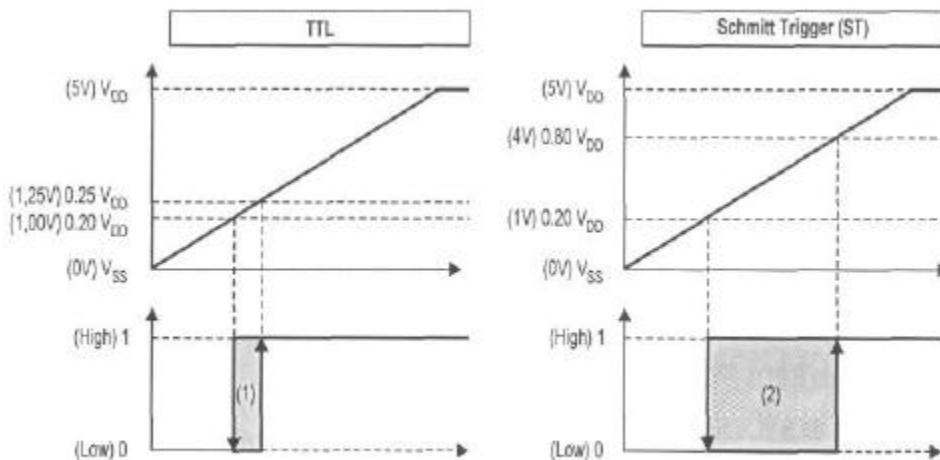
PORTE

Analizando os diagramas de blocos das ligações internas conheceremos melhor cada uma das características importantes:



Nos diagramas, a WR representa a escrita (Write) e a RD representa a leitura (Read).

O primeiro ponto a ser observado é que algumas portas são do tipo TTL e outras são do tipo Schmitt Trigger. Este dado é muito relevante quando estamos operando com uma porta como entrada, porque interfere diretamente nos níveis de tensão interpretados pelo PIC como 0 (zero) e 1 (um). Vejamos como isso acontece.



Nota 1: Região incerta. Mantém nível anterior.

Nota 2: Região incerta. Mantém nível anterior.

Repare, então, que nas entradas ST necessitamos de um nível de tensão bem mais elevado para que o PIC interprete a mudança de estado. Isso é muito útil para enquadrarmos uma senóide, por exemplo. Por outro lado, isso é ruim quando possuímos um hardware que gera níveis de tensão menores, como pode acontecer quando sobrecarregamos o mesmo pino com diversas funções. Essa explicação sobre TTL e ST é aplicável a todas as portas.

Outra característica comum a todos os I/Os é quanto as diferenças existentes entre escrita e leitura dos mesmos. Repare que a leitura é feita diretamente sobre o pino, enquanto a escrita passa por um Latch. Isso pode ocasionar um atraso entre o comando de escrita e a efetiva alteração da saída. É por esse motivo que não é recomendável efetuar uma operação de leitura imediatamente após a alteração uma saída. O certo é aguardarmos pelo menos um ciclo de máquina entre essas duas operações. Este Latch possibilita também que uma operação de escrita seja executada mesmo que o pino esteja configurado como entrada, através do TRIS. O Latch é alterado e, quando o estado do pino for modificado para saída, o valor atual do Latch lhe será imposto. Lembre-se de que a configuração das entradas e saídas deste port é feita através do registrador **TRISA**, enquanto as operações de escrita e leitura são executadas pelo registrador **PORTA**.

Para os pinos de **RA0** a **RA3** e o **RA5**, o diagrama mostra também uma ligação para o conversor analógico/digital (A/D). "Analog Input Mode" representa a seleção entre modo digital ou analógico para esses pinos. Um capítulo será completamente dedicado ao sistema de conversão, mas a seleção entre as portas digitais ou analógicas é realizada pelo registrador **ADCON1**,

Em contrapartida, o pino **RA4** não possui circuito analógico, mas é ligado internamente ao sistema de incremento do Timer 0 (**TMR0**). Isso possibilita que um clock imposto a esse pino incremente automaticamente o TMRO, independentemente do clock real da máquina. O registrador onde se encontra a configuração para incremento do **TMR0** é o **OPTION_REG**. Outra característica elétrica que pode ser observada no diagrama é que este pino não possui o FET do tipo P capaz de impor VDD à saída. Por isso, essa saída é denominada *open drain*. Isso significa que o PIC não é capaz de liberar V só (5V) em **RA4**. Caso isso seja necessário, um resistor de *pull-up* externo deve ser providenciado.

Resumo dos registradores associados ao PORTA

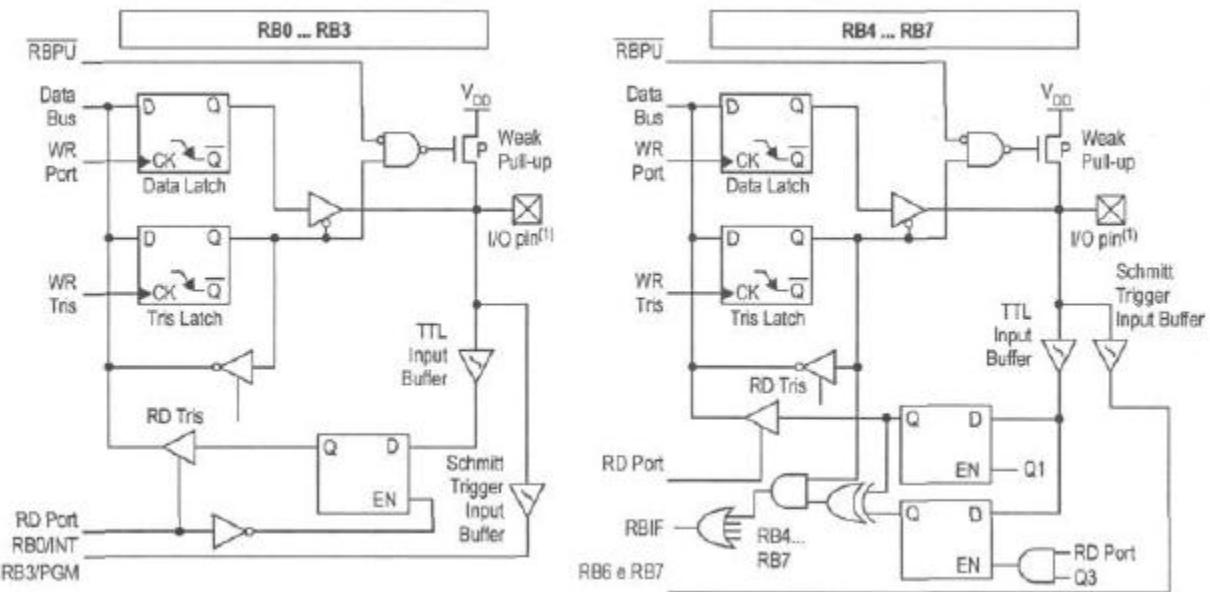
Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
05h	PORTA	-	-	RA5	RA4	RA3	RA2	RA1	RA0
85h	TRISA	-	-	Configuração como Entrada (1) ou Saída (0)					
9Fh	ADCON1	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
81h/181h	OPTION_REG	/RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

PORTB

O **PORTB** possui 8 I/Os, estando 5 deles associados também às interrupções: externa e mudança de estado. Na próxima figura podemos observar os diagramas deste PORT.

Uma característica elétrica bem interessante para todos os pinos do **PORTB** é a ligação do *pull-up* interno. Esta ligação para V_{DD} pode ser ligada ou não (em todos os pinos ao mesmo tempo) através da chave **/RBPU** localizada em **OPTION_REG**. Além da ativação desta chave, o pino deve ser configurado como entrada para poder ter seu *pull-up* ativado. A resistência interna para V_{DD} é de aproximadamente $12,5\text{k}\Omega$, para uma corrente máxima de $400\mu\text{A}$. Na verdade, o *pull-up* é feito por intermédio de um FET e não de um resistor, e sua corrente típica é de $250\mu\text{A}$, o que aumenta a resistividade.



Nota 1: I/Os protegidos por diodos para V_{DD} e V_{SS} quando em modo de programa.

Todo o **PORTB** é do tipo TTL, exceto o pino **RB0** quando configurado para utilizar a interrupção externa e os pinos **RB3**, **RB6** e **RB7** quando em modo de programação serial. Neste caso, estes pinos tornam-se ST.

Os pinos de **RB4** a **RB7** são interligados à interrupção de mudança de estado e o pino **RB0** é associado à interrupção externa, desde que configurados como entrada. As chaves e flags dessas interrupções encontram-se em **INTCON**. A configuração da borda que gera a interrupção externa (**RB0**) é feita em **OPTION_REG<INTEDG>**.

Resumo dos registradores associados ao PORTB

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
06h/106h	PORTB	RB 7	RB 6	RB5	RB4	RB3	RB2	RB1	RB0
86h/186h									
TRISB Configuração como Entrada (1) ou Saída (0)									
0Bh...	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
81h/181h	OPTION_REG	/RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

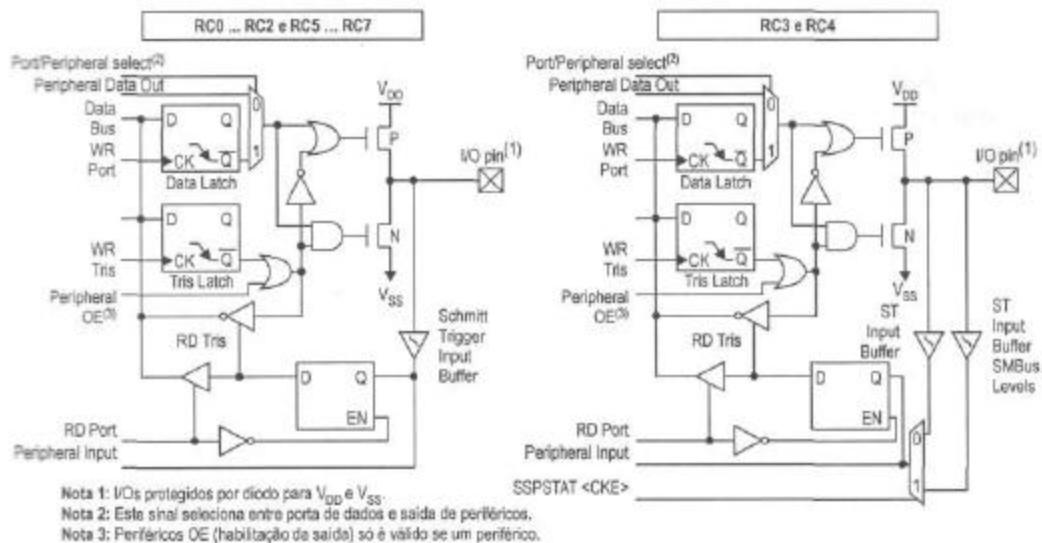
PORTE

Neste PORT, além dos 8 I/Os digitais, existe uma série de periféricos ligados aos seus pinos. Cada um desses periféricos será estudado em um capítulo específico. No momento, vejamos uma simplificação das suas ligações internas através dos próximos diagramas.

Resumo dos registradores associados ao PORTE

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
07h	PORTE	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
87h									
TRISC Configuração como Entrada (1) ou Saída (0)									

Não usado para essa finalidade. Para obter mais informações, consulte apêndice A



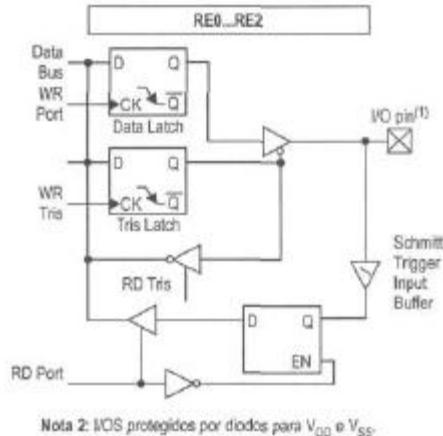
Todos os pinos deste PORT são do tipo ST quando configurados como entrada.

PORTD

O **PORTD** é um dos mais simples de todos, pois seus oito pinos são I/Os digitais. Além disso, estes pinos só são ligados ao sistema de porta paralela (Parallel Slave Port) que será estudado no capítulo 13.

Observe que, de acordo com o diagrama ao lado, todos os pinos do PORTD são do tipo ST quando configurados para entrada. Entretanto, para o sistema de porta paralela, esses pinos operam em modo TTL.

A chave de seleção do sistema de porta paralela encontra-se em **TRISE<PSMODE>**



Resumo dos registradores associados ao PORTD.

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
08h	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
88h	TRISD	Configuração como Entrada (1) ou Saída (0)							
89h	TRISE	IBF	OBF	IBOV	PSPMODE	-	Configuração I/Os		

Não usado para essa finalidade. Para obter mais informações, consulte pêndice A

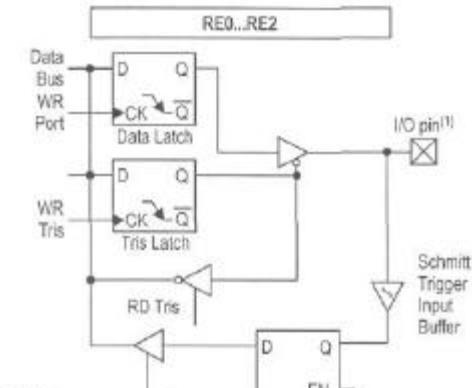
PORTE

Este PORT é o menor de todos, possuindo somente três pinos.

Observe que, de acordo com o diagrama ao lado, todos os pinos do **PORTE** são do tipo ST quando configurados para entrada. Entretanto, para o sistema de porta paralela (**/RD**, **/RW** e **/CS**), estes pinos operam em modo TTL.

A chave de seleção do sistema de porta paralela encontra-se em **TRISE<PSMODE>**.

Apesar do diagrama simplificado não apresentar, estes três pinos também operam como entradas analógicas, ligadas ao conversor interno.



Nota 2: I/Os protegidos por diodos para V_{DD} e V_{SS} .

Resumo dos registradores associados ao PORTE

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
09h	PORTE	IOF	-	-	-	-	RE2	RE1	RE0
89h	TRISE	-	OBF	IBOV	PSPMODE	-	Configuração dos I/Os		
9Fh	ADCON1	AMF	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0

Não usado para essa finalidade. Para obter mais informações, consulte pêndice A

Estudo dos Timers

Vamos aprender agora sobre os recursos dos três timers disponíveis no PIC 16F877A e como utilizá-los.

Cada um desses contadores internos possui características próprias, como o limite de contagem, o tipo de incremento, os pré e postscalers, a geração de interrupções, periféricos **associados**, etc. Vamos, então, explorá-los individualmente:

Timer0

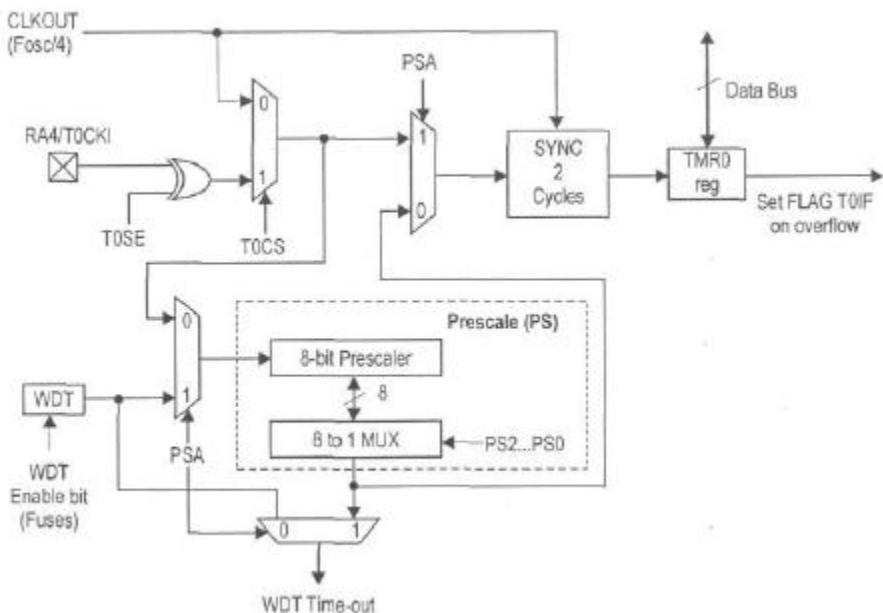
Também conhecido como **TMR0**, é um contador de 8 bits, cujo valor atual encontra-se disponível no registrador **TMR0**. Este registrador pode ser tanto lido quanto escrito, possibilitando a inicialização do contador. Outra característica interessante do **TMR0** é que apesar dele ser somente incremental, seus incrementos podem ocorrer de duas maneiras distintas, através da chave **OPTION_REG<TOCS>**:

- **T0CS = 1**: Incremento a cada transição no pino **RA4 / T0CKI**.
- **T0CS = 0**: Incremento a cada ciclo de máquina.

Quando selecionado o incremento pelo pino **T0CKI**, isto é, um pulso ou clock externo, é possível ainda selecionar se o incremento acontecerá na borda de subida ou na borda de descida do sinal de entrada, através de **OPTION_REG<TOSE>**:

- **T0SE = 1:** Incremento na borda de descida;
- **T0SE = 0:** Incremento na borda de subida.

Entretanto, para incremento externo ou interno, será aplicado um prescaler (PS) antes do registrador **TMR0** ser realmente alterado. Desta forma, caso o prescaler esteja configurado em 1:4, por exemplo, serão necessários quatro ciclos de máquina (ou pulsos externos) para que **TMR0** seja incrementado. O contador interno do PS é de 8 bits, mas não está disponível nem para leitura nem para escrita. Vale lembrar que toda vez que algum dado é escrito em **TMR0**, o contador de PS é zerado.



Para a utilização do prescaler é necessário primeiramente direcioná-lo ao Timer 0 ou ao WDT. Isso é possível por intermédio do **OPTION_REG<PSA>**:

- **PSA = 1:** Prescale aplicado ao WDT;
- **PSA = 0:** Prescale aplicado ao TMR0.

Depois disso, é necessário ainda a configuração do valor do PS, conforme ajustes em **OPTION_REG<PS2:PS0>**:

PS2	PS1	PS0	TMR0 (PSA=0)	WTD (PSA=1)
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Desta forma, a única maneira de incrementarmos o **TMR0** com uma relação de 1:1 é se aplicarmos o PS ao WDT. Isso pode ser verificado também por meio do diagrama de bloco.

Para aumentar o poder de operação do **TMR0**, associado a ele existe uma interrupção acontecendo toda vez que o registrador **TMR0** estourar, isto é, sempre que o limite de 8 bits (FFh ou 255d) for ultrapassado. Quando isso acontecer, o flag **INTCON<T0IF>** será setado. Para utilizar esta interrupção, a chave **INTCON<T0IE>** deve ser ligada, assim como a chave geral **INTCON<GIE>**. Não esquecer de limpar o flag (**T0IF**) manualmente durante o tratamento da interrupção.

Resumo dos registradores associados ao Timer0

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
01h...	TMR0	Contador – 8 bits (após prescaler)							
0Bh...	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
81h...	OPTIN_REG	/RBU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

Não usado para essa finalidade. Para obter mais informações, consulte pêndice A

Timer1

Este timer é bem mais poderoso que o Timer0 por uma série de motivos. Primeiramente, trata-se de um timer de 16 bits, composto de dois registradores de 8 bits (**TMR1H e TMR1 L**) que podem ser escritos e lidos pelo programador. Além disso, como no Timer0, também pode operar com um sinal externo, mas com uma vantagem: existe um circuito interno que possibilita ligar diretamente um cristal aos pinos **RC0/T1OSO** e **RC1/T1OSI**. Outra forma de operar com sinal externo é ligando um sinal pulsado qualquer ao pino **RC0/T1CKI**, de forma similar ao Timer0.

Para escolher entre incremento interno ou externo, a chave **T1CON<TMR1CS>** deve ser configurada:

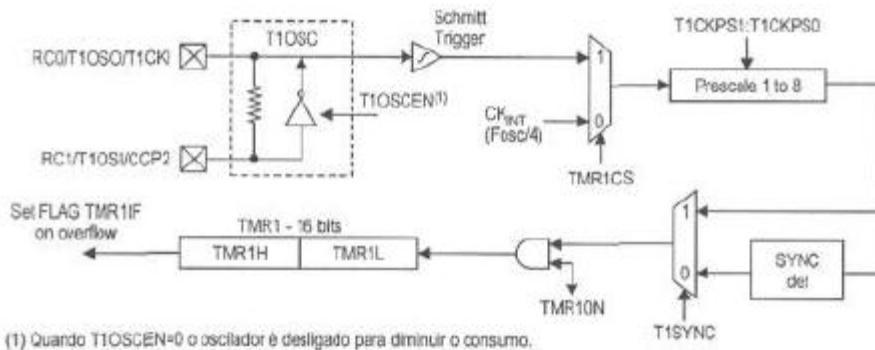
- **TMR1CS = 1:** Incremento externo, através de sinal RC0/TOCKI ou oscilador em **RC0/T1OSO** e **RC1/T1OSI**;
- **TMR1CS = 0:** Incremento interno através dos ciclos de máquina.

A chave **T1CON<T1OSCEN>** é a responsável pela habilitação ou não do circuito de oscilação interna:

- **T1OSCEN = V:** Oscilador ligado. **RC0** e **RC1** são configurados como entrada, independentemente do estado do **TRISC**. Estes pinos são perdidos como I/Os;
- **T1OSCEN = 0:** Circuito oscilador desligado. **RC0** opera como **T1CKI** (sinal externo para contagem) e **RC1** opera como I/O normal.

Esse circuito de oscilação é idêntico ao utilizado pelo próprio cristal do PIC quando na opção LP. Desta forma, ele é de baixo consumo, mas também tem sua freqüência de trabalho limitada em 200 kHz.

Para melhorar o sinal externo utilizado para incrementar o TMR1, o mesmo passa por uma porta *Schmitt Trigger*. Isso pode ser observado no diagrama de blocos deste timer.



Depois de selecionada a origem do sinal, o sistema possui um prescaler próprio para efetivar o - remonto. Esse prescaler deve ser configurado através de **T1CON<T1CKPS1:T1CKPS0>**:

T1CKPS1	T1CKPS0	Prescale
0	0	1:1
1	0	1:2
1:2	1	1:4
0	1	1:8
1.4.1		

De forma análoga ao Timer0, este timer também possui um sistema de sincronismo do clock externo com o clock interno. Entretanto, ao contrário do Timer0, esse sincronismo pode ser desligado através da chave **T1CON</T1SYNC>**:

- **/T1SYNC = 1:** Sincronismo desligado;
- **/T1SYNC = 0:** Sincronismo ligado.

Este recurso é muito interessante, pois se mantivermos o sincronismo desligado podemos continuar operando com o Timer1 mesmo com o PIC em modo **SLEEP**, desde que seu incremento seja externo. Quando o sincronismo está ligado, como no caso do Time0, isso não é possível, pois o sistema de sincronismo para de operar no **SLEEP**, uma vez que o clock interno é desligado.

Quando estamos trabalhando com um sinal externo, o incremento do **TMR1** será efetuado na borda de subida do sinal. Entretanto, após o timer ser ligado, é necessária uma borda de descida antes da primeira borda de subida que será considerada.

Este timer possui ainda uma chave de habilitação, que pode ser utilizada para ligar e desligar os incrementos de **TMR1**. Esta chave localiza-se em **T1CON<TMR1ON>**:

- **TMR1ON = 1:** Incremento de TMR1 habilitado;
- **TMR1ON = 0:** Incremento de TMR1 desabilitado.

Escrever e ler o Timer1 pode ser um tanto complicado, pois como ele possui dois registradores, enquanto estamos trabalhando com um o outro pode ser alterado. A maneira mais convencional seria desligarmos o timer antes de executarmos essas operações, mas nem sempre isso é possível. Outra forma é lermos os valores (primeiro **TMR1H** e depois **TMR1L**) guardando-os em variáveis temporárias. Depois da leitura, comparamos o valor de **TMR1H**. Caso ele tenha aumentado, lemos ambos os valores novamente. Para a escrita, devemos primeiro limpar **TRM1L**, depois carregamos **TMR1H** e **TMR1L**. Assim como no TMRO, a escrita nestes registradores reseta o prescaler.

Associado ao estouro do Timer1 também temos uma interrupção, que deve ser habilitada por **PIE1<TMR1IE>**, além de **INTCON<PEIE:GIE>**. O flag de indicação para essa interrupção é o **PIR1<TMR1IF>**.

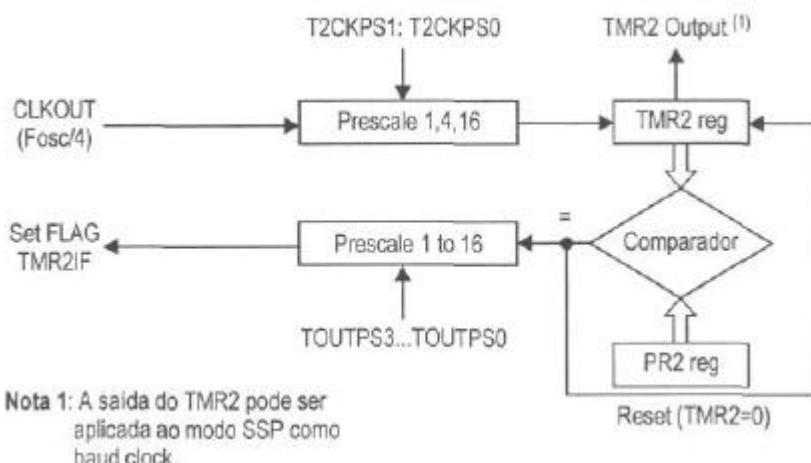
Resumo dos registradores associados ao Timer1

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0Bh...	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
0Ch...	PIR1	PSPIF	ADIF	RCIF	RXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
8Ch...	PIER1	PSPIE	ADIE	RCIE	RXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
0Eh	TMR1L	Contador – 8 bits (parte menos significativa)							
0Fh	TMR1H	Contador – 8 bits (parte menos significativa)							
10h	T1CON	-	-	TCKPS1	TCKPS0	T1OSCEN	/T1SYNC	TMR1CS	TMR1ON

Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Timer2

O Timer2 é um contador de 8 bits, relacionado somente ao clock interno, que possui um prescaler um postscale, sendo ambos configuráveis.



Uma característica que o difere bastante dos demais Timers é que ele não conta de 0 (zero) até o limite imposto pelos 8 bits. Na verdade, quem impõe o limite de incremento é o valor escrito no registrador **PR2**. Desta forma, sempre que **TMR2 = PR2**, o timer é resetado, voltando a zero. Neste mesmo momento, o contador de postscale é incrementado. Quando o postscale terminar, a interrupção associada ao Timer2 será gerada.

O ajuste do prescaler deve ser efetuado através de

T2CON<T2CKPS1:T2CKPS0>:

T2CKPS1	T2CKPS0	Prescale
0	0	1:1
0	1	1:4
1.41 1	0	1:16
1	1	1:16
1.8		

Apesar de parecer um erro, as duas últimas opções são realmente idênticas.

Quanto ao *postscale*, sua função está diretamente ligada à interrupção cie Timer2. Isso porque ele da mais é que um contador de estouros do **TMR2** (não se esqueça de que esses estouros são em comparação com **PR2**, e não no limite de 8 bits), Somente quando a quantidade de estouros for igual ao valor configurado no *postscale* é que a Interrupção irá acontecer.

Uma característica interessante para o *postscale* é que ele pode ser ajustado entre 1 e 16, com incrementos unitários. Em relação aos SFRs, sua configuração deve ser feita em **T2CON<TOUPPS3:TOUPPS0>**:

TOUPPS3	TOUPPS2	TOUPPS1	TOUPPS0	Postscale
0	0	0	0	1:1
0	0	0	1	1:2
0	0	1	0	1:3
0	0	1	1	1:4
0	1	0	0	1:5
0	1	0	1	1:6
0	1	1	0	1:7
0	1	1	1	1:8
1	0	0	0	1:9
1	0	0	1	1:10
1	0	1	0	1:11
1	0	1	1	1:12
1	1	0	0	1:13
1	1	0	1	1:14
1	1	1	0	1:15
1	1	1	1	1:16

Observe que, tanto o *prescaler* quanto o *postscale*, serão zerados sempre que uma das seguintes iterações acontecer:

- Uma operação de escrita envolvendo o registrador **TMR2**;
- Uma operação de escrita envolvendo o registrador **T2CON**;
- Qualquer tipo de reset do PIC.

Entretanto, o TMR2 é zerado somente durante algum tipo de reset, diferentemente dos demais timers, que, nos resets, ou mantêm seu valor ou possuirão um valor desconhecido.

Apesar do diagrama simplificado de blocos não mostrar, o Timer2 possui ainda uma chave de habilitação idêntica à existente no Time1. Esta chave encontra-se em **T2CON<TMR2ON>**:

- **TMR2ON = 1:** Incremento de TMR2 habilitado;
- **TMR2ON = 0:** Incremento de TMR2 desabilitado.

Resumo dos registradores associados ao Timer2

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0Bh...	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
0Ch	PIR1	PSP1IF	ADIF	RCIF	RXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
8Ch	PIE1	PSP1IE	ADIE	RCIE	RXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
11H	TMR2					Contador – 8 bits			
12h	T2CON	-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
92h	PR2					Limite de incremento para TMR2			

— Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Maximização de pino

Este assunto é muito solicitado e necessário nos projetos atuais. Isso porque é de grande valia economizarmos recursos de hardware, seja para podermos utilizar um PIC menor, seja para viabilizarmos o projeto ou ainda para obtermos I/Os sobrando para futuras expansões. O importante é lembrarmos que o investimento em um projeto mais aprimorado é feito uma única vez, enquanto o custo de um PIC maior é inerente a toda a vida do produto.

A maximização mais comum que podemos efetuar é o reaproveitamento de um mesmo I/O para mais de uma função, muitas vezes extremamente diferenciadas. Mas como isso é possível? Uma maneira para realizarmos isso é, por exemplo, através do chaveamento de um MUX externo, fazer com que hardwares diferentes possam ser interligados no mesmo pino do PIC, um de cada vez.

Outra forma de maximização é a varredura, onde trabalhamos com matrizes do tipo linhas versus colunas. Essa técnica é comumente utilizada para teclados com grande número de teclas, Leds ou displays de segmentos. O próximo capítulo será dedicada ao sistema de varredura de displays.

Por fim, outra técnica muito utilizada é a ligação de dois hardwares diferentes no mesmo pino, ao mesmo tempo, sem a separação por MUX. Mas como isso é possível? Obviamente nem sempre isso é real, mas existem diversos casos em que podemos ligar um hardware equivalente a um sinal de entrada junto a outro hardware responsável pelo controle de uma saída.

Está parecendo um pouco confuso não é mesmo? Mas, na verdade, é muito simples. Vamos pegar o modelo mais utilizado para essa aplicação, sobre o qual trabalharemos em nosso exemplo. Trata-se da ligação de um botão e um Led ao mesmo pino do PIC.

Para que isso funcione corretamente, projetamos um hardware capaz de operar com o botão normalmente, informando ao PIC dois níveis de tensão diferenciados (alto e baixo). Conjuntamente, acrescentamos ao esquema um Led que será controlado quando o PIC impor o nível de tensão (alto e baixo). Para que isso funcione, o segredo está na operação conjunta entre esse hardware e o controle efetuado pelo software, pois o pino que está sendo utilizado deve ficar constantemente chaveado como saída (para controlar o Led) e entrada, para ler o valor do botão.

Vamos, então, nos aprofundar um pouco mais nesta teoria. Começamos pelo esquema elétrico do hardware necessário à aplicação.

Do lado esquerdo temos a ligação padrão para um botão, Isto é, um pull-up para garantir o nível alto quando o botão está liberado (aberto) e o botão propriamente dito, ligado ao V_{ss}, que garante o nível baixo quando pressionado (fechado).

Depois temos a ligação de um Led para terra com um resistor para o PIC, limitando a corrente máxima permitida.

Entre o circuito do botão e do Led existe um terceiro resistor para evitarmos um curto durante o momento em que queremos acender o Led e pressionamos o botão.

Vamos agora entender a lógica do sistema.

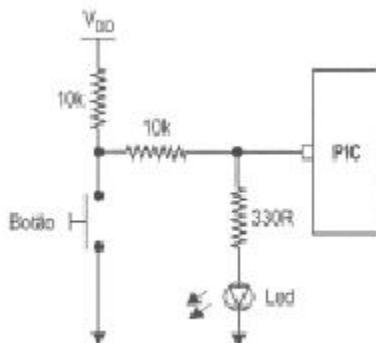
Na maior parte do tempo, o pino do PIC é configurado como saída, já que, se ele estiver impondo 0 (V_{ss}) o Led permanece apagado. Se ele impor 1 (V_{DD}), o Led será se. Devido ao resistor de proteção, o pressionamento ou não do botão não interfere no funcionamento do Led.

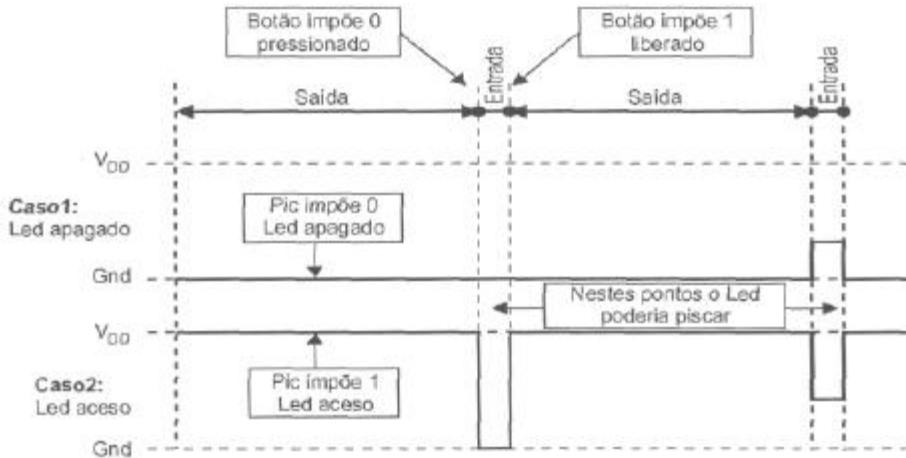
Até aí tudo bem, mas como vamos manter o Led controlado e lermos o botão ao mesmo tempo? A questão é muito mais simples do que parece. Vamos fazer uso da técnica de configurarmos o pino como sarada para lermos o estado do botão. Acontece que faremos isso tão rápido que será imperceptível para o led. Assim sendo, na maior parte do tempo o pino permanece como saída, e periodicamente ele é invertido para entrada, o botão é lido e em seguida, vira saída novamente.

A figura seguinte apresentará claramente o sinal em relação ao pino do PIC. Repare que a relação entre o tempo de entrada e saída é muito mais propícia ao controle do Led (saída). Como isso é feito em uma freqüência relativamente alta para o olho humano (no mínimo 100 Hz), a piscada do Led durante o estado de entrada será imperceptível.

Desta forma, o botão não é lido o tempo inteiro, mas somente uma vez dentro de cada período da freqüência de chaveamento. Por isso, para que possamos implementar bons filtros para o debounce do botão, é necessário uma boa amostragem do estado do botão. Isso nos obriga a trabalharmos com freqüências mais altas (em torno de 1 kHz).

Outro ponto importante que deve ser observado nesse diagrama é o fato de que, durante a leitura do botão, o nível de tensão sobre o pino do PIC não é muito alto. Isso se dá devido a uma certa polarização do Led quando o botão não está pressionado. Por isso, o cálculo dos resistores deve ser feito para garantir um nível mínimo que será interpretado pelo PIC como sendo 1 (nível alto). Isso é ainda mais crítico se estivermos utilizando uma entrada do tipo Schmitt Trigger (ST).





Lógica do exemplo

Chegamos agora ao nosso primeiro exemplo prático. Está na hora de visualizarmos as primeiras linhas de código para esse nosso treinamento.

Comecemos, então, entendendo qual a lógica empregada ao exemplo para que possamos ver na prática os conceitos já estudados na parte teórica deste capítulo.

Nosso exemplo será composto de um programa capaz de ler quatro botões e tocar o buzzer com uma freqüência diferente para cada combinação de botões. Para cada botão existe um Led ligado ao mesmo pino, que será utilizado para indicar os botões pressionados no momento. Utilizaremos os timers e duas interrupções para controlarmos a freqüência de leitura dos botões e a freqüência do buzzer.

- **Timer 0:** controlará a freqüência de varredura dos botões;
- **Timer 2:** controlará a freqüência do som. O som será feito excitando o buzzer com uma onda quadrada de freqüência variável. Variaremos essa freqüência alterando o valor de PR2, que controla o estouro deste timer.

Desta forma, nosso programa principal será um loop infinito sem nenhum efeito. Somente ficaremos esperando o acontecimento das interrupções.

Para a varredura dos botões, ajustaremos a interrupção de **TMR0** para aproximadamente 500 Hz:

Ciclo de Maq.	Prescale	Conta TMR0	Auxiliar	Período	Freqüência
1µs	8	256	-	2048µs	448 Hz

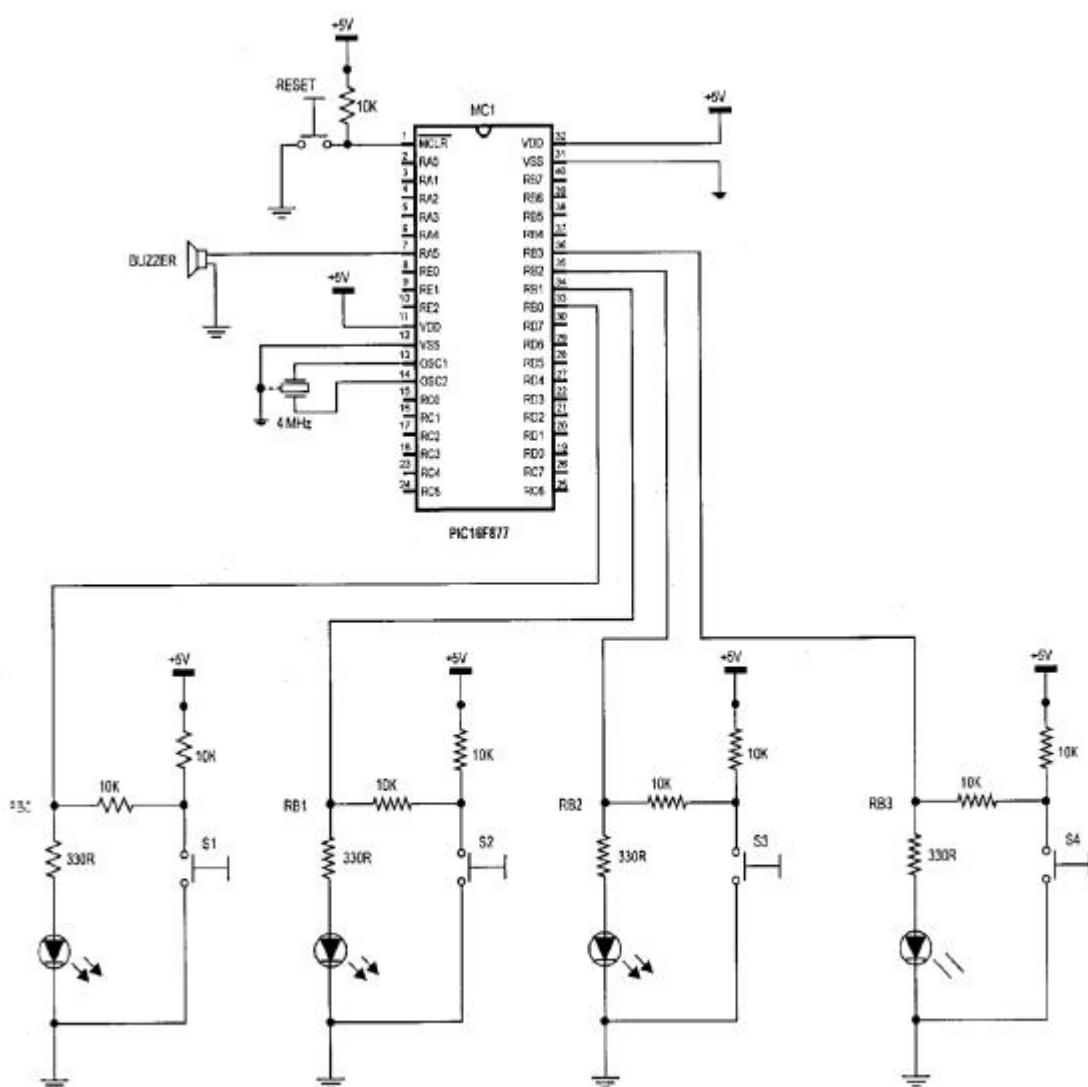
Além disso, o sistema possui um filtro, regulado pela constante **FILTRO_BOTAO**, para evitar o debounce da tecla. Esse filtro garante que a tecla fique pressionada pelo tempo de **FILTRO_BOTAO** x 2048µs.

Quanto a freqüência do buzzer, esta será controlada por **TMR2**. Calibraremos os valores de pré e postscale para que a freqüência da interrupção do **TMR2** varie entre 100 Hz e 2 kHz, com a variação de **PR2** entre 16 e 240:

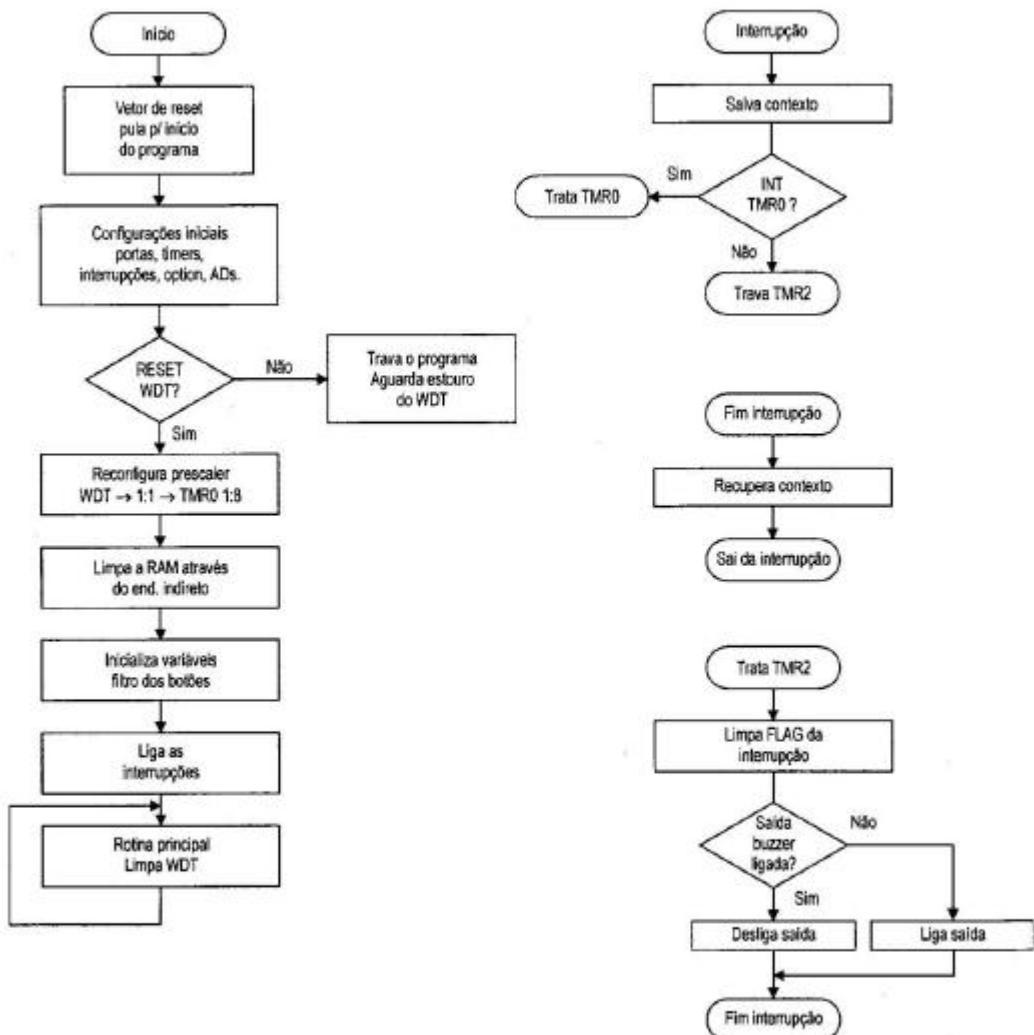
Ciclo de Maq.	Prescale	Postscale	PR2	Período	Freqüência
1µs	16	2	16	512µs	1953 Hz
1µs	16	2	240	7680µs	130Hz

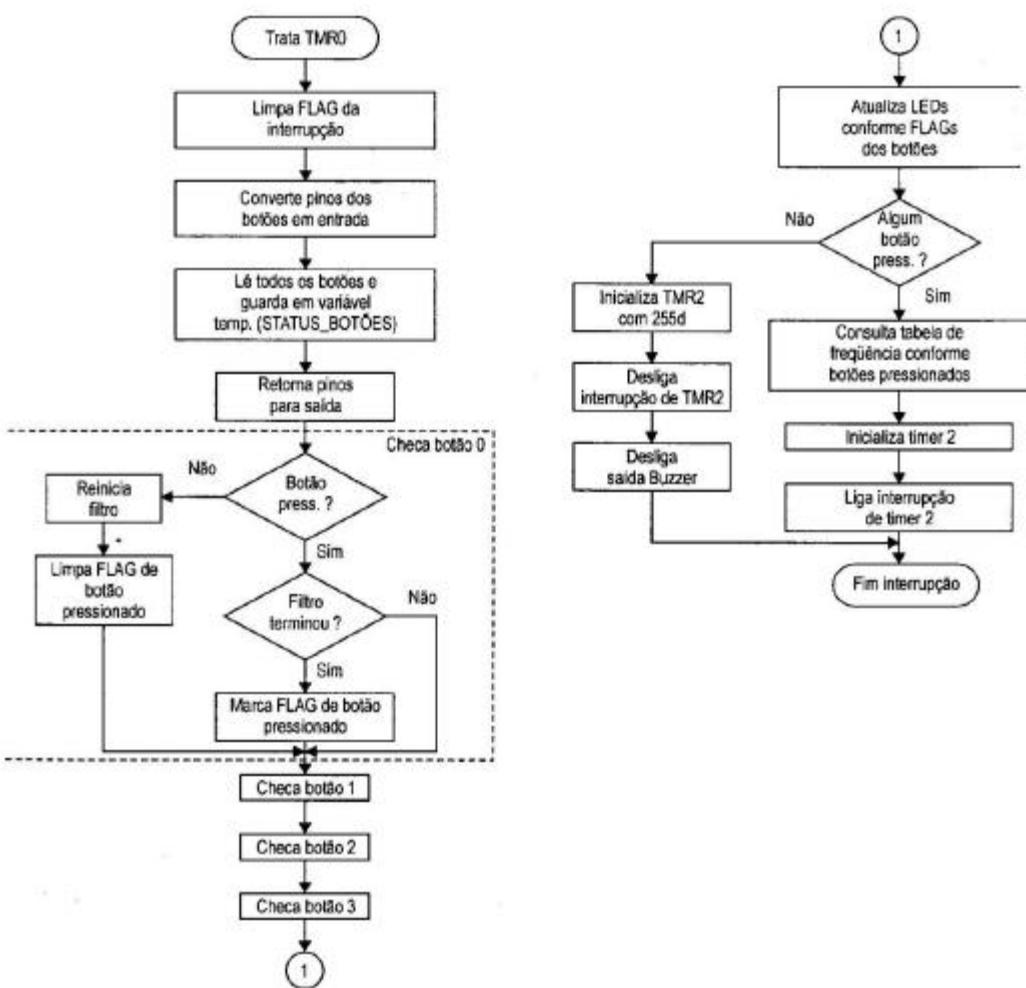
Para facilitar a implementação, a cada interrupção inverteremos o estado do pino de acionamento do buzzer. Desta forma, a freqüência deste será equivalente à metade da freqüência da interrupção do **TMR2**.

Esquema elétrico



Fluxograma





```

;*****
;*          CONECTANDO O PIC - RECURSOS AVANÇADOS      *
;*          EXEMPLO 1                                     *
;*
;*          NICOLÁS CÉSAR LAVINIA e DAVID JOSÉ DE SOUZA   *
;*
;*****  

;*  VERSÃO : 2.0                                         *
;*  DATA : 24/02/2003                                     *
;*****  

;*****  

;*          DESCRIÇÃO GERAL                            *
;*****  

; ESTE SOFTWARE ESTÁ PREPARADO PARA LER QUATRO BOTÕES E TOCAR O BUZZER COM  

; DURAÇÃO VARIÁVEL CONFORME A TECLA PRESSIONADA, ALÉM DE ACENDER O LED  

; INDICANDO A ÚLTIMA TECLA PRESSIONADA.  

;  

;*****  

;*          CONFIGURAÇÕES PARA GRAVAÇÃO                  *
;*****  

;_CONFIG _CP_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &  

;_PWRTE_ON & _WDT_ON & _XT_OSC  

;  

;*****  

;*          DEFINIÇÃO DAS VARIÁVEIS                   *
;*****  

; ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO NO FINAL DO BANCO 0, A PARTIR  

; DO ENDEREÇO 0X70, POIS ESTA LOCALIZAÇÃO É ACESSADA DE QUALQUER BANCO,  

; FACILITANDO A OPERAÇÃO COM AS VARIÁVEIS AQUI LOCALIZADAS.  

;  

        CBLOCK 0X70           ; POSIÇÃO COMUM A TODOS OS BANCOS  

;  

        W_TEMP                ; REGISTRADOR TEMPORÁRIO PARA W  

        STATUS_TEMP            ; REGISTRADOR TEMPORÁRIO PARA STATUS  

        BOTOES_TEMP            ; REGISTRADOR TEMPORÁRIO PARA BOTÕES  

;  

        STATUS_BOTOES          ; ARMAZENA O ESTADO DOS BOTÕES  

        STATUS_LEDS              ; ARMAZENA O ESTADO DOS LEDS  

;  

        FILTRO_BT0               ; FILTRO PARA BOTAO 0  

        FILTRO_BT1               ; FILTRO PARA BOTAO 1  

        FILTRO_BT2               ; FILTRO PARA BOTAO 2  

        FILTRO_BT3               ; FILTRO PARA BOTAO 3  

;  

        ENDC  

;  

;*****  

;*          DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC    *
;*****  

; O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE  

; OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE  

; DE REDIGITAÇÃO.  

;  

        #INCLUDE <P16F877A.INC>           ; MICROCONTROLADOR UTILIZADO  

;  

;*****  

;*          DEFINIÇÃO DOS BANCOS DE RAM                  *
;*****  

; OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR  

; ENTRE OS BANCOS DE MEMÓRIA.  

;  

#define BANK1  BSF      STATUS,RP0      ; SELECCIONA BANK1 DA MEMORIA RAM  

#define BANK0  BCF      STATUS,RP0      ; SELECCIONA BANK0 DA MEMORIA RAM  

;  

;*****  

;*          CONSTANTES INTERNAS                      *
;*****  

; A DEFINIÇÃO DE CONSTANTES FACILITA A PROGRAMAÇÃO E A MANUTENÇÃO.  

;  

        FILTRO_BOTAO  EQU  .20           ; FILTRO P/ EVITAR RUIDOS DOS BOTÕES  

;  

;*****  

;*          DECLARAÇÃO DOS FLAGs DE SOFTWARE          *
;*****  

; A DEFINIÇÃO DE FLAGs AJUDA NA PROGRAMAÇÃO E ECONOMIZA MEMÓRIA RAM.

```

```

; ESTE PROGRAMA NÃO UTILIZA NENHUM FLAG DE USUÁRIO

;*****
; *          ENTRADAS           *
;*****

; AS ENTRADAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#define BOTOES      PORTB      ; ENTRADA DOS BOTÕES (RB0 ATÉ RB3)

#define BOTAO_0      BOTOES_TEMP,0 ; ESTADO DO BOTÃO 0
; 1 -> LIBERADO
; 0 -> PRESSIONADO

#define BOTAO_1      BOTOES_TEMP,1 ; ESTADO DO BOTÃO 1
; 1 -> LIBERADO
; 0 -> PRESSIONADO

#define BOTAO_2      BOTOES_TEMP,2 ; ESTADO DO BOTÃO 2
; 1 -> LIBERADO
; 0 -> PRESSIONADO

#define BOTAO_3      BOTOES_TEMP,3 ; ESTADO DO BOTÃO 3
; 1 -> LIBERADO
; 0 -> PRESSIONADO

;*****
; *          SAÍDAS           *
;***** 

; AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#define LEDS      PORTB      ; SAÍDA PARA LEDS (RB0 ATÉ RB3)

#define LED_BOTAO_0 STATUS_LEDs,0 ; LED CORRESPONDENTE AO BOTÃO 0
; 1 -> LED LIGADO
; 0 -> LED DESLIGADO

#define LED_BOTAO_1 STATUS_LEDs,1 ; LED CORRESPONDENTE AO BOTÃO 1
; 1 -> LED LIGADO
; 0 -> LED DESLIGADO

#define LED_BOTAO_2 STATUS_LEDs,2 ; LED CORRESPONDENTE AO BOTÃO 2
; 1 -> LED LIGADO
; 0 -> LED DESLIGADO

#define LED_BOTAO_3 STATUS_LEDs,3 ; LED CORRESPONDENTE AO BOTÃO 3
; 1 -> LED LIGADO
; 0 -> LED DESLIGADO

#define BUZZER     PORTA,5      ; SAÍDA PARA BUZZER

;*****
; *          VETOR DE RESET DO MICROCONTROLADOR           *
;***** 

; POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA

    org    0x0000      ; ENDEREÇO DO VETOR DE RESET
    goto  CONFIG      ; PULA PARA CONFIG DEVIDO A REGIÃO
; DESTINADA ÀS INTERRUPÇÕES

;*****
; *          VETOR DE INTERRUPÇÃO DO MICROCONTROLADOR        *
;***** 

; POSIÇÃO DE DESVIO DO PROGRAMA QUANDO UMA INTERRUPÇÃO ACONTECE

    org    0x0004      ; ENDEREÇO DO VETOR DE INTERRUPÇÃO

; É MUITO IMPORTANTE QUE OS REGISTRADORES PRIORITÁRIOS AO FUNCIONAMENTO DA
; MÁQUINA, E QUE PODEM SER ALTERADOS TANTO DENTRO QUANTO FORA DAS INTS SEJAM
; SALVOS EM REGISTRADORES TEMPORÁRIOS PARA PODEREM SER POSTERIORMENTE
; RECUPERADOS.

SALVA_CONTEXTO
    movwf W_TEMP       ; COPIA W PARA W_TEMP
    swapf STATUS,W     ; COPIA STATUS PARA STATUS_TEMP
    movwf STATUS_TEMP   ; COPIA STATUS PARA STATUS_TEMP

```

```

;*****
; * TESTA QUAL INTERRUPÇÃO FOI SOLICITADA *
;*****
; TESTA O FLAG DAS INTERRUPÇÕES PARA SABER PARA QUAL ROTINA DESVIAR.

TESTA_INT
    BTFSC  INTCON,T0IF          ; FOI INTERRUPÇÃO DE TMR0 ?
    GOTO   INT_TMR0             ; SIM - PULA P/ INT_TMR0
                                ; NÃO - ENTÃO FOI TMR2

;*****
; * TRATAMENTO DA INTERRUPÇÃO DE TIMER 2 *
;*****
; ROTINA PARA TRATAMENTO DA INTERRUPÇÃO DE TIMER 2.
; INVERTE O ESTADO DO PINO DO BUZZER.

INT_TMR2
    BCF    PIR1,TMR2IF         ; LIMPA FLAG DA INTERRUPÇÃO
    BTFSS  BUZZER              ; BUZZER LIGADO?
    GOTO   LIGA_BUZZER         ; NÃO - ENTÃO LIGA
                                ; SIM
    BCF    BUZZER              ; DESLIGA O BUZZER
    GOTO   SAI_INT              ; SAI DA INTERRUPÇÃO

LIGA_BUZZER
    BSF    BUZZER              ; LIGA O BUZZER
    GOTO   SAI_INT              ; SAI DA INTERRUPÇÃO

;*****
; * TRATAMENTO DA INTERRUPÇÃO DE TIMER 0 *
;*****
; TRATAMENTO DA INTERRUPÇÃO DE TIMER 0. RESPONSÁVEL POR CONVERTER OS PINOS
; DOS BOTÕES EM ENTRADA, SALVAR A SITUAÇÃO DOS MESMOS NUMA VARIÁVEL
; TEMPORÁRIA, CONVERTER NOVAMENTE OS PINOS PARA SAÍDA E ATUALIZAR OS LEDS.

INT_TMR0
    BCF    INTCON,T0IF         ; LIMPA FLAG DA INTERRUPÇÃO
; ***** FOTOGRAFA O ESTADO DOS BOTÕES *****
    BANK1
    MOVLW B'00001111'
    IORWF  TRISB,F            ; SELECCIONA BANCO 1 DA RAM
                                ; PREPARA MASCARA
                                ; EXECUTA MASCARA (RB0...RB3 ENTRADA)
    BANK0
    GOTO   $+1                 ; SELECCIONA BANCO 0 DA RAM
    GOTO   $+1                 ; DELAY(ESTABILIZAÇÃO ENTR.=4 CICLOS)
    MOVF   BOTOES,W            ; CARREGA NO WORK O ESTADO DOS BOTÕES
    XORLW  0xFF                ; INverte todos os bits devido à
                                ; lógica invertida dos botões
    ANDLW  B'00001111'          ; LIMPA a parte alta, que não é usada
    MOVWF  BOTOES_TEMP          ; SALVA EM BOTOES_TEMP
    BANK1
    MOVLW B'11110000'
    ANDWF  TRISB,F            ; SELECCIONA BANCO 1 DA RAM
                                ; PREPARA MASCARA
                                ; EXECUTA MASCARA (RB0...RB3 SAÍDA)
    BANK0
; ***** ATUALIZA STATUS_BOTOES CONFORME BOTÕES PRESSIONADOS *****
TESTA_BT0
    BTFSS  BOTA0_0              ; BOTÃO 0 PRESSIONADO?
    GOTO   BT0_LIB               ; NÃO - TRATA BOTÃO COMO LIBERADO
                                ; SIM
    DECFSZ FILTRO_BT0,F        ; DECREMENTA FILTRO DO BOTÃO. ACABOU?
    GOTO   TESTA_BT1             ; NÃO - TESTA PRÓXIMO BOTÃO
    BSF    STATUS_BOTOES,0       ; SIM - MARCA BOTÃO COMO
PRESSIONADO
    GOTO   TESTA_BT1             ; TESTA PRÓXIMO BOTÃO

BT0_LIB
    MOVLW FILTRO_BOTA0          ; REINICIALIZA FILTRO
    MOVWF  FILTRO_BT0


```

```

BCF      STATUS_BOTOES,0           ; MARCA BOTÃO COMO LIBERADO

TESTA_BT1
    BTFSS  BOTAO_1               ; BOTÃO 1 PRESSIONADO?
    GOTO   BT1_LIB                ; NÃO - TRATA BOTÃO COMO LIBERADO
                                    ; SIM
                                    ; DECREMENTA FILTRO DO BOTÃO. ACABOU?
                                    ; NÃO - TESTA PRÓXIMO BOTÃO
                                    ; SIM - MARCA BOTÃO COMO
PRESSIONADO
    GOTO   TESTA_BT2              ; TESTA PRÓXIMO BOTÃO

BT1_LIB
    MOVLW FILTRO_BOTAO
    MOVWF FILTRO_BT1
    BCF    STATUS_BOTOES,1        ; REINICIALIZA FILTRO
                                    ; MARCA BOTÃO COMO LIBERADO

TESTA_BT2
    BTFSS  BOTAO_2               ; BOTÃO 2 PRESSIONADO?
    GOTO   BT2_LIB                ; NÃO - TRATA BOTÃO COMO LIBERADO
                                    ; SIM
                                    ; DECREMENTA FILTRO DO BOTÃO. ACABOU?
                                    ; NÃO - TESTA PRÓXIMO BOTÃO
                                    ; SIM - MARCA BOTÃO COMO
PRESSIONADO
    GOTO   TESTA_BT3              ; TESTA PRÓXIMO BOTÃO

BT2_LIB
    MOVLW FILTRO_BOTAO
    MOVWF FILTRO_BT2
    BCF    STATUS_BOTOES,2        ; REINICIALIZA FILTRO
                                    ; MARCA BOTÃO COMO LIBERADO

TESTA_BT3
    BTFSS  BOTAO_3               ; BOTÃO 3 PRESSIONADO?
    GOTO   BT3_LIB                ; NÃO - TRATA BOTÃO COMO LIBERADO
                                    ; SIM
                                    ; DECREMENTA FILTRO DO BOTÃO. ACABOU?
                                    ; NÃO - CONTINUA EXECUÇÃO DO PROGRAMA
                                    ; SIM - MARCA BOTÃO COMO
PRESSIONADO
    GOTO   CONTINUA              ; E CONTINUA EXECUÇÃO DO PROGRAMA

BT3_LIB
    MOVLW FILTRO_BOTAO
    MOVWF FILTRO_BT3
    BCF    STATUS_BOTOES,3        ; REINICIALIZA FILTRO
                                    ; MARCA BOTÃO COMO LIBERADO

CONTINUA
    MOVF   STATUS_BOTOES,W       ; ATUALIZA LEDS CONFORME BOTÕES
    MOVWF LEDs

    MOVF   STATUS_BOTOES,F
    BTFSS  STATUS,Z
    GOTO   MUDA_FREQ              ; TODOS OS BOTÕES SOLTOS?
                                    ; NÃO - DEVE ALT. A FREQ.
                                    ; SIM
                                    ; MUDA PARA BANK1

BANK1
    MOVLW .255
    MOVWF PR2
    BCF    PIE1,TMR2IE
BANK0
    BCF    BUZZER
    GOTO   SAI_INT                ; PR2 = 255
                                    ; DESLIGA INT. TIMER2
                                    ; MUDA PARA BANK0
                                    ; GARANTE PINO DO BUZZER EM 0
                                    ; SAI DA INTERRUPÇÃO

MUDA_FREQ
    CALL   ACERTA_FREQ            ; CHAMA TABELA DE FREQ.
    BANK1
    MOVWF PR2
                                    ; MUDA PARA BANK1
                                    ; ACERTA VALOR DE PR2 CONFORME
                                    ; RETORNO DA TABELA
    BSF    PIE1,TMR2IE
    BANK0
                                    ; LIGA INT. TIMER2
                                    ; MUDA PARA BANK0

;*****
; *          FIM DA ROTINA DE INTERRUPÇÃO
; *****
; RESTAURAR OS VALORES DE "W" E "STATUS" ANTES DE RETORNAR.

SAI_INT

```

```

SWAPF STATUS_TEMP,W          ; COPIA STATUS_TEMP PARA STATUS
MOVWF STATUS
SWAPF W_TEMP,F              ; COPIA W_TEMP PARA W
SWAPF W_TEMP,W              ; RETORNA DA INTERRUPÇÃO
RETFIE

;*****
; *      TABELA DE ACERTO DA FREQUÊNCIA DO BUZZER      *
;*****


ACERTA_FREQ
MOVF STATUS_BOTOES,W        ; COLOCA STATUS_BOTOES EM W
ADDWF PCL,F                 ; SOMA STATUS_BOTOES AO PCL
; CRIANDO UMA SELEÇÃO TIPO "CASE"
; CONFORME TABELA ABAIXO:
; -> LIBERADO
; X-> PRESSIONADO
; BT3 BT2 BT1 BT0

RETLW .255
RETLW .16
RETLW .32
RETLW .48
RETLW .64
RETLW .80
RETLW .96
RETLW .112
RETLW .128
RETLW .144
RETLW .160
RETLW .176
RETLW .192
RETLW .208
RETLW .224
RETLW .240

; - - - -
; - - X
; - X -
; - X X X
; - X X -
; X - -
; X - X
; X - X -
; X - -
; X X - -
; X X - X
; X X - X
; X X - -
; X X X -
; X X X X

;*****
; *      CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE      *
;*****


; NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS
; CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA A
; MÁQUINA E AGUARDA O ESTOURO DO WDT.

CONFIG
CLRF PORTA                  ; GARANTE TODAS AS SAÍDAS EM ZERO
CLRF PORTB
CLRF PORTC
CLRF PORTD
CLRF PORTE

BANK1                         ; SELECIONA BANCO 1 DA RAM

MOVLW B'11011111'
MOVWF TRISA                   ; CONFIGURA I/O DO PORTA
MOVLW B'11110000'
MOVWF TRISB                   ; CONFIGURA I/O DO PORTB
MOVLW B'11111111'
MOVWF TRISC                   ; CONFIGURA I/O DO PORTC
MOVLW B'11111111'
MOVWF TRISD                   ; CONFIGURA I/O DO PORTD
MOVLW B'00000011'
MOVWF TRISE                   ; CONFIGURA I/O DO PORTE
MOVLW B'11001111'
MOVWF OPTION_REG               ; CONFIGURA OPTIONS
; PULL-UPS DESABILITADOS
; INTER. NA BORDA DE SUBIDA DO RBO
; TIMER0 INCREM. PELO CICLO DE MÁQUINA
; WDT - 1:128
; TIMER - 1:1

MOVLW B'01100000'
MOVWF INTCON                  ; CONFIGURA INTERRUPÇÕES
; HABILITA AS INT. DE TMRO E PERIF.

```

```

        MOVLW B'00000000'
        MOVWF PIE1           ; CONFIGURA INTERRUPÇÕES
                                ; DESABILITA AS INT. DE TMR1 E TMR2

        MOVLW B'00000111'
        MOVWF ADCON1          ; CONFIGURA CONVERSOR A/D
                                ; CONFIGURA PORTA E PORTE COMO I/O DIGITAL

        BANK0                 ; SELECCIONA BANCO 0 DA RAM

        MOVLW B'00001111'
        MOVWF T2CON            ; TIMER2: PRESCALE - 1:16
                                ; POSTSCALE - 1:2

; AS INSTRUÇÕES A SEGUIR FAZEM COM QUE O PROGRAMA TRAVE QUANDO HOUVER UM
; RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT. DESTA FORMA,
; SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA UM ESTOURO DE WDT
; E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO START-UP DO PIC.

        BTFSC STATUS,NOT_TO    ; RESET POR ESTOURO DE WDT?
        GOTO   $                ; NÃO - AGUARDA ESTOURO DO WDT
                                ; SIM

; RECONFIGURA O VALOR DO OPTION_REG PARA ACERTAR O PRESCALE.

        BANK1                 ; SELECCIONA BANCO 1 DA RAM
        MOVLW B'11000010'
        MOVWF OPTION_REG       ; RECONFIGURA OPTIONS
                                ; PULL-UPS DESABILITADOS
                                ; INTER. NA BORDA DE SUBIDA DO RB0
                                ; TIMER0 INCR. PELO CICLO DE MÁQUINA
                                ; WDT - 1:1
                                ; TIMER0 - 1:8

        BANK0                 ; SELECCIONA BANCO 0 DA RAM

;*****
; *          INICIALIZAÇÃO DA RAM
; ****
; ESTA ROTINA IRÁ LIMPAR TODA A RAM DO BANCO 0, INDO DE 0X20 A 0X7F.
; EM SEGUIDA, AS VARIÁVEIS DE RAM DO PROGRAMA SÃO INICIALIZADAS.

        MOVLW 0X20
        MOVWF FSR               ; APONTA O ENDEREÇAMENTO INDIRETO PARA
                                ; A PRIMEIRA POSIÇÃO DA RAM

LIMPA_RAM
        CLRF  INDF             ; LIMPA A POSIÇÃO
        INCF  FSR,F            ; INCREMENTA PONTEIRO P/ A PRÓX. POS.
        MOVF  FSR,W            ; COMPARA PONTEIRO COM A ÚLT. POS. +1
        XORLW 0X80              ; JÁ LIMPOU TODAS AS POSIÇÕES?
        BTFS  STATUS,Z          ; NÃO - LIMPA A PRÓXIMA POSIÇÃO
        GOTO  LIMPA_RAM         ; SIM

        MOVLW FILTRO_BOTAO     ; INICIALIZA OS FILTROS DOS BOTÕES
        MOVWF FILTRO_BT0
        MOVWF FILTRO_BT1
        MOVWF FILTRO_BT2
        MOVWF FILTRO_BT3

;*****
; *          LOOP PRINCIPAL
; ****
; ESTA ROTINA PRINCIPAL SIMPLESMENTE LIMPA O WDT, POIS TODA A LÓGICA DO
; PROGRAMA É TRATADA DENTRO DAS INTERRUPÇÕES.

        BSF    INTCON,GIE        ; LIGA AS INTERRUPÇÕES

LOOP
        CLRWDT                  ; LIMPA WATCHDOG TIMER
        GOTO  LOOP                ; VOLTA AO LOOP

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

        END                     ; FIM DO PROGRAMA

```

Dicas e comentários

Observe que, pela lógica e funcionalidade do sistema, não seria necessário o tratamento do filtro dos botões, uma vez que o som só é gerado enquanto o botão é pressionado. Entretanto, este sistema já foi implementado desta forma para fornecer uma lógica eficiente de tratamento de debounce. Lembre-se também de que em muitos sistemas onde será necessário filtrar corretamente uma entrada este filtro poderá ter de ser aplicado duas vezes, uma para cada mudança de estado, isto é, quando o botão é escoado e quando é liberado.

Este programa também não precisaria salvar o contexto para a interrupção, mas já deixamos pronto efeito ilustrativo.

Exercícios propostos

Agora que o exemplo já foi estudado e esclarecido, aproveite para gerar novos problemas e soluções, seguindo os exercícios propostos:

1. Faça o sistema trabalhar em outra faixa de freqüência, alterando os valores de ajuste do TMR2.
2. Acrescentar um delay enquanto os pinos (leds/botões) estão configurados como entrada e verificar o efeito sobre os leds.
3. Inibir o *buzzer* através do desligamento do Timer2 no lugar da desabilitação da interrupção.
4. Inverter o estado do pino do *buzzer* por intermédio do XOR.
5. Inverter a escala de freqüências, trocando a mais alta pela mais baixa.

Anotações

Varredura de Display de Quatro Dígitos

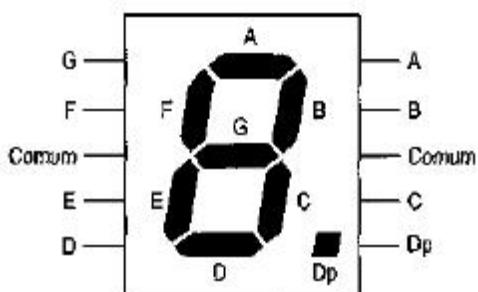
Introdução

Atualmente, um dos pontos mais importantes de qualquer projeto pode ser considerado a interface com o usuário, já que esta interface é composta por entradas e saídas. No capítulo anterior começamos a estudar estas interfaces, mas, como ponto de saída, utilizamos somente os Leds e o buzzer. Para aprimorarmos mais a interface de saída, podemos fazer uso de displays. Neste capítulo estaremos estudando displays de Leds, que apesar de possuírem recursos limitados, ainda são os mais utilizados no mercado devido ao seu baixo custo e fácil visualização.b

Teoria e recursos do PIC

Os displays com os quais trabalharemos são conhecidos como displays de Leds de sete segmentos, pois os números são compostos por sete traços. Estes componentes possuem ainda o ponto décima e se considerados displays numéricos, por não possuírem traços suficientes para a exibição de todas as eras do nosso alfabeto.

Para facilitar a vida do projetista, o mercado padronizou uma nomenclatura para todos os traços do display, possibilitando que tratemos cada um deles individualmente:



Desta forma, temos um pino para controlar cada um dos segmentos (A...G) e mais o ponto (Dp). Os dois pinos adicionais são os comuns, que podem ser ligados a todos os catodos ou anodos dos Leds internos. Por causa disso, este displays são fornecidos em dois tipos: catodo comum ou anodo comum. No nosso caso, os displays utilizados são do tipo catodo comum, isto é, o pino comum deve se ligado ao terra e os segmentos devem ser ligados ao V_{DD} para acenderem.

Outra observação importante é que a pinagem descrita no desenho é válida para o tipo de display utilizado na placa proposta (McLab2). Existem displays de outros tamanhos que possuem uma disposição de pinos diferente.

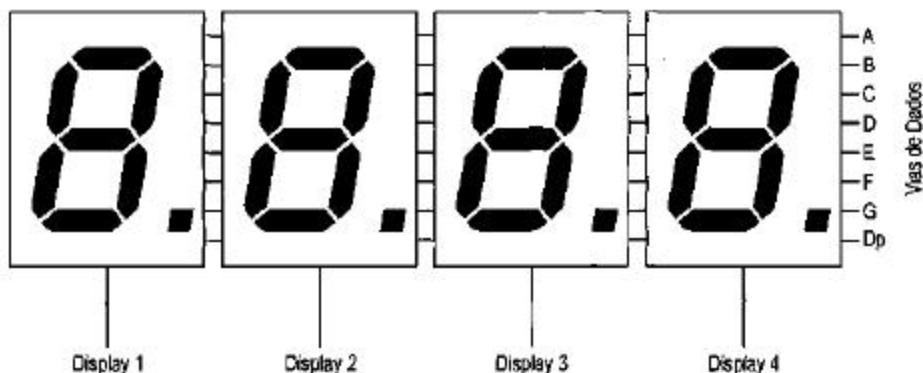
Bem, agora que já conhecemos melhor o componente, vejamos como operá-lo.

Como cada segmento é um Led individual, precisaremos de um pino do PIC para controlar cada um deles. Desta forma, precisaremos de oito pinos para acionar os sete segmentos e mais o ponto decimal. Até aí tudo bem. Acontece que, na nossa placa (e na maioria dos projetos existentes) possuímos não um único display, mas um conjunto deles. Por isso, uma maneira de controlarmos vários displays é disponibilizarmos um conjunto de oito pinos para cada um. Acontece que, na nossa placa, possuímos quatro displays, sendo necessário então 32 pinos no total (8x4). Usaríamos o PIC inteiro somente para controlar os quatro displays. Mas qual a solução, então?

O segredo para minimizarmos a quantidade de pinos utilizados é o emprego de um conceito denominado varredura. Para isso, interligamos todos os pinos de um mesmo segmento (para todos os displays) criando assim oito vias de dados, de A a D_p. Depois, utilizaremos um pino para controlar o comum de cada um dos displays (total de quatro pinos). Assim, quando quisermos escrever em um dos displays, basta informar os segmentos a serem acionados nas vias de dados e ligarmos o comum do display desejado.

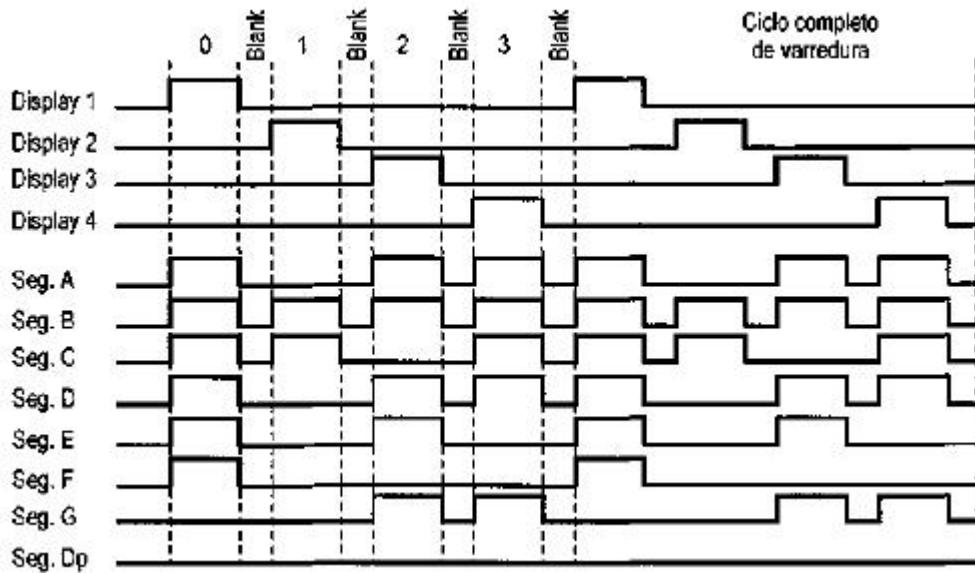
Mas, então, só poderemos escrever em um display de cada vez? Na verdade, sim. Acontece que faremos a escrita na seqüência, isto é, primeiro escrevemos no display 1, depois no 2, no 3 e no 4, voltando novamente para o 1 e repetindo o ciclo infinitamente. Desta forma estaremos "varrendo" todos os displays. Faremos isso numa freqüência tão rápida que nossos olhos não serão capazes de perceber que somente um display está realmente aceso de cada vez.

Assim controlaremos os quatro displays com somente 12 pinos do PIC. Uma melhoria e tanto, não?



Para que não seja possível percebermos a varredura dos displays, sugerimos que a freqüência do ciclo completo (chaveamento de todos os displays) seja de no mínimo 100 Hz. Freqüências bem superiores a esta podem ser utilizadas para facilitar a integração da varredura com as demais funções do sistema. Por outro lado, freqüências mais altas comprometem a velocidade dos demais componentes utilizados, como os transistores ligados aos pinos comuns dos displays. Caso esses transistores não estejam respondendo corretamente à freqüência aplicada, começará a surgir um problema denominado sombreamento. O sombreamento nada mais é que a exibição do valor de um dígito no display seguinte. Além disso, como o tempo de resposta dos Leds também não é muito rápido, o problema do sombreamento pode tornar-se crítico.

Para evitarmos esses sombreamentos, além da velocidade dos transistores, utilizamos também recurso de desligarmos todos os segmentos durante o chaveamento de um display para o outro display com todos os segmentos apagados é chamado de blank.



Outro ponto importante a ser observado na utilização da varredura é que o tempo de permanência se um segmento aceso é o inverso da quantidade de display. No nosso caso, cada display permanece-a aceso aproximadamente 25% (1/4) do tempo. Por isso, a intensidade luminosa do Led durante a varredura é bem menor que se o mesmo fosse controlado individualmente. Para compensar isto, a corrente aplicada ao Led é normalmente maior. No caso da varredura ser feita com muitos displays talvez seja necessária a utilização de displays de alto brilho.

Lógica do exemplo

O exemplo desenvolvido para este capítulo faz muito mais que simplesmente implementa- a varredura dos displays. Trata-se de um contador regressivo de segundos, ou seja, um temporizador capaz de contar até 9.999 segundos.

Para isso, utilizaremos os displays para indicar o valor atual do temporizador. A primeira tecla (S1) não possui nenhuma função. Por outro lado, o Led relacionado a ela (L1) será utilizado para indicar: estado do temporizador:

L1	Descrição
Aceso	Temporizador em contagem regressiva
Apagado	Temporizador paralisado

Os demais botões apresentam as funções de controle do temporizador:

Botão	Descrição
S2	Incrementa o valor inicial em 1 segundo
S3	Decrementa o valor inicial em 1 segundo
S4	Inicia e paralisa o temporizador

Os botões de incremento e decremento operam de forma rotativa, isto é, comutam automaticamente entre 0000 e 9999. Outra característica desses botões é que eles executam suas funções repetidamente quando mantidos pressionados e só funcionam quando o temporizador está paralisado. Ao atingir o valor zero (0000), o temporizador é automaticamente paralisado, desligando-se o Led indicativo (L1).

Para o sistema de varredura foram criadas quatro variáveis para armazenamento dos dígitos mostrados nos respectivos displays: UNIDADE, DEZENA, CENTENA e MILHAR. Essas variáveis representam o valor atual do temporizador e são incrementadas e decrementadas através dos botões. Na verdade, os botões alteram diretamente o valor da unidade. A lógica do sistema compara esse valor com os limites (0 e 9) para alterar ou não os demais dígitos.

A freqüência de varredura é controlada pela interrupção de Timer0. Ajustamos seus parâmetros para que a comutação entre displays (tempo da interrupção) seja de aproximadamente 4 kHz:

Ciclo de Maq.	Prescale	Conta TMRQ	Auxiliar	Período	Freqüência
1µs	1	256	-	256µs	3900 Hz

A freqüência de varredura será a freqüência de comutação dividida pelo número de displays, que no nosso caso será de aproximadamente 1 kHz. Dentro do tratamento da interrupção de TMRO são desligados todos os segmentos (blank), depois é comutado para o próximo display (desligado o atual e ligado o seguinte), dando um delay (cerca de 11µs no total) para resposta do transistor e só depois é que será carregado o valor do dígito correspondente ao display atualmente ativo.

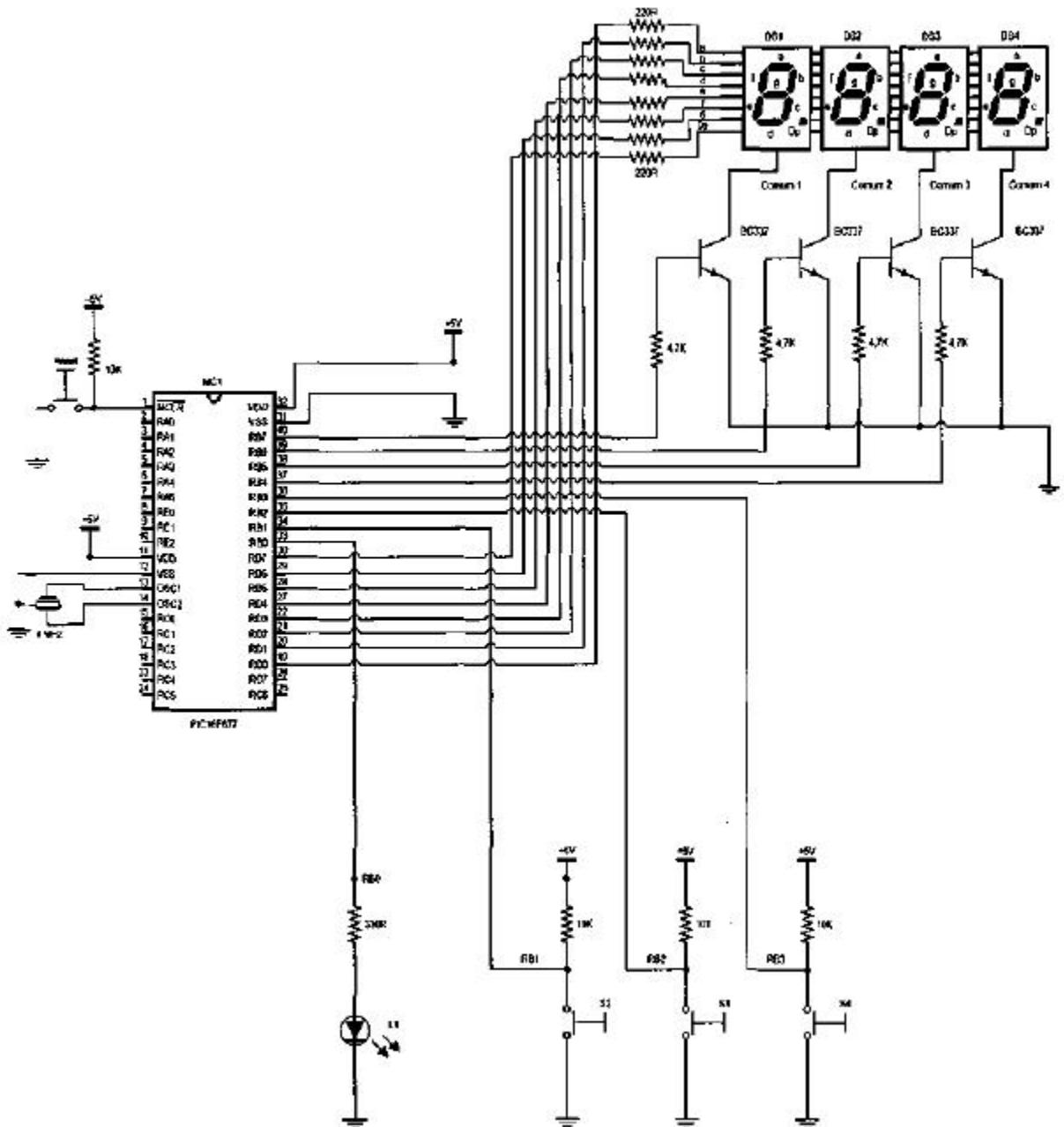
Quanto a contagem dos segundos, utilizamos a interrupção de Timer1 para essa finalidade. Veja os ajustes dos parâmetros para essa interrupção:

Ciclo de Maq.	Prescale	Conta TMR1	Auxiliar	Período	Freqüência
1µs	8	62500	2	1.000.000µs	1 Hz

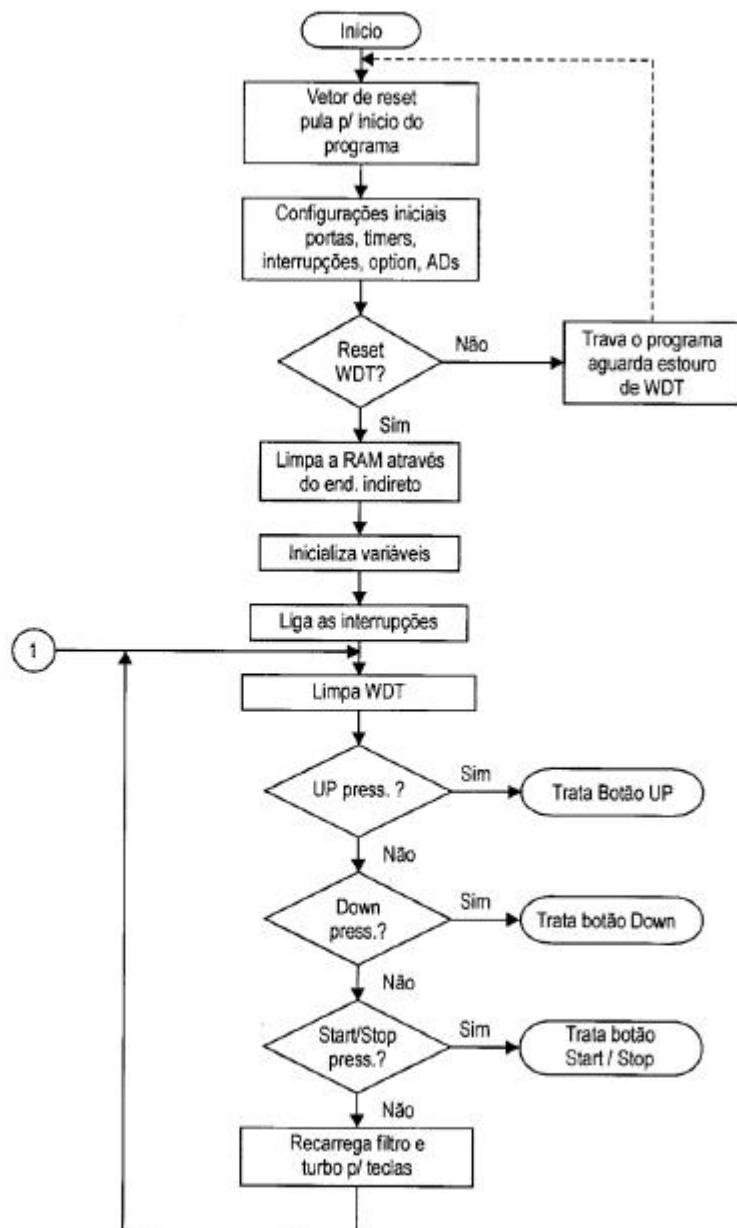
Para isso, configuramos o *prescaler* de **TMR1** em 1:8 e iniciamos o contador com o valor total menos o desejado para a contagem (65.536 - 62.500). Como este valor é de 16 bits, a constante de inicialização foi dividida em TMR1_HIGH e TMR1_LOW. Desta maneira, a interrupção acontecerá a cada 0,5 segundo. Para podermos contar um segundo foi criada uma variável auxiliar denominada DIVISOR_TMR1.

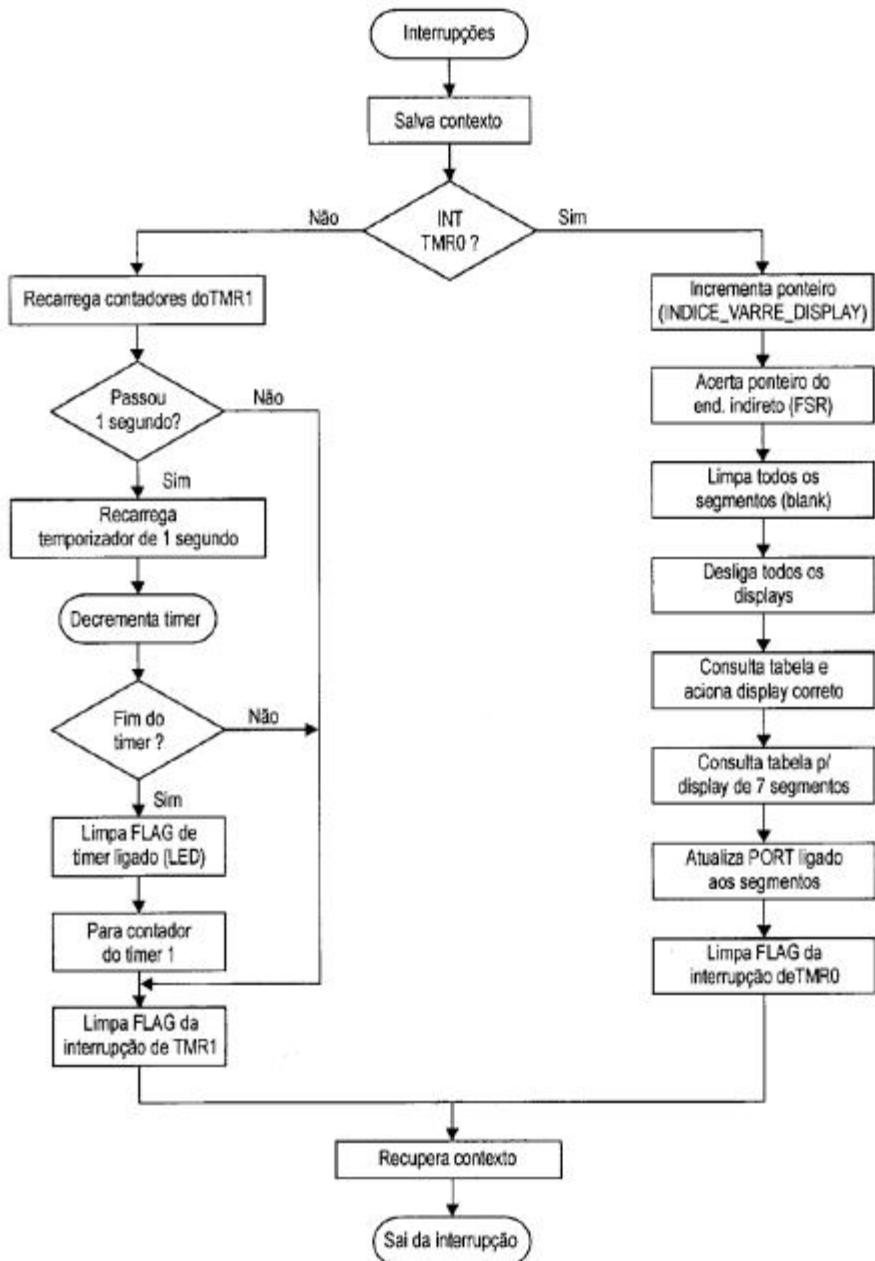
Cada vez que o sistema entrar na interrupção de TMR1 e o contador auxiliar (DIVISOR_TMR1) terminar, o tempo é decrementado, começando pela unidade e chegando até a milhar, se for necessário. Quando o tempo termina (0000), tanto o Led quanto o TMR1 são desligados.

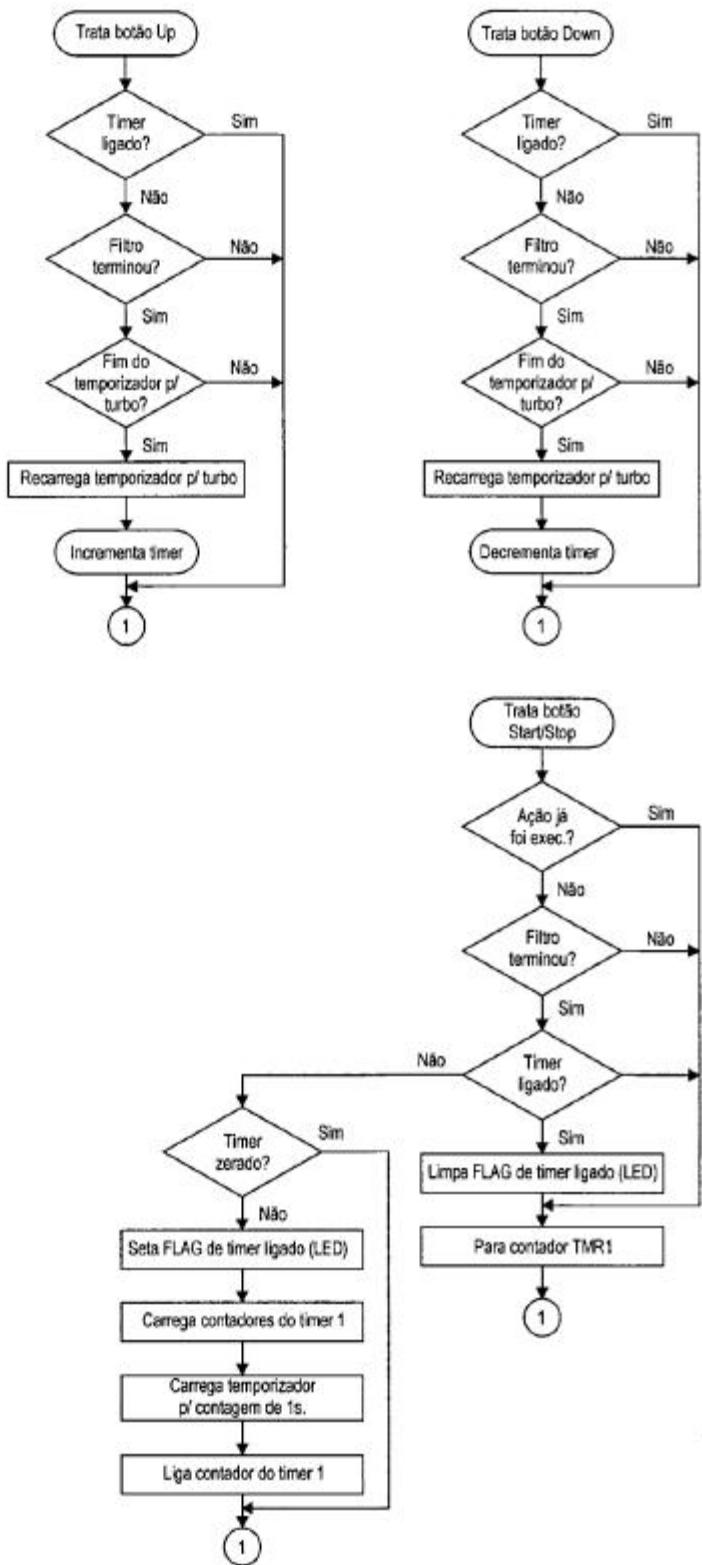
Esquema elétrico

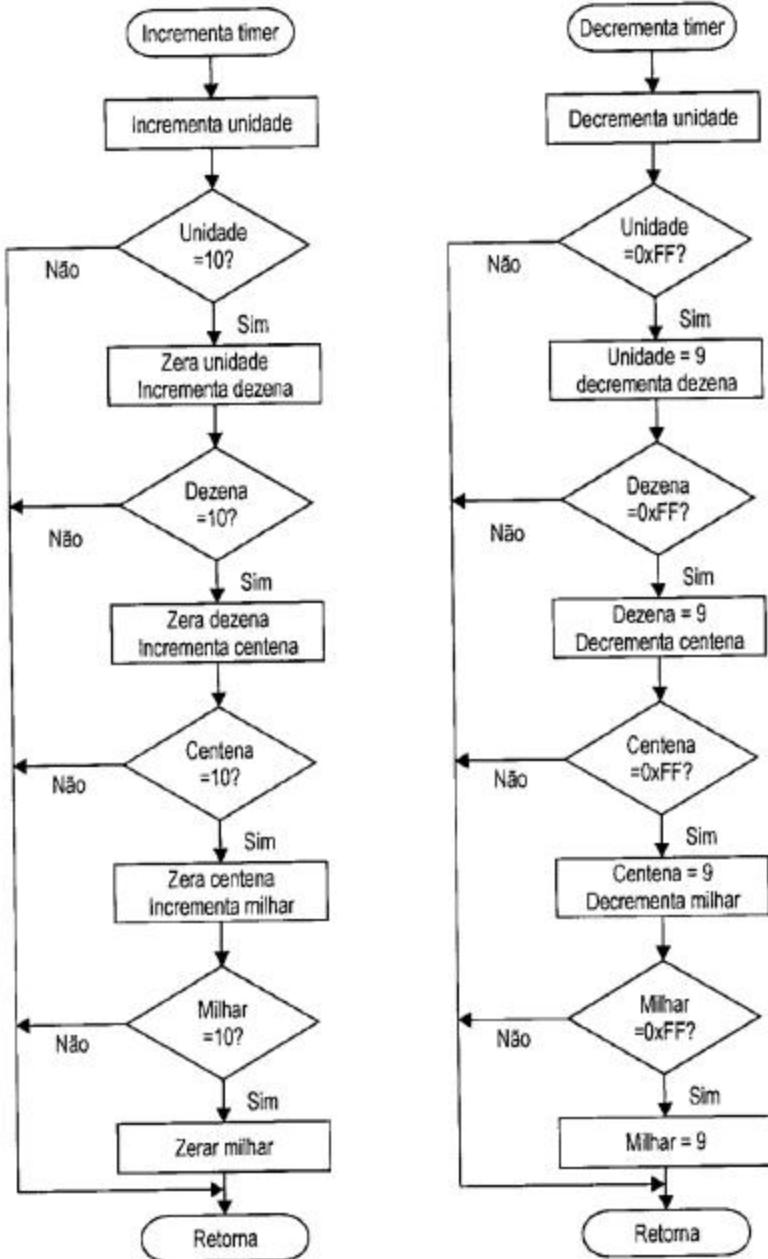


Fluxograma









```

;*****
;*      CONECTANDO O PIC - RECURSOS AVANÇADOS      *
;*      EXEMPLO 2          *
;*      *
;*      NICOLÁS CÉSAR LAVINIA e DAVID JOSÉ DE SOUZA      *
;*      *
;*      *
;*****;
; * VERSÃO : 2.0          *
; * DATA : 24/02/2003      *
;*****;

;*****
;*      DESCRIÇÃO GERAL      *
;*****;

; ESTE EXEMPLO FOI PREPARADO PARA DEMONSTRAR O FUNCIONAMENTO DO TIMER DE
; 16 BITS DO PIC (TMR1) E DA VARREDURA DE DISPLAYS.

; CONSISTE NUM TEMPORIZADOR DE SEGUNDOS. DOIS BOTÕES FORAM UTILIZADOS PARA
; PROGRAMAR O TEMPO DA CONTAGEM. UM OUTRO BOTÃO FOI UTILIZADO PARA DISPARAR
; O CONTADOR. O TEMPORIZADOR CONSEGUE CONTAR ATÉ 9999 SEGUNDOS, DE FORMA QUE
; OS 4 DISPLAYS DE 7 SEGMENTOS FORAM NECESSÁRIOS. A CONTAGEM É REGRESSIVA.
; UM LED INDICA QUE O TEMPORIZADOR ESTÁ OPERANDO. QUANDO O SISTEMA CHEGA
; A 0000 (ZERO) O LED É DESLIGADO AUTOMATICAMENTE.

;

;*****
;*      CONFIGURAÇÕES PARA GRAVAÇÃO      *
;*****;

__CONFIG _CP_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &
_PWRTE_ON & _WDT_ON & _XT_OSC

;*****
;*      DEFINIÇÃO DAS VARIÁVEIS      *
;*****;

; O PRIMEIRO BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO LOGO NO INÍCIO DO BANCO 0

```

CBLOCK 0X20 ; POSIÇÃO INICIAL DA RAM

UNIDADE ; (LSD)

DEZENA ;

CENTENA ;

MILHAR ; (MSD)

FILTRO_BOTOES ; FILTRO PARA RUIDOS

TEMPO_TURBO ; TEMPORIZADOR P/ TURBO DAS TECLAS

INDICE_VARRE_DISPLAY ; INDEXADOR P/ VARREDURA DOS DISPLAYS

DIVISOR_TMR1 ; CONTADOR AUXILIAR P/ SEGUNDOS

ENDC

; O SEGUNDO BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO NO FINAL DO BANCO 0, A PARTIR

; DO ENDEREÇO 0X70, POIS ESTA LOCALIZAÇÃO É ACESSADA DE QUALQUER BANCO,

; FACILITANDO A OPERAÇÃO COM AS VARIÁVEIS AQUI LOCALIZADAS.

CBLOCK 0X70 ; REGIÃO COMUM A TODOS OS BANCOS

STATUS_TEMP ; REGISTRADOR DE STATUS TEMPORÁRIO

WORK_TEMP ; REGISTRADOR DE TRABALHO TEMPORÁRIO

FSR_TEMP ; REG. DE ENDERECO INDIRETO TEMPORÁRIO

PCLATH_TEMP ; REGISTRADOR DE PAGINAÇÃO TEMPORÁRIO

ENDC

;* DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC *

; O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE

; OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE

; DE REDIGITAÇÃO.

```
#INCLUDE <P16F877A.INC> ; ARQUIVO DE DEFINIÇÕES DO PIC ATUAL
```

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

```
/* DEFINIÇÃO DOS BANCOS DE RAM */
```

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

```
; OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR  
; ENTRE OS BANCOS DE MEMÓRIA.
```

```
#DEFINE BANK1 BSF STATUS,RPO ; SELECCIONA BANK1 DA MEMORIA RAM
```

```
#DEFINE BANK0 BCF STATUS,RPO ; SELECCIONA BANK0 DA MEMORIA RAM
```

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

```
/* CONSTANTES INTERNAS */
```

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

```
; A DEFINIÇÃO DE CONSTANTES FACILITA A PROGRAMAÇÃO E A MANUTENÇÃO.
```

```
FILTRO_TECLA EQU .200 ; FILTRO P/ EVITAR RUIDOS DOS BOTÕES
```

```
TURBO_TECLA EQU .70 ; TEMPORIZADOR P/ TURBO DAS TECLAS
```

```
TMR1_HIGH EQU HIGH (.65536-.62500)
```

```
TMR1_LOW EQU LOW (.65536-.62500) ; VALOR PARA CONTAGEM DE  
; 62500 CICLOS DE CONTAGEM  
; DO TMR1 (PROGRAMADO P/  
; PRESCALER DE 1:8)
```

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

```
/* DECLARAÇÃO DOS FLAGs DE SOFTWARE */
```

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

```
; A DEFINIÇÃO DE FLAGs AJUDA NA PROGRAMAÇÃO E ECONOMIZA MEMÓRIA RAM.
```

```
; ESTE PROGRAMA NÃO UTILIZA NENHUM FLAG DE USUÁRIO
```

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

```
/* ENTRADAS */
```

```
Conectando o PIC 16F877A - Recursos Avançados
```

```
;*****  
; AS ENTRADAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E  
; FUTURAS ALTERAÇÕES DO HARDWARE.
```

```
#DEFINE BT_UP PORTB,1 ; ESTADO DO BOTÃO 1  
;  
; 1 -> LIBERADO  
; 0 -> PRESSIONADO
```

```
#DEFINE BT_DOWN PORTB,2 ; ESTADO DO BOTÃO 2  
;  
; 1 -> LIBERADO  
; 0 -> PRESSIONADO
```

```
#DEFINE BT_START_STOP PORTB,3 ; ESTADO DO BOTÃO 3  
;  
; 1 -> LIBERADO  
; 0 -> PRESSIONADO
```

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

```
;* SAÍDAS *
```

```
;*****  
; AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E  
; FUTURAS ALTERAÇÕES DO HARDWARE.
```

```
#DEFINE ESTADO_TIMER PORTB,0 ; LED DE ESTADO DO TIMER  
;  
; (FUNCIONA TAMBÉM COMO FLAG)  
;  
; 1 -> TIMER CONTANDO  
;  
; 0 -> TIMER PARADO
```

```
#DEFINE MUX PORTB ; MUX PARA ACIONAMENTO DOS DISPLAYS  
;  
; (DE RB4 ATÉ RB7)
```

```
#DEFINE SEGMENTOS PORTD ; SEGMENTOS DOS DISPLAYS
```

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

```
;* VETOR DE RESET DO MICROCONTROLADOR *
```

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

; POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA

```
ORG    0X0000          ; ENDEREÇO DO VETOR DE RESET
GOTO   CONFIG          ; PULA PARA CONFIG DEVIDO A REGIÃO
                      ; DESTINADA ÀS INTERRUPÇÕES
```

;*****

/* VETOR DE INTERRUPÇÃO */

;*****

; POSIÇÃO DE DESVIO DO PROGRAMA QUANDO UMA INTERRUPÇÃO ACONTECE

```
ORG    0X0004          ; ENDEREÇO DO VETOR DE INTERRUPÇÃO
```

; É MUITO IMPORTANTE QUE OS REGISTRADORES PRIORITÁRIOS AO FUNCIONAMENTO DA
; MÁQUINA, E QUE PODEM SER ALTERADOS TANTO DENTRO QUANTO FORA DAS INTS SEJAM
; SALVOS EM REGISTRADORES TEMPORÁRIOS PARA PODEREM SER POSTERIORMENTE
; RECUPERADOS.

SALVA_CONTEXTO

```
MOVWF WORK_TEMP        ; SALVA REGISTRADOR DE TRABALHO E
SWAPF STATUS,W         ; DE STATUS DURANTE O TRATAMENTO
MOVWF STATUS_TEMP       ; DA INTERRUPÇÃO.
MOVF   FSR,W
MOVWF FSR_TEMP          ; SALVA REGISTRADOR FSR
MOVF   PCLATH,W
MOVWF PCLATH_TEMP        ; SALVA REGISTRADOR PCLATH

CLRF   PCLATH           ; LIMPA REGISTRADOR PCLATH
                      ; (SELECCIONA PÁGINA 0)
CLRF   STATUS            ; LIMPA REGISTRADOR STATUS
                      ; (SELECCIONA BANCO 0)
```

;*****

/* TESTA QUAL INTERRUPÇÃO FOI SOLICITADA */

;*****

; TESTA O FLAG DAS INTERRUPÇÕES PARA SABER PARA QUAL ROTINA DESVIAR.

```
BTFS  INTCOM,T0IF          ; FOI INTERRUPÇÃO DE TMR0 ?  
GOTO  INT_TMR1           ; NÃO - ENTÃO PULA P/ INT_TMR1  
                           ; SIM
```

```
;  
*          TRATAMENTO DA INTERRUPÇÃO DE TIMER 0      *  
;  
; ROTINA PARA EXECUTAR AS AÇÕES NECESSÁRIAS SEMPRE QUE A INTERRUPÇÃO  
; ACONTECE. NESTE CASO, A INTERRUPÇÃO ESTA SENDO UTILIZADA PARA GERAR A  
; FREQUÊNCIA DE VARREDURA DOS DISPLAYS. POR ISSO, CADA VEZ QUE ELA ACONTECER,  
; O PRÓXIMO DISPLAY SERÁ ACIONADO.
```

INT_TMR0

```
INCF   INDICE_VARRE_DISPLAY,F ; INCR. O ÍNDICE DE VAR. DOS DISPLAYS
```

```
MOVLW B'00000011'
```

```
ANDWF  INDICE_VARRE_DISPLAY,F ; LIMITA CONTAGEM DE 0 A 3
```

```
MOVF   INDICE_VARRE_DISPLAY,W      ; CARREGA NO WORK O VALOR DO ÍNDICE  
ADDLW  UNIDADE                 ; SOMA ENDEREÇO DO PRIMEIRO DÍGITO  
MOVWF  FSR                      ; SALVA RESULTADO NO FSR, APONTANDO  
                           ; PARA O ENDEREÇO DO DÍGITO ATUAL.  
                           ; (ENDEREÇAMENTO INDIRETO)
```

```
CLRF   SEGMENTOS            ; LIMPA OS SEGMENTOS (BLANK)  
                           ; UTILIZADO P/ EVITAR SOMBRAIS NOS  
                           ; DISPLAYS
```

```
MOVLW B'00001111'          ; PREPARA MÁSCARA  
ANDWF  MUX,F                ; EXECUTA MÁSCARA (DESLIGA OS DISP.)
```

```
MOVF   INDICE_VARRE_DISPLAY,W    ; SALVA NO WORK O VALOR DO ÍNDICE  
CALL   TABELA_MUX             ; CONSULTA TABELA MUX  
                           Conectando o PIC 16F877A - Recursos Avançados
```

```

IORWF MUX,F           ; ATUALIZA MUX, SELECIONANDO O
                        ; DISPLAYS CORRETO PARA O MOMENTO

GOTO $+1               ; DELAY DE 2US
                        ; (TEMPO DE RESPOSTA DO TRANSISTOR)

MOVF INDF,W           ; RECUPERA NO WORK O VALOR DO DÍGITO
CALL TABELA_DISPLAY_7_SEG ; CONSULTA TABELA P/ DISPLAYS
MOVWF SEGMENTOS        ; ATUALIZA OS SEGMENTOS, ESCREVENDO
                        ; O VALOR DO DÍGITO CORRETO (PORTD)

```

SAI_INT_TMR0

```

BCF INTCON,T0IF         ; LIMPA FLAG DA INTERRUPÇÃO DE TMR0
GOTO SAI_INT            ; PULA P/ SAI_INT

```

;*****

/* TRATAMENTO DA INTERRUPÇÃO DE TIMER 1 */

;*****

```

; ROTINA PARA EXECUTAR AS AÇÕES NECESSÁRIAS SEMPRE QUE A INTERRUPÇÃO
; ACONTECE. NESTE CASO, A INTERRUPÇÃO ESTA SENDO UTILIZADA PARA CONTAR O
; TEMPO DO TEMPORIZADOR. POR ISSO, CADA VEZ QUE ELA ACONTECER O VALOR DO
; TIMER SERÁ DECREMENTADO, CASO JÁ TENHA SE PASSADO 1SEG.
; PERÍODO DA INTERRUPÇÃO: 1US (CICLO DE MAQUINA) * 8 (PRESCALER DO TMR1) *
;                                     * 62500 (CONTAGEM DO TMR1) = 0,5SEG.

```

INT_TMR1

```

MOVLW TMR1_HIGH
MOVWF TMR1H
MOVLW TMR1_LOW
MOVWF TMR1L           ; RECARREGA CONTADOR DO TMR1
                        ; PERIODICIDADE DE 0,5SEG.

```

```

DECFSZ DIVISOR_TMR1,F   ; PASSOU-SE 1 SEGUNDO ?
GOTO SAI_INT_TMR1        ; NÃO - ENTÃO SAI DA INTERRUPÇÃO
                        ; SIM

```

```

MOVlw .2

MOVwf DIVISOR_TMR1      ; RECARREGA CONTADOR DE 1SEG.

CALL    DECREMENTA_TIMER ; DECREMENTA O VALOR DO TIMER

MOVF   UNIDADE,F

BTFS S STATUS,Z

GOTO  SAI_INT_TMR1

MOVF   DEZENA,F

BTFS S STATUS,Z

GOTO  SAI_INT_TMR1

MOVF   CENTENA,F

BTFS S STATUS,Z

GOTO  SAI_INT_TMR1

MOVF   MILHAR,F

BTFS S STATUS,Z          ; FINAL DA CONTAGEM ? (TIMER=0?)

GOTO  SAI_INT_TMR1       ; NÃO - SAI DA INTERRUPÇÃO

                           ; SIM

BCF   ESTADO_TIMER       ; DESLIGA LED DE TIMER OPERANDO

BCF   T1CON,TMR1ON       ; PARALIZA CONTADOR DO TMR1

```

SAI_INT_TMR1

```
BCF   PIR1,TMR1IF        ; LIMPA FLAG DA INTERRUPÇÃO DE TMR1
```

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

```
;*          SAÍDA DA INTERRUPÇÃO          *
```

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

```
; ANTES DE SAIR DA INTERRUPÇÃO, O CONTEXTO SALVO NO INÍCIO DEVE SER
```

```
; RECUPERADO PARA QUE O PROGRAMA NÃO SOFRÁ ALTERAÇÕES INDESEJADAS.
```

SAI_INT

```

MOVf   PCLATH_TEMP,W

MOVwf PCLATH              ; RECUPERA REG. PCLATH (PAGINAÇÃO)

MOVf   FSR_TEMP,W

MOVwf FSR                  ; RECUPERA REG. FSR (END. INDIRETO)

```

Conectando o PIC 16F877A - Recursos Avançados

```

SWAPF STATUS_TEMP,W
MOVWF STATUS           ; RECUPERA REG. STATUS
SWAPF WORK_TEMP,F
SWAPF WORK_TEMP,W      ; RECUPERA REG. WORK
RETFIE                 ; RETORNA DA INTERRUPÇÃO (HABILITA GIE)

```

;*****

;**TABELA PARA OS DISPLAYS DE 7 SEGMENTOS**

*

;*****

; ROTINA PARA CONVERSÃO DO VALOR NÚMÉRICO DO DÍGITO EM RELAÇÃO AOS SEGMENTOS

; QUE DEVEM SER ACESOS E APAGADOS NO DISPLAY

TABELA_DISPLAY_7_SEG

```

ANDLW B'00001111'          ; EXECUTA MASCARA P/ EVITAR PULOS ERRADOS
ADDWF PCL,F                ; SOMA DESLOCAMENTO AO PROGRAM COUNTER,
                            ; GERANDO UMA TABELA DO TIPO "CASE".
;
PGFEDCBA                  ; POSIÇÃO RELATIVA AOS SEGMENTOS
RETLW B'00111111'          ; 0H - 0
RETLW B'00000110'          ; 1H - 1
RETLW B'01011011'          ; 2H - 2
RETLW B'01001111'          ; 3H - 3
RETLW B'01100110'          ; 4H - 4
RETLW B'01101101'          ; 5H - 5
RETLW B'01111101'          ; 6H - 6
RETLW B'00000111'          ; 7H - 7
RETLW B'01111111'          ; 8H - 8
RETLW B'01101111'          ; 9H - 9
RETLW B'00000000'          ; AH - BLANK
RETLW B'00000000'          ; BH - BLANK
RETLW B'00000000'          ; CH - BLANK
RETLW B'00000000'          ; DH - BLANK
RETLW B'00000000'          ; EH - BLANK
RETLW B'00000000'          ; FH - BLANK

```

;*****

;
* TABELA PARA ACIONAMENTO DOS DISPLAYS *

;
; ROTINA PARA CONVERTER O DÍGITO ATUAL EM RELAÇÃO AO PORT QUE DEVE SER
; LIGADO PARA ACIONAMENTO DO DISPLAY RELACIONADO.

TABELA_MUX

ADDWF PCL,F	; SOMA DESLOCAMENTO AO PROGRAM COUNTER ; GERANDO UMA TABELA DO TIPO "CASE".
RETLW B'00010000'	; 0 - ACIONA DISPLAY 0
RETLW B'00100000'	; 1 - ACIONA DISPLAY 1
RETLW B'01000000'	; 2 - ACIONA DISPLAY 2
RETLW B'10000000'	; 2 - ACIONA DISPLAY 3

;
* ROTINA PARA INCREMENTAR O VALOR DO TIMER (BCD) *

;
; ROTINA UTILIZADA PARA INCREMENTAR O VALOR DOS REGISTRADORES UNIDADE,
; DEZENA, CENTENA E MILHAR, QUE SÃO OS CONTADORES DO TIMER. A CONTAGEM É
; FEITA DIRETAMENTE EM BCD.

INCREMENTA_TIMER

INCF UNIDADE,F	; INCREMENTA UNIDADE
MOVLW .10	
XORWF UNIDADE,W	
BTFSS STATUS,Z	; UNIDADE = 10 ?
RETURN	; NÃO - RETORNA
	; SIM
CLRF UNIDADE	; ZERA A UNIDADE
INCF DEZENA,F	; INCREMENTA A DEZENA

MOVLW .10	
XORWF DEZENA,W	
BTFSS STATUS,Z	; DEZENA = 10 ?
RETURN	; NÃO - RETORNA

; SIM
CLRF DEZENA ; ZERA A DEZENA
INCF CENTENA,F ; INCREMENTA A CENTENA

MOVLW .10
XORWF CENTENA,W
BTFS S STATUS,Z ; CENTENA = 10 ?
RETURN ; NÃO - RETORNA
; SIM
CLRF CENTENA ; ZERA A CENTENA
INCF MILHAR,F ; INCREMENTA O MILHAR

MOVLW .10
XORWF MILHAR,W
BTFS C STATUS,Z ; MILHAR = 10 ?
CLRF MILHAR ; SIM - ZERA MILHAR
RETURN ; NÃO - RETORNA

;* ROTINA PARA DECREMENTAR O VALOR DO TIMER (BCD) *

; ROTINA UTILIZADA PARA DECREMENTAR O VALOR DOS REGISTRADORES UNIDADE,
; DEZENA, CENTENA E MILHAR, QUE SÃO OS CONTADORES DO TIMER. A CONTAGEM É
; FEITA DIRETAMENTE EM BCD.

DECREMENTA_TIMER
DECF UNIDADE,F ; INCREMENTA UNIDADE

MOVLW 0xFF
XORWF UNIDADE,W
BTFS S STATUS,Z ; UNIDADE = 0xFF ?
RETURN ; NÃO - RETORNA
; SIM
MOVLW .9
MOVWF UNIDADE ; CARREGA UNIDADE COM 9
Conectando o PIC 16F877A - Recursos Avançados

```

DECF    DEZENA,F           ; DECREMENTA A DEZENA

MOVLW 0xFF
XORWF DEZENA,W
BTFS S STATUS,Z           ; DEZENA = 0xFF ?
RETURN ; NÃO - RETORNA
; SIM

MOVLW .9
MOVWF DEZENA              ; CARREGA A DEZENA COM 9
DECF    CENTENA,F          ; DECREMENTA A CENTENA

MOVLW 0xFF
XORWF CENTENA,W
BTFS S STATUS,Z           ; CENTENA = 0xFF ?
RETURN ; NÃO - RETORNA
; SIM

MOVLW .9
MOVWF CENTENA              ; CARREGA CENTENA COM 9
DECF    MILHAR,F            ; DECREMENTA O MILHAR

MOVLW 0xFF
XORWF MILHAR,W
BTFS S STATUS,Z           ; MILHAR = 0xFF ?
RETURN ; NÃO - RETORNA
; SIM

MOVLW .9
MOVWF MILHAR              ; CARREGA O MILHAR COM 9
RETURN ; RETORNA

```

```

;*****
;*      CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE      *
;*****  

; NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS  

; CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA A  

; MÁQUINA E AGUARDA O ESTOURO DO WDT.  

Conectando o PIC 16F877A - Recursos Avançados

```

CONFIG

```
CLRF    PORTA           ; GARANTE TODAS AS SAÍDAS EM ZERO
CLRF    PORTB
CLRF    PORTC
CLRF    PORTD
CLRF    PORTE

BANK1          ; SELECCIONA BANCO 1 DA RAM

MOVLW  B'11111111'
MOVWF  TRISA      ; CONFIGURA I/O DO PORTA

MOVLW  B'00001110'
MOVWF  TRISB      ; CONFIGURA I/O DO PORTB

MOVLW  B'11111111'
MOVWF  TRISC      ; CONFIGURA I/O DO PORTC

MOVLW  B'00000000'
MOVWF  TRISD      ; CONFIGURA I/O DO PORTD

MOVLW  B'00000111'
MOVWF  TRISE      ; CONFIGURA I/O DO PORTE

MOVLW  B'11011111'
MOVWF  OPTION_REG   ; CONFIGURA OPTIONS
                     ; PULL-UPS DESABILITADOS
                     ; INTER. NA BORDA DE SUBIDA DO RB0
                     ; TIMER0 INCREM. PELO CICLO DE MÁQUINA
                     ; WDT - 1:128
                     ; TIMER0- 1:1

MOVLW  B'01100000'
MOVWF  INTCON       ; CONFIGURA INTERRUPÇÕES
                     ; Conectando o PIC 16F877A - Recursos Avançados
```

; HABILITADA A INTERRUPÇÃO DE TIMER0
; HABILITA AS INTERRUPÇÕES DE PERIFÉRICO

MOVLW B'00000001'
MOVWF PIE1 ; CONFIGURA INTERRUPÇÕES DE PERIFIÉRICOS
; HABILITADA A INTERRUPÇÃO DE TMR1

MOVLW B'00000111'
MOVWF ADCON1 ; CONFIGURA CONVERSOR A/D
; CONFIGURA PORTA E PORTE COMO I/O DIGITAL

BANK0 ; SELECCIONA BANCO 0 DA RAM

MOVLW B'00110000'
MOVWF T1CON ; CONFIGURA TMR1
; PRESCALER -> 1:8
; INCREMENTADO PELO CICLO DE MÁQUINA

; AS INSTRUÇÕES A SEGUIR FAZEM COM QUE O PROGRAMA TRAVE QUANDO HOUVER UM
; RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT. DESTA FORMA,
; SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA UM ESTOURO DE WDT
; E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO START-UP DO PIC.

BTFSR STATUS,NOT_TO ; RESET POR ESTOURO DE WATCHDOG TIMER?
GOTO \$; NÃO - AGUARDA ESTOURO DO WDT
; SIM

;

* INICIALIZAÇÃO DA RAM *

;

; ESTA ROTINA IRÁ LIMPAR TODA A RAM DO BANCO 0, INDO DE 0X20 A 0X7F.
; EM SEGUIDA, AS VARIÁVEIS DE RAM DO PROGRAMA SÃO INICIALIZADAS.

MOVLW 0X20
MOVWF FSR ; APONTA O ENDEREÇAMENTO INDIRETO PARA
; A PRIMEIRA POSIÇÃO DA RAM
Conectando o PIC 16F877A - Recursos Avançados

LIMPA_RAM

```
CLRF    INDF           ; LIMPA A POSIÇÃO
INCF    FSR,F          ; INCREMENTA O PONTEIRO P/ A PRÓX. POS.
MOVF    FSR,W
XORLW  0X80           ; COMPARA O PONTEIRO COM A ÚLT. POS. +1
BTFSS  STATUS,Z        ; JÁ LIMPOU TODAS AS POSIÇÕES?
GOTO   LIMPA_RAM       ; NÃO - LIMPA A PRÓXIMA POSIÇÃO
                           ; SIM
```

```
BCF    ESTADO_TIMER    ; INICIA COM ESTADO EM OFF
```

```
MOVLW .2
```

```
MOVWF  DIVISOR_TMR1    ; CARREGA CONTADOR DE 1SEG.
```

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

```
;*      VARREDURA DOS BOTÕES      *
```

```
;*      LOOP PRINCIPAL      *
```

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

```
; A ROTINA PRINCIPAL FICA CHECANDO O ESTADO DOS BOTÕES. CASO ALGUM SEJA
; PRESSIONADO, A ROTINA DE TRATAMENTO DO BOTÃO É CHAMADA.
```

```
BSF    INTCON,GIE      ; HABILITA AS INTERRUPÇÕES
                           ; USADA INT. TMR0 PARA VARREDURA
                           ; DOS DISPLAYS
```

VARRE

```
CLRWDT            ; LIMPA WATCHDOG TIMER
```

```
BTFSS  BT_UP          ; O BOTÃO DE UP ESTÁ PRESSIONADO?
```

```
GOTO   TRATA_BT_UP    ; SIM - PULA P/ TRATA_BT_UP
                           ; NÃO
```

```
BTFSS  BT_DOWN         ; O BOTÃO DE DOWN ESTÁ PRESSIONADO?
```

```
GOTO   TRATA_BT_DOWN   ; SIM - PULA P/ TRATA_BT_DOWN
                           ; NÃO
```

BTFSS BT_START_STOP ; O BOTÃO START/STOP ESTÁ PRESSIONADO?
GOTO TRATA_BT_START_STOP ; SIM - PULA P/ TRATA_BT_START_STOP
; NÃO

MOVLW FILTRO_TECLA ; CARREGA NO WORK O VALOR DE FILTRO_TECLA
MOVWF FILTRO_BOTOES ; SALVA EM FILTRO_BOTOES
; RECARREGA FILTRO P/ EVITAR RUIDOS

MOVLW .1

MOVWF TEMPO_TURBO ; CARREGA TEMPO DO TURBO DAS TECLAS
; COM 1 - IGNORA O TURBO A PRIMEIRA
; VEZ QUE A TECLA É PRESSIONADA

GOTO VARRE ; VOLTA PARA VARRER TECLADO

;*****
;* TRATAMENTO DOS BOTÕES *
;*****

; ***** TRATAMENTO DO BOTÃO DE UP *****

TRATA_BT_UP

BTFSR ESTADO_TIMER ; TIMER ESTÁ PARADO ?
GOTO VARRE ; NÃO - VOLTA P/ VARRE
; SIM

DECFSZ FILTRO_BOTOES,F ; FIM DO FILTRO ? (RUIDO?)
GOTO VARRE ; NÃO - VOLTA P/ VARRE
; SIM - BOTÃO PRESSIONADO

DECFSZ TEMPO_TURBO,F ; FIM DO TEMPO DE TURBO ?
GOTO VARRE ; NÃO - VOLTA P/ VARRE
; SIM

MOVLW TURBO_TECLA

```

MOVWF TEMPO_TURBO           ; RECARREGA TEMPORIZADOR DO TURBO
                                ; DAS TECLAS

CALL    INCREMENTA_TIMER     ; INCREMENTA O VALOR DO TIMER

GOTO    VARRE                ; VOLTA P/ VARREDURA DOS BOTÕES

; ***** TRATAMENTO DO BOTÃO DE DOWN *****
; *****

TRATA_BT_DOWN

BTFSC  ESTADO_TIMER         ; TIMER ESTÁ PARADO ?
GOTO    VARRE                ; NÃO - VOLTA P/ VARRE
                                ; SIM

DECFSZ FILTRO_BOTOES,F      ; FIM DO FILTRO ? (RUIDO?)

GOTO    VARRE                ; NÃO - VOLTA P/ VARRE
                                ; SIM - BOTÃO PRESSIONADO

DECFSZ TEMPO_TURBO,F        ; FIM DO TEMPO DE TURBO ?

GOTO    VARRE                ; NÃO - VOLTA P/ VARRE
                                ; SIM

MOVLW TURBO_TECLA

MOVWF TEMPO_TURBO           ; RECARREGA TEMPORIZADOR DO TURBO
                                ; DAS TECLAS

CALL    DECREMENTA_TIMER     ; DECREMENTA O VALOR DO TIMER

GOTO    VARRE                ; VOLTA P/ VARREDURA DOS BOTÕES

; ***** TRATAMENTO DO BOTÃO START / STOP *****
; *****

TRATA_BT_START_STOP

MOVF    FILTRO_BOTOES,F
BTFSC  STATUS,Z              ; FILTRO JÁ IGUAL A ZERO ?
                                ; (FUNÇÃO JÁ FOI EXECUTADA?)

Conectando o PIC 16F877A - Recursos Avançados

```

GOTO VARRE ; SIM - VOLTA P/ VARREDURA DO TECLADO
; NÃO

DECFSZ FILTRO_BOTOES,F ; FIM DO FILTRO ? (RUIDO?)
GOTO VARRE ; NÃO - VOLTA P/ VARRE
; SIM - BOTÃO PRESSIONADO

BTFSZ ESTADO_TIMER ; TIMER ESTA LIGADO ?
GOTO LIGA_TIMER ; NÃO - ENTÃO LIGA
; SIM - ENTÃO DESLIGA

DESLIGA_TIMER

BCF ESTADO_TIMER ; DESLIGA LED E FLAG DO ESTADO DO TIMER
BCF T1CON,TMR1ON ; PARA CONTADOR DO TMR1
GOTO VARRE ; VOLTA P/ VARREDURA DOS BOTÕES

LIGA_TIMER

MOVF UNIDADE,F
BTFSZ STATUS,Z ; UNIDADE ESTÁ ZERADA ?
GOTO LIGA_TIMER_2 ; NÃO - PULA P/ LIGA_TIMER_2
; SIM - TESTA DEZENA
MOVF DEZENA,F
BTFSZ STATUS,Z ; DEZENA ESTÁ ZERADA ?
GOTO LIGA_TIMER_2 ; NÃO - PULA P/ LIGA_TIMER_2
; SIM - TESTA CENTENA
MOVF CENTENA,F
BTFSZ STATUS,Z ; CENTENA ESTÁ ZERADA ?
GOTO LIGA_TIMER_2 ; NÃO - PULA P/ LIGA_TIMER_2
; SIM - TESTA MILHAR
MOVF MILHAR,F
BTFSZ STATUS,Z ; MILHAR ESTÁ ZERADO ?
GOTO LIGA_TIMER_2 ; NÃO - PULA P/ LIGA_TIMER_2
GOTO VARRE ; SIM - VOLTA P/ VARRER TECLADO
; SEM LIGAR O TIMER

```
LIGA_TIMER_2

    BSF      ESTADO_TIMER          ; LIGA LED E FLAG DO ESTADO DO TIMER

    MOVLW TMR1_HIGH
    MOVWF TMR1H
    MOVLW TMR1_LOW
    MOVWF TMR1L          ; INICIALIZA CONTADORES

    MOVLW .2
    MOVWF DIVISOR_TMR1  ; INICIALIZA DIVISOR

    BSF      T1CON,TMR1ON        ; LIGA CONTAGEM DO TMR1

    GOTO    VARRE          ; VOLTA P/ VARREDURA DOS BOTÕES

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

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

END          ; FIM DO PROGRAMA
```

Dicas e comentários

Observe que neste exemplo, ao entrarmos no tratamento das interrupções, a operação de salvar contexto é maior que no exemplo anterior. Isto por que agora salvamos também os valores de FSR e PCLATH pois os mesmos podem ser alterados dentro da interrupção.

Como a varredura dos displays não precisa ter uma freqüência 100% ajustada, ela poderia ser feita sem o emprego de interrupção, isto é, através do loop principal, como foi o caso da checagem dos botões. O único cuidado a ser tomado neste caso é que determinadas ações podem atrapalhar o tempo deste loop, interferindo na freqüência de varredura. Por isso, é muito comum encontrarmos certos projetos onde o display cintila quando pressionamos uma tecla.

Exercícios propostos

Agora que o exemplo já foi estudado e esclarecido, aproveite para gerar novos problemas e soluções, seguindo os exercícios propostos;

1. Implemente o quarto botão, para resetar o temporizador (voltar a zero). Este botão também só deve funcionar quando o temporizador estiver parado;
2. Implemente uma segunda velocidade para os botões de incremento e decremento, de forma que facilite o ajuste de valores maiores;
3. Em vez de fazer um timer somente de segundos, utilize os dois dígitos da esquerda para mostrar o tempo em minutos e os da direita para mostrar o tempo em segundos. O ponto do display DS2 pode ser usado para marcar a separação. Não se esqueça que agora os displays da direita devem contar somente de 0 a 59 e não mais de 0 a 99.

Operação com Display de Cristal Líquido (LCD)

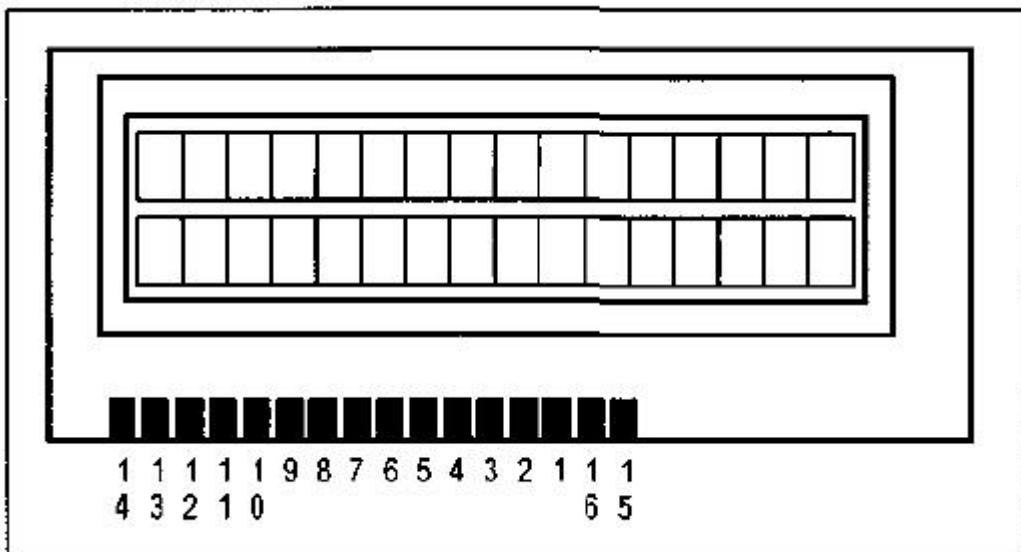
Introdução

Dando continuidade às melhorias no sistema de interface com o usuário, veremos agora a operação com display de cristal líquido. Esses displays são muito mais poderosos que os displays de segmentos, pois possuem muito mais caracteres e são alfanuméricos. Entretanto, são também mais caros e com uma visualização inferior, pois os caracteres não possuem iluminação própria e são de tamanho bem reduzido.

Teoria e recursos do PIC

Existem muitos displays de cristal líquido (LCD) no mercado, mas trabalharemos com o modelo mais padrão de todos. Trata-se de um display de duas linhas com 16 caracteres cada uma. Esse display é bem conhecido por ser o modelo utilizado nos telefones públicos (orelhões). Outra característica importante do display que estaremos estudando é que ele já possui um drive de controle interno. Desta forma, nos comunicaremos com ele através de uma comunicação paralela, passando comandos e os caracteres que desejamos escrever, diretamente em código ASCII.

Vejamos agora o formato do LCD que estaremos estudando:



Neste LCD possuímos 16 pinos para ligação do mesmo ao nosso projeto. A tabela seguinte identifica cada um desses pinos:

Pino	Função	Pino	Função
1	V _{ss}	9	DB2
2	V _{DD}	10	DB3
3	V _O	11	DB4
4	RS	12	DB5
5	R/W	13	DB6
6	E	14	DB7
7	DB0	15	A
8	DB1	16	K

Os dois primeiros pinos (V_{ss} e V_{DD}) são relativos à alimentação do componente e devem ser ligados a uma tensão nominal de 5VDC. As tensões mínima e máxima para a alimentação podem variar conforme o fabricante, mas na maioria dos casos ficam entre 4,75 e 5,25 V_{DC}.

O pino **V_O** é utilizado para controle do contraste e normalmente o ligamos ao centro de um potenciômetro de 10k com as extremidades ligadas ao V_{ss} e ao V_{DD}. Na verdade, esse pino deve possuir uma tensão variável ou fixa entre V_{ss} e V_{DD}.

O pino **RS** (Register Select) é utilizado para definirmos o tipo de informação passada através da comunicação paralela:

RS	Descrição
0	A informação é um comando ou instrução.
1	A informação é um dado.

O pino **R/W** muda o estado do LCD entre Leitura (Read) e Escrita (Write). Essa mudança pode ser feita para escrevermos um comando ou dado e checarmos quando o LCD terminou a operação e está pronto para darmos continuidade ao processo.

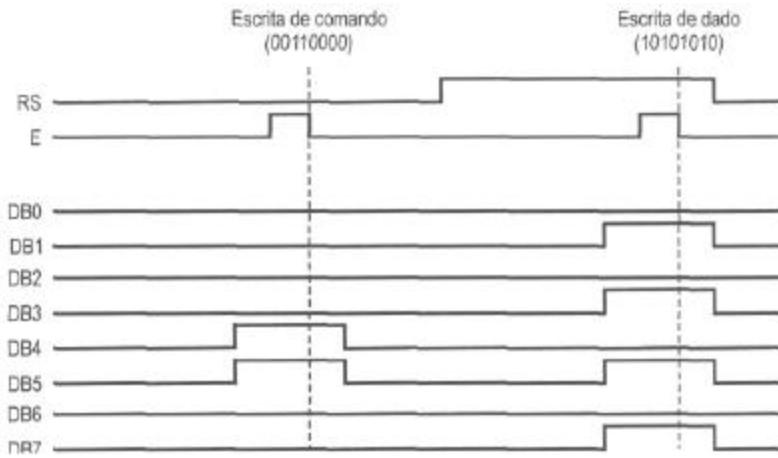
R/W	Descrição
0	Operação de escrita.
1	Operação de leitura.

Em muitos projetos, como no caso da placa proposta (McLab2), esse pino não é utilizado, ficando permanentemente ligado ao V_{ss}. Neste caso, o LCD só opera em modo de escrita. Por isso, precisamos garantir o término das operações internas do módulo de LCD através de tempos pré-estabelecidos. Esses tempos serão descritos adiante.

O pino **E** (Enable) é utilizado para efetivar a leitura da informação escrita no barramento de dados. Essa leitura é efetuada na borda de descida deste sinal.

Os pinos de **DB0** a **DB7** equivalem ao barramento de dados paralelo. Apesar de existirem oito vias de dados, esses displays também podem operar com quatro vias (**DB4** a **DB7**), já que as demais vias ficam sem funções. Neste caso, as informações são enviadas em dois pacotes de 4 bits cada um. Os pinos **A** (Anode) e **K** (Katode) são usados para ligação do *Backligth* (iluminação de fundo). O fato é que, apesar da existência dos pinos, nem todos os displays possuem essa iluminação.

Para nós, então, valerá a comunicação em oito vias de dados, acrescida dos controles **RS** e **E**, dando um total de dez pinos interligando o PIC ao LCD. Desta forma, para envirmos uma informação ao LCD precisaremos primeiramente ajustar **RS** para informarmos se é um comando ou um dado. Em seguida, devemos escrever a informação no barramento de dados. O próximo passo é darmos um pulso em **E**.



Inicialização do LCD

A primeira ação a ser efetuada no display é sua inicialização, garantindo a comunicação em 8 vias. Para isso, devemos criar uma rotina baseada no seguinte roteiro:

1. Aguarde pelo menos 15ms após a energização do LCD para garantir que ele já está operando corretamente.
2. Envie o comando 0x30 para o display.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	1	0	0	0	0

3. Aguarde pelo menos 4ms para garantir o término da operação.
4. Envie novamente o comando 0x30 para o display.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	1	0	0	0	0

5. Aguarde pelo menos 100µs.
6. Envie novamente o comando 0x30 para o display.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	1	0	0	0	0

7. Aguarde pelo menos 40µs.

8. Estabeleça as condições de utilização. Neste caso, comunicação em oito vias, display de duas linhas e matriz de 7x5.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	1	1	0	0	0

9. Aguarde pelo menos 40µS.

10. Comando para limpar o display e posicionar o cursor na primeira linha, primeira coluna (esquerda).

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1

11. Aguarde pelo menos 1,8 ms.

12. Envie o comando para ligar o display sem cursor.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	1	0	0

13. Aguarde pelo menos 40µs.

14. Envie o comando para estabelecer o modo de operação. Por exemplo, deslocamento automático do cursor para a direita.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	1	0

15. Aguarde pelo menos 40µs.

Para o caso de utilização com quatro vias, o roteiro é ligeiramente diferente.

1. Aguarde pelo menos 15ms após a energização do LCD para garantir que ele já esteja operando corretamente.
2. Envie o comando 0x30 para o display.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	,0	0	1	1	-	-	-	-

3. Aguarde pelo menos 4ms para garantir o término da operação.

4. Envie novamente o comando 0x30 para o display.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	1	-	-	-	-

5. Aguarde pelo menos 100µs.

6. Envie novamente o comando 0x30 para o display.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	1	-	-	-	-

7. Aguarde pelo menos 40µs.

8. Estabeleça a comunicação em 4 vias.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	0	-	-	-	-

9. Aguarde pelo menos 40µs.

10. Estabeleça as condições de utilização. Neste caso, comunicação em quatro vias, display de duas linhas e matriz de 7x5.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	0		-	-	-
0	1	0	0	0	-	-	-	-

11. Aguarde pelo menos 40µs.

12. Comando para limpar o display e posicionar o cursor na primeira linha, primeira coluna (esquerda).

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	-	-	-	-
0	0	0	0	1	-	-	-	-

13. Aguarde pelo menos 1,8 ms.

14. Envie o comando para ligar o display sem cursor.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	-	-	-	-
0	1	1	0	0	-	-	-	-

15. Aguarde pelo menos 40µs.

16. Envie o comando para estabelecer o modo de operação. Por exemplo, deslocamento automático do cursor para a direita.

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	-	-	-	-
0	0	1	1	0	-	-	-	-

17. Aguarde pelo menos 40µs.

Comandos do LCD

Vejamos agora quais são realmente os comandos existentes no display. As nomenclaturas utilizadas respeitam a maioria dos data sheets existentes para esse tipo de componente. Para informações mais

detalhadas, recomendamos a consulta direta ao manual do modelo em uso no seu projeto. Os tempos apresentados também podem variar de um modelo para outro.

Limpeza do display

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1

Este comando apaga todo o display, escrevendo um espaço em branco (ASCII 20h) em todas as 32 posições disponíveis na memória interna do display (DDRAM). Depois disso, o cursor é posicionado no primeiro caractere (lado esquerdo) da primeira linha.

Este comando dura cerca de 1,64ms.

Retorno do cursor

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	(1/0)

Este comando faz com que o cursor retorne à posição inicial (primeira linha/primeira coluna) sem afetar a memória do display.

Este comando dura cerca de 1,64ms.

Modo de operação

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	I/D	S

Este comando possui dois bits de configuração que podem ser ajustados para deslocamento automático do cursor e deslocamento automático da mensagem:

Bit	Descrição
I/D = 0	0 cursor desloca-se automaticamente para a esquerda (decrementado) após uma operação de escrita ou leitura.
I/D=1	0 cursor desloca-se automaticamente para a direita (incrementado) após uma operação de escrita ou leitura.
S=0	Desliga o deslocamento da mensagem.
S=1	Liga o deslocamento da mensagem. A mensagem desloca-se para a direita ou para a esquerda, conforme o valor do bit I/D. Serve para implementar as funções Insert e Backspace.

Este comando dura cerca de 40µs.

Controle do display e cursor

RS	DB7	DB6	085	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	D	C	B

Este comando possui 3 bits para a configuração da visualização dos caracteres e do cursor:

Bit	Descrição
D=0	Inibe a visualização dos caracteres no display. A memória interna do LCD permanece inalterada.
D=1	Habilita a visualização dos caracteres no display, conforme os dados existentes na memória interna.
C=0	O cursor não é visível.
C=1	O cursor é visível como uma linha embaixo do caractere (oitava linha da matriz), como se fosse um sublinhado.
B=0	Desativa o cursor piscante.
B=1	Ativa o cursor piscante. Na verdade, neste modo o caractere da posição atual é alternado com um bloco negro em intervalos de 0,4s. Pode ser combinado com o cursor tipo sublinhado quando C=1.

Este comando dura cerca de 40µs.

Deslocamento do cursor ou da mensagem

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	S/C	R/L	-	-

Este comando possui dois bits de configuração para que o cursor ou a mensagem sejam deslocados para a direita ou para a esquerda, sem a necessidade de uma operação de escrita ou de leitura:

S/C	R/L	Descrição
0	0	O cursor desloca-se para a esquerda. Decrementa o endereço da memória interna DDRAM.
0	1	O cursor desloca-se para a direita. Incrementa o endereço da memória interna DDRAM.
1	0	Desloca a mensagem para a esquerda, em relação à posição atual
1	1	Desloca a mensagem e o cursor para a direita, em relação à posição atual do cursor. Função Backspace.

Este comando dura cerca de 40µs.

Configuração para utilização

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	DL	N	F	-	-

Este comando possui um bit para configuração da quantidade de vias de comunicação:

DL	Descrição
1	Comunicação feita pelas oito vias de dados, de DB0 a DB7.
0	Comunicação feita em quatro vias de dados, de DB4 a DB7. Inicialmente deve ser enviado, a parte alta do byte e, em seguida, a parte baixa.

Existem ainda mais dois bits de configuração para definir a quantidade de linhas e o tamanho da matriz do caractere:

N	F	Descrição
0	0	1 linha com matriz de 7x5 + cursor
0	1	1 linha com matriz de 10x5 + cursor
1	-	2 linhas com matriz de 7x5 + cursor (McLab2)

Este comando dura cerca de 40µs.

Posicionamento do cursor (DDRAM)

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	A	A	A	A	A	A	A

Para escrever um caractere em qualquer local do display, basta posicionar corretamente o cursor antes da operação de escrita do dado. Para isso, cada posição possui um endereço representado por AAAAAAAA. Para ficar ainda mais fácil, vamos montar uma tabela com os 32 caracteres do display e seus endereços absolutos (em Hexadecimal), já considerando DB7=1:

Coluna	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Linha 0	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
Linha 1	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF

Desta forma, fica fácil posicionar o cursor em qualquer local. Basta enviar ao LCD o comando relativo à posição desejada.

Este comando dura cerca de 40µs.

Escrita de um caractere

RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	D	D	D	D	D	D	D	D

Uma vez posicionado o cursor corretamente, basta enviar o caractere que se deseja escrever. Para isso devemos manter **RS=1** e colocar o código ASCII do caractere desejado nas vias de dados. Como no caso dos comandos, um pulso em **E** também deve ser executado. Este comando dura cerca de 40µs.

A tabela ASCII interna do display, gravada em ROM, pode alterar de um modelo para outro. Por isso, forneceremos somente a tabela ASCII não extendida (primeiros 127 caracteres) que permanece inalterada para a maioria dos modelos disponíveis. Quanto à parte alta, os displays mais comuns encontrados no Brasil possuem os caracteres na língua japonesa nessas posições.

Observe que as primeiras posições são definidas como oito endereços da memória CGRAM duplicados. Esses endereços acessam os caracteres especiais que podem ser desenhados na RAM dos módulos de LCD. Para a definição e uso destes caracteres especiais, consulte o manual do LCD.

Valores em Hexa		Digito mais significativo (primeiro)							
		0	1	2	3	4	5	6	7
	0	CGRAM (1)			0	@	P	"	p
	1	CGRAM (2)		!	1	A	Q	a	q
	2	CGRAM (3)		"	2	B	R	b	r
	3	CGRAM (4)		#	3	C	S	c	s
	4	CGRAM (5)		\$	4	D	T	d	t
	5	CGRAM (6)		%	5	E	U	e	u
	6	CGRAM (7)		&	6	F	V	f	v
	7D	CGRAM (8)		"	7	G	W	g	w
	8	CGRAM (1)	(8	H	x	h	x	
	9	CGRAM (2))	9	1	Y	i	y	
	A	CGRAM (3)	*	:	J	Z	j	z	
	B	CGRAM (4)	+	;	K	[k	{	
	C	CGRAM (5)	,	<	L	¥	l		
	D	CGRAM (6)	-	=	M]	m	}	
	E	CGRAM (7)	.	>	N	^	n	→	
	F	CGRAM (8)	/	?	O	_	o	↔	

Lógica do exemplo

Nosso exemplo para demonstração do LCD é bem reduzido. Simplesmente utilizaremos esse display para informar ao usuário qual botão foi pressionado. Para isso elaboramos uma rotina chamada ESCREVE que envia a informação passada pelo W para o display. Para evitarmos problemas de temporização, esta rotina já garante um tempo de espera de 1ms. Para os comandos que exigem um tempo maior, um delay adicional será dado após a chamada dessa rotina.

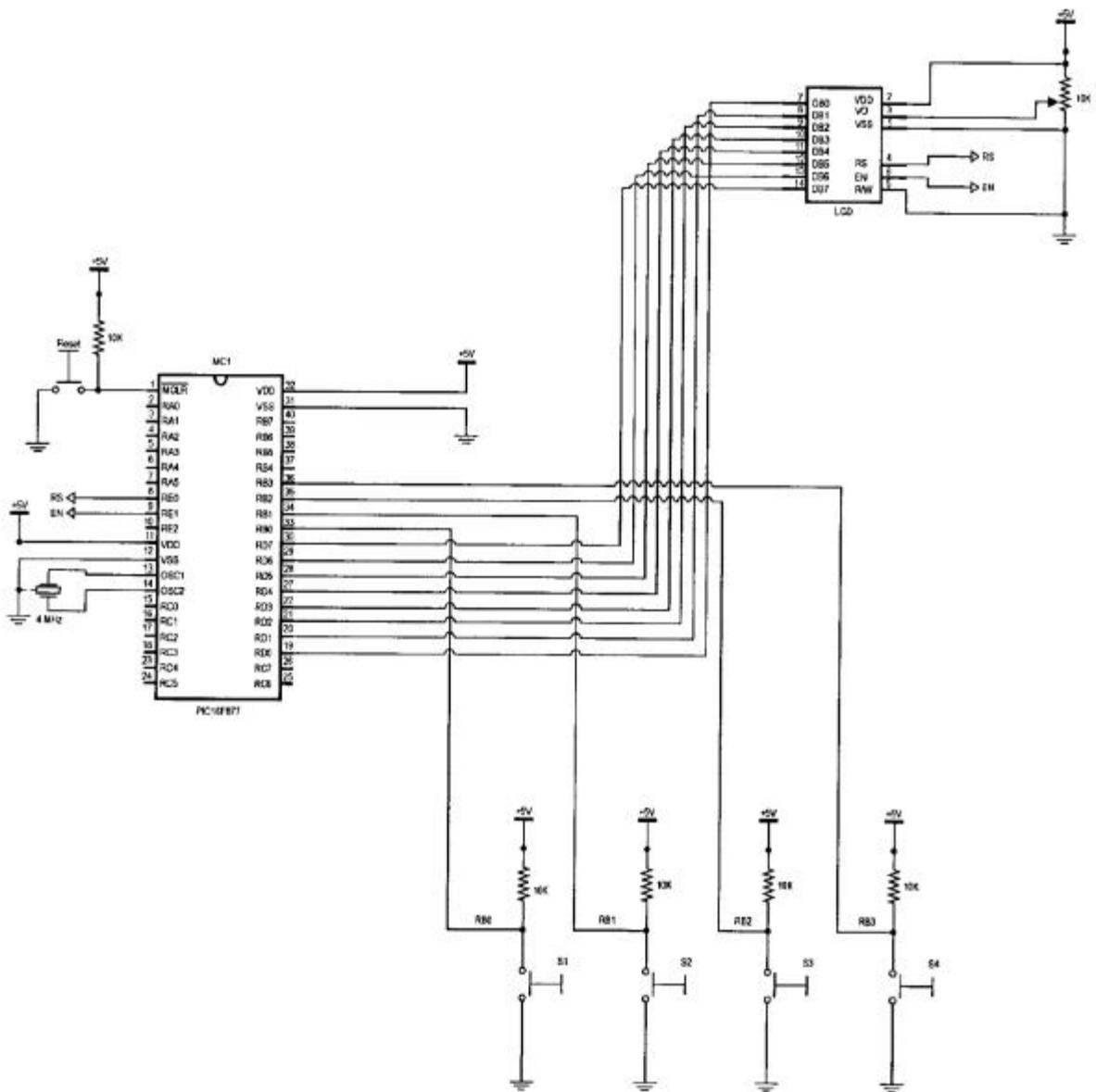
A rotina ESCREVE pode ser usada tanto para enviar comandos quanto dados para o display. Acontece que, como essa rotina não verifica ou altera o valor de RS, essa saída deve ser configurada antes da rotina ser chamada.

Seguindo o roteiro descrito na parte teórica deste capítulo, implementamos também uma rotina de preparação do LCD, chamada INICIALIZACAO_DISPLAY. Essa rotina configura o sistema para comunicação com oito vias, duas linhas, sem cursor visível e com movimento automático do cursor para a direita. Além disso, ela já limpa a tela e posiciona o cursor na primeira linha, primeiro caractere da esquerda.

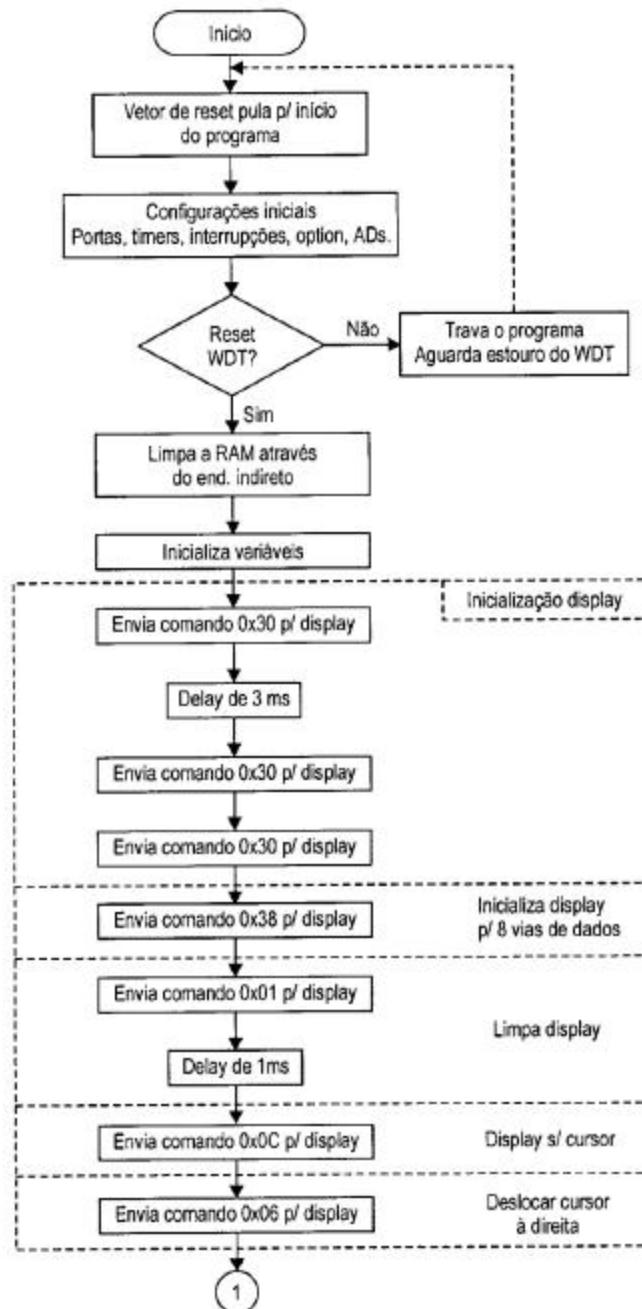
Para cada botão pressionado, posicionamos o cursor em um local diferente da tela e escrevemos o referido número do botão. Após a liberação, uma tela padrão é utilizada.

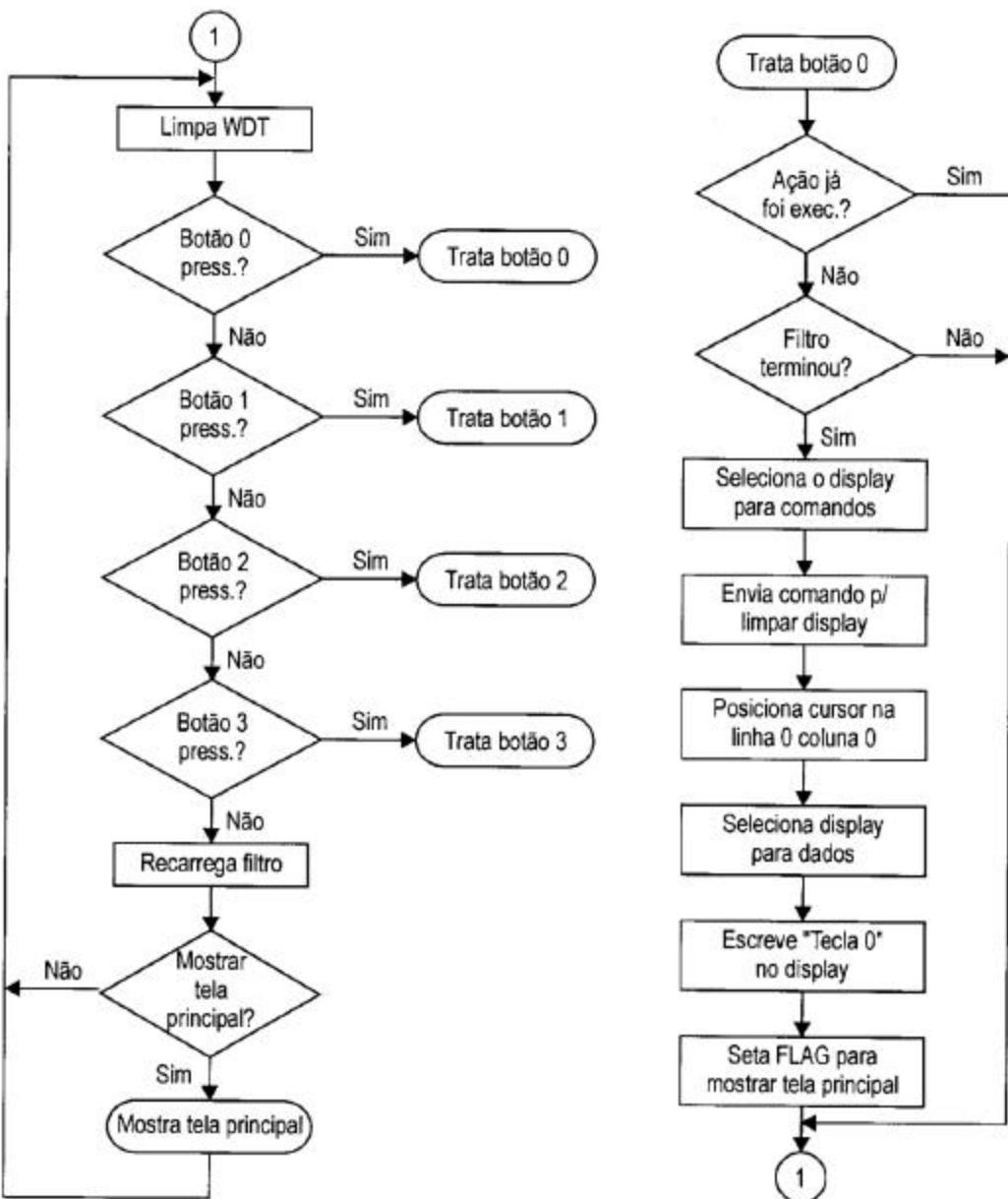
Este exemplo não utiliza nenhuma interrupção.

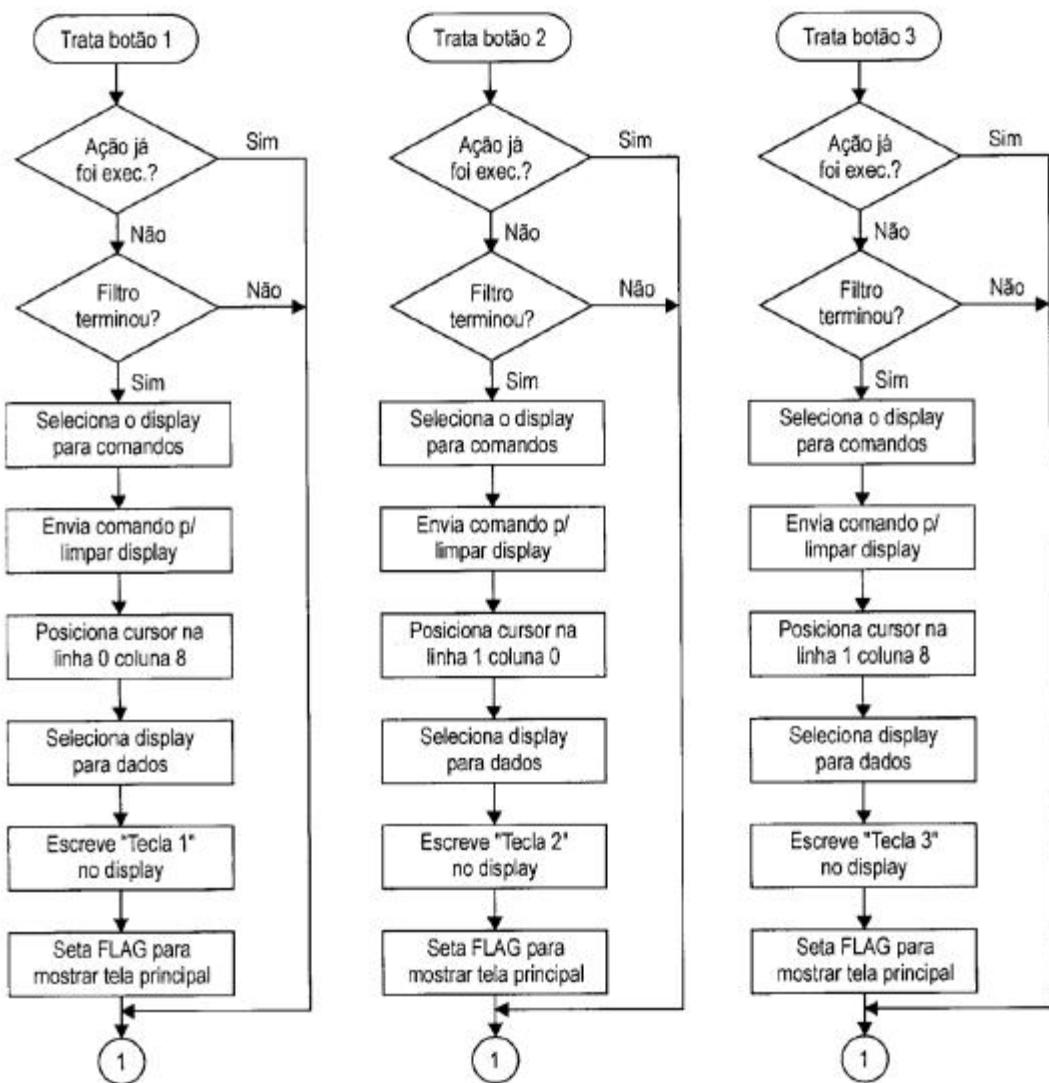
Esquema elétrico

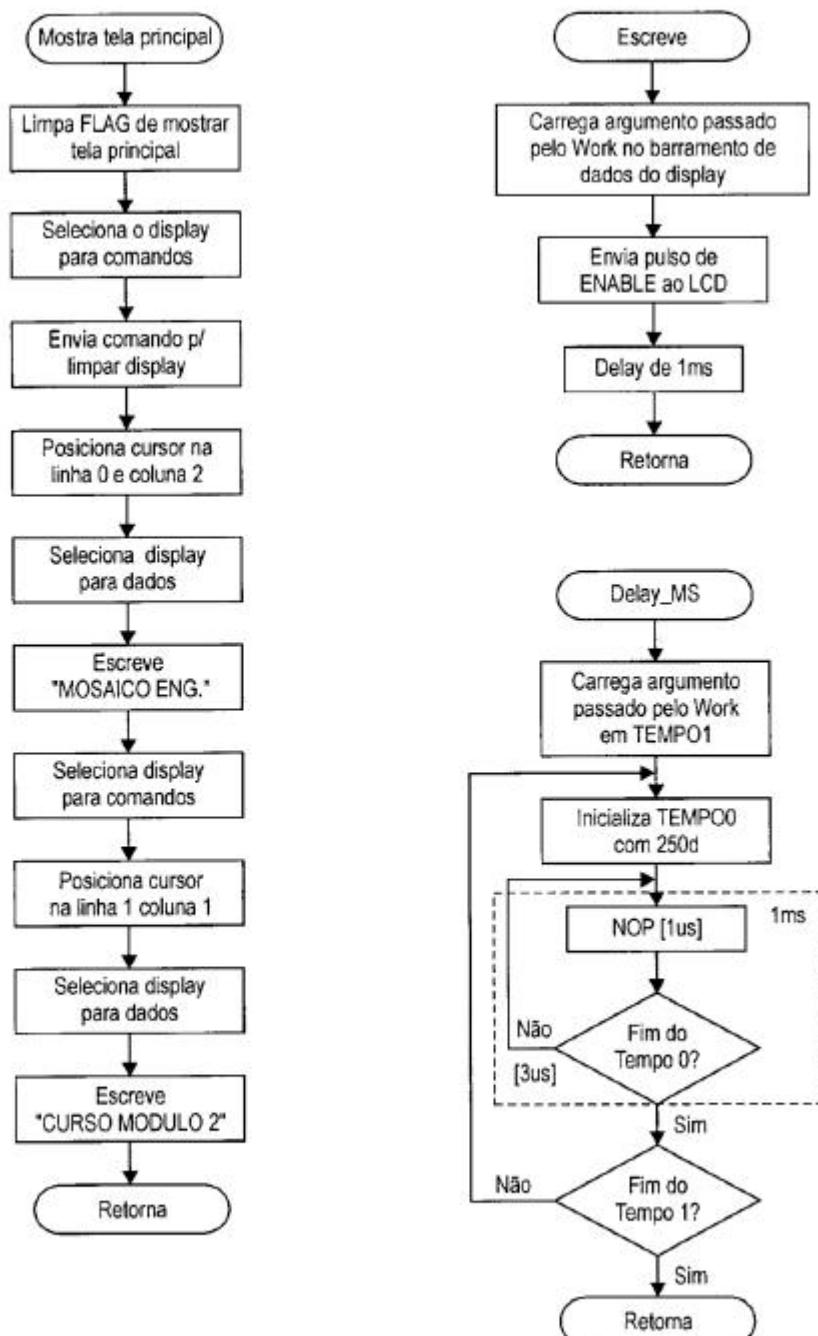


Fluxograma









```

;*****
;*      CONECTANDO O PIC - RECURSOS AVANÇADOS      *
;*      EXEMPLO 3          *
;*          *
;*      NICOLÁS CÉSAR LAVINIA e DAVID JOSÉ DE SOUZA  *
;*          *
;*****  

;* VERSÃO : 2.0          *
;* DATA : 24/02/2003      *
;*****  

;*****  

;*      DESCRIÇÃO GERAL      *
;*****  

; ESTE EXEMPLO FOI ELABORADO PARA EXPLICAR O FUNCIONAMENTO DO MÓDULO DE LCD.  

; FOI CRIADA UMA ROTINA PARA ESCREVER COMANDOS OU CACACTRES NO LCD. EXISTE  

; TAMBÉM UMA ROTINA DE INICIALIZAÇÃO NECESSÁRIA PARA A CORRETA CONFIGURAÇÃO  

; DO LCD. OS BOTÕES CONTINUAM SENDO MONITORADOS. UMA MENSAGEM É ESCRITA  

; NO LCD PARA CADA UM DOS BOTÕES, QUANDO O MESMO É PRESSIONADO.  

;  

;*****  

;*      CONFIGURAÇÕES PARA GRAVAÇÃO      *
;*****  

;*****  

;__CONFIG _CP_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &  

;_PWRTE_ON & _WDT_ON & _XT_OSC  

;  

;*****  

;*      DEFINIÇÃO DAS VARIÁVEIS      *
;*****  

; ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO LOGO NO INÍCIO DO BANCO 0  

;  

CBLOCK 0X20           ; POSIÇÃO INICIAL DA RAM

```

TEMPO1

TEMPO0 ; CONTADORES P/ DELAY

FILTRO_BOTOES ; FILTRO PARA RUIDOS

FLAG ; FLAG DE USO GERAL

ENDC

;*****

/* DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC */

;*****

; O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE
; OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE
; DE REDIGITAÇÃO.

#INCLUDE <P16F877A.INC> ; MICROCONTROLADOR UTILIZADO

;*****

/* DEFINIÇÃO DOS BANCOS DE RAM */

;*****

; OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR
; ENTRE OS BANCOS DE MEMÓRIA.

#DEFINE BANK1 BSF STATUS,RP0 ; SELECCIONA BANK1 DA MEMORIA RAM

#DEFINE BANK0 BCF STATUS,RP0 ; SELECCIONA BANK0 DA MEMORIA RAM

;*****

/* CONSTANTES INTERNAS */

;*****

; A DEFINIÇÃO DE CONSTANTES FACILITA A PROGRAMAÇÃO E A MANUTENÇÃO.

FILTRO_TECLA EQU .200 ; FILTRO P/ EVITAR RUIDOS DOS BOTÕES

;*****

```

;*          DECLARAÇÃO DOS FLAGs DE SOFTWARE          *
;*****  

;  

; A DEFINIÇÃO DE FLAGs AJUDA NA PROGRAMAÇÃO E ECONOMIZA MEMÓRIA RAM.  

;  

#DEFINE TELA_PRINCIPAL FLAG,0           ; FLAG P/ INDICAR QUE DEVE MOSTRAR  

                                         ; A TELA PRINCIPAL  

                                         ; 1-> MOSTRA TELA PRINCIPAL  

                                         ; 0-> TELA PRINCIPAL JÁ FOI MOSTRADA  

;  

;*****  

;  

;*          ENTRADAS          *  

;*****  

;  

; AS ENTRADAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E  

; FUTURAS ALTERAÇÕES DO HARDWARE.  

;  

#DEFINE BOTAO_0             PORTB,0      ; ESTADO DO BOTÃO 0  

                           ; 1 -> LIBERADO  

                           ; 0 -> PRESSIONADO  

;  

#DEFINE BOTAO_1             PORTB,1      ; ESTADO DO BOTÃO 1  

                           ; 1 -> LIBERADO  

                           ; 0 -> PRESSIONADO  

;  

#DEFINE BOTAO_2             PORTB,2      ; ESTADO DO BOTÃO 2  

                           ; 1 -> LIBERADO  

                           ; 0 -> PRESSIONADO  

;  

#DEFINE BOTAO_3             PORTB,3      ; ESTADO DO BOTÃO 3  

                           ; 1 -> LIBERADO  

                           ; 0 -> PRESSIONADO  

;  

;*****  

;  

;*          SAÍDAS          *  

;*****  

;  

; AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E  

Conectando o PIC 16F877A - Recursos Avançados

```

; FUTURAS ALTERAÇÕES DO HARDWARE.

```
#DEFINE DISPLAY      PORTD      ; BARRAMENTO DE DADOS DO DISPLAY

#DEFINE RS           PORTE,0    ; INDICA P/ O DISPLAY UM DADO OU COMANDO
                           ; 1 -> DADO
                           ; 0 -> COMANDO

#DEFINE ENABLE       PORTE,1    ; SINAL DE ENABLE P/ DISPLAY
                           ; ATIVO NA BORDA DE DESCIDA
```

/* VETOR DE RESET DO MICROCONTROLADOR */

; POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA

```
ORG 0X0000      ; ENDEREÇO DO VETOR DE RESET
GOTO CONFIG      ; PULA PARA CONFIG DEVIDO A REGIÃO
                  ; DESTINADA AS ROTINAS SEGUINTES
```

/* ROTINA DE DELAY (DE 1MS ATÉ 256MS) */

; ESTA É UMA ROTINA DE DELAY VARIÁVEL, COM DURAÇÃO DE 1MS X O VALOR PASSADO
; EM WORK (W).

DELAY_MS

```
MOVWF TEMPO1      ; CARREGA TEMPO1 (UNIDADES DE MS)
```

```
MOVLW .250
```

```
MOVWF TEMPO0      ; CARREGA TEMPO0 (P/ CONTAR 1MS)
```

```
CLRWDT          ; LIMPA WDT (PERDE TEMPO)
```

```
DECFSZ TEMPO0,F   ; FIM DE TEMPO0 ?
```

```
GOTO $-2         ; NÃO - VOLTA 2 INSTRUÇÕES
```

; SIM - PASSOU-SE 1MS

Conectando o PIC 16F877A - Recursos Avançados

```
DECFSZ TEMPO1,F          ; FIM DE TEMPO1 ?  
GOTO    $-6                ; NÃO - VOLTA 6 INSTRUÇÕES  
                           ; SIM  
RETURN                   ; RETORNA
```

```
;*****  
;*      ROTINA DE ESCRITA DE UM CARACTER NO DISPLAY      *  
;*****  
; ESTA ROTINA ENVIA UM CARACTER PARA O MÓDULO DE LCD. O CARACTER A SER  
; ESCRITO DEVE SER COLOCADO EM WORK (W) ANTES DE CHAMAR A ROTINA.
```

ESCREVE

```
MOVWF DISPLAY            ; ATUALIZA DISPLAY (PORTD)  
NOP                     ; PERDE 1US PARA ESTABILIZAÇÃO  
BSF      ENABLE           ; ENVIANDO PULSO DE ENABLE AO DISPLAY  
GOTO    $+1                ;..  
BCF      ENABLE           ;..
```

```
MOVLW .1  
CALL    DELAY_MS          ; DELAY DE 1MS  
RETURN                  ; RETORNA
```

```
;*****  
;*      ROTINA DE ESCRITA DA TELA PRINCIPAL      *  
;*****  
; ESTA ROTINA ESCREVE A TELA PRINCIPAL DO PROGRAMA, COM AS FRASES:  
; LINHA 1 - " MOSAICO ENG. "  
; LINHA 2 - "CURSO MODULO 2"
```

MOSTRA_TELA_PRINCIPAL

```
BCF      TELA_PRINCIPAL     ; LIMPA FLAG DE MOSTRAR TELA PRINCIPAL  
BCF      RS                 ; SELECCIONA O DISPLAY P/ COMANDOS  
MOVLW 0X01                  ; ESCREVE COMANDO PARA  
                           Conectando o PIC 16F877A - Recursos Avançados
```

```
CALL    ESCREVE           ; LIMPAR A TELA
MOVLW .1
CALL    DELAY_MS          ; DELAY DE 1MS

MOVLW 0X82                ; COMANDO PARA POSICIONAR O CURSOR
CALL    ESCREVE           ; LINHA 0 / COLUNA 2
BSF    RS                 ; SELECCIONA O DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS
; LETRAS DE "MOSAICO ENG."
MOVLW 'M'
CALL    ESCREVE
MOVLW 'O'
CALL    ESCREVE
MOVLW 'S'
CALL    ESCREVE
MOVLW 'A'
CALL    ESCREVE
MOVLW 'I'
CALL    ESCREVE
MOVLW 'C'
CALL    ESCREVE
MOVLW 'O'
CALL    ESCREVE
MOVLW ''
CALL    ESCREVE
MOVLW 'E'
CALL    ESCREVE
MOVLW 'N'
CALL    ESCREVE
MOVLW 'G'
CALL    ESCREVE
MOVLW ':'
CALL    ESCREVE
```

BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS
MOVLW 0XC1 ; COMANDO PARA POSICIONAR O CURSOR
CALL ESCREVE ; LINHA 1 / COLUNA 0
BSF RS ; SELECCIONA O DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS
; LETRAS DE "CURSO MODULO 2"
MOVLW 'C'
CALL ESCREVE
MOVLW 'U'
CALL ESCREVE
MOVLW 'R'
CALL ESCREVE
MOVLW 'S'
CALL ESCREVE
MOVLW 'O'
CALL ESCREVE
MOVLW ''
CALL ESCREVE
MOVLW 'M'
CALL ESCREVE
MOVLW 'O'
CALL ESCREVE
MOVLW 'D'
CALL ESCREVE
MOVLW 'U'
CALL ESCREVE
MOVLW 'L'
CALL ESCREVE
MOVLW 'O'
CALL ESCREVE
MOVLW ''
CALL ESCREVE
MOVLW '2'
CALL ESCREVE

RETURN

```
;*****  
;*      CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE *  
;*****  
;  
; NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS  
; CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA A  
; MÁQUINA E AGUARDA O ESTOURO DO WDT.
```

CONFIG

CLRF PORTA ; GARANTE AS SAÍDAS EM ZERO

CLRF PORTB

CLRF PORTC

CLRF PORTD

CLRF PORTE

BANK1 ; SELECCIONA BANCO 1 DA RAM

MOVLW B'11011111'

MOVWF TRISA ; CONFIGURA I/O DO PORTA

MOVLW B'11111111'

MOVWF TRISB ; CONFIGURA I/O DO PORTB

MOVLW B'11111111'

MOVWF TRISC ; CONFIGURA I/O DO PORTC

MOVLW B'00000000'

MOVWF TRISD ; CONFIGURA I/O DO PORTD

MOVLW B'00000100'

MOVWF TRISE ; CONFIGURA I/O DO PORTE

MOVLW B'11011111'

MOVWF OPTION_REG ; CONFIGURA OPTIONS

Conectando o PIC 16F877A - Recursos Avançados

; PULL-UPs DESABILITADOS
; INTER. NA BORDA DE SUBIDA DO RB0
; TIMER0 INCREM. PELO CICLO DE MÁQUINA
; WDT - 1:128
; TIMER - 1:1

MOVLW B'00000000'

MOVWF INTCON ; CONFIGURA INTERRUPÇÕES
; DESABILITADA TODAS AS INTERRUPÇÕES

MOVLW B'00000111'

MOVWF ADCON1 ; CONFIGURA CONVERSOR A/D
; CONFIGURA PORTA E PORTE COM I/O DIGITAL

BANK0 ; SELECCIONA BANCO 0 DA RAM

; AS INSTRUÇÕES A SEGUIR FAZEM COM QUE O PROGRAMA TRAVE QUANDO HOUVER UM
; RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT. DESTA FORMA,
; SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA UM ESTOURO DE WDT
; E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO START-UP DO PIC.

BTFSC STATUS,NOT_TO ; RESET POR ESTOURO DE WATCHDOG TIMER?
GOTO \$; NÃO - AGUARDA ESTOURO DO WDT
; SIM

,*****

;%* INICIALIZAÇÃO DA RAM *

,*****

; ESTA ROTINA IRÁ LIMPAR TODA A RAM DO BANCO 0, INDO DE 0X20 A 0X7F.
; EM SEGUITA, AS VARIÁVEIS DE RAM DO PROGRAMA SÃO INICIALIZADAS.

MOVLW 0X20

MOVWF FSR ; APONTA O ENDEREÇAMENTO INDIRETO PARA
; A PRIMEIRA POSIÇÃO DA RAM

LIMPA_RAM

```
CLRF    INDF          ; LIMPA A POSIÇÃO
INCF    FSR,F         ; INCREMENTA O PONTEIRO P/ A PRÓX. POS.
MOVF    FSR,W
XORLW  0X80          ; COMPARA O PONTEIRO COM A ÚLT. POS. +1
BTFSS  STATUS,Z      ; JÁ LIMPOU TODAS AS POSIÇÕES?
GOTO   LIMPA_RAM     ; NÃO - LIMPA A PRÓXIMA POSIÇÃO
                      ; SIM
```

```
BSF     TELA_PRINCIPAL ; INICIALIZA MOSTRANDO TELA PRINCIPAL
```

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

```
/* CONFIGURAÇÕES INICIAIS DO DISPLAY */
```

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

```
; ESTA ROTINA INICIALIZA O DISPLAY P/ COMUNICAÇÃO DE 8 VIAS, DISPLAY PARA 2
; LINHAS, CURSOR APAGADO E DESLOCAMENTO DO CURSOR À DIREITA.
```

```
INICIALIZACAO_DISPLAY
```

```
BCF     RS          ; SELECCIONA O DISPLAY P/ COMANDOS
```

```
MOVLW  0X30          ; ESCREVE COMANDO 0X30 PARA
```

```
CALL   ESCREVE       ; INICIALIZAÇÃO
```

```
MOVLW .3
```

```
CALL   DELAY_MS      ; DELAY DE 3MS (EXIGIDO PELO DISPLAY)
```

```
MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
```

```
CALL   ESCREVE       ; INICIALIZAÇÃO
```

```
MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
```

```
CALL   ESCREVE       ; INICIALIZAÇÃO
```

```
MOVLW B'00111000'    ; ESCREVE COMANDO PARA
```

```
CALL   ESCREVE       ; INTERFACE DE 8 VIAS DE DADOS
```

```
MOVLW B'00000001'    ; ESCREVE COMANDO PARA
```

```
Conectando o PIC 16F877A - Recursos Avançados
```

```

CALL    ESCREVE           ; LIMPAR TODO O DISPLAY

MOVlw .1

CALL    DELAY_MS           ; DELAY DE 1MS

MOVlw B'00001100'          ; ESCREVE COMANDO PARA

CALL    ESCREVE           ; LIGAR O DISPLAY SEM CURSOR

MOVlw B'00000110'          ; ESCREVE COMANDO PARA INCREM.

CALL    ESCREVE           ; AUTOMÁTICO À DIREITA

BSF     RS                ; SELECCIONA O DISPLAY P/ DADOS

;*****
;*          VARREDURA DOS BOTÕES
;*          LOOP PRINCIPAL
;*****


; A ROTINA PRINCIPAL FICA CHECANDO O ESTADO DOS BOTÕES. CASO ALGUM SEJA
; PRESSIONADO, A ROTINA DE TRATAMENTO DO BOTÃO É CHAMADA.

VARRE

CLRWDt           ; LIMPA WATCHDOG TIMER

BTFSs  BOTAO_0           ; O BOTÃO 0 ESTÁ PRESSIONADO ?

GOTO   TRATA_BOTAO_0      ; SIM - PULA P/ TRATA_BOTAO_0
                           ; NÃO

BTFSs  BOTAO_1           ; O BOTÃO 1 ESTÁ PRESSIONADO ?

GOTO   TRATA_BOTAO_1      ; SIM - PULA P/ TRATA_BOTAO_1
                           ; NÃO

BTFSs  BOTAO_2           ; O BOTÃO 2 ESTÁ PRESSIONADO ?

GOTO   TRATA_BOTAO_2      ; SIM - PULA P/ TRATA_BOTAO_2
                           ; NÃO

```

BTFS S BOTAO_3 ; O BOTÃO 3 ESTÁ PRESSIONADO ?

GOTO TRATA_BOTAO_3 ; SIM - PULA P/ TRATA_BOTAO_3

; NÃO

MOVLW FILTRO_TECLA ; CARREGA NO WORK O VALOR DE FILTRO_TECLA

MOVWF FILTRO_BOTOES ; SALVA EM FILTRO_BOTOES

; RECARREGA FILTRO P/ EVITAR RUIDOS

BTFS S TELA_PRINCIPAL ; DEVE MOSTRAR TELA PRINCIPAL ?

GOTO VARRE ; NÃO - VOLTA P/ VARRE

; SIM

CALL MOSTRA_TELA_PRINCIPAL

GOTO VARRE ; VOLTA PARA VARRER TECLADO

;*****

* TRATAMENTO DOS BOTÕES *

;*****

; ***** TRATAMENTO DO BOTÃO 0 *****

TRATA_BOTAO_0

MOVF FILTRO_BOTOES,F

BTFS C STATUS,Z ; FILTRO JÁ IGUAL A ZERO ?

; (FUNÇÃO JÁ FOI EXECUTADA?)

GOTO VARRE ; SIM - VOLTA P/ VARREDURA DO TECLADO

; NÃO

DECFSZ FILTRO_BOTOES,F ; FIM DO FILTRO ? (RUIDO?)

GOTO VARRE ; NÃO - VOLTA P/ VARRE

; SIM - BOTÃO PRESSIONADO

; OS COMANDOS A SEGUIR SÃO PARA ESCREVER A FRASE RELACIONADA AO BOTÃO 0

BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS

MOVLW 0X01

```

CALL    ESCREVE           ; COMANDO P/ LIMPAR A TELA
MOVLW .1
CALL    DELAY_MS          ; DELAY DE 1MS

MOVLW 0X80                ; COMANDO PARA POSICIONAR O CURSOR
CALL    ESCREVE           ; LINHA 0 / COLUNA 0
BSF    RS                 ; SELECCIONA O DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS
; LETRAS DE "TECLA 0"

MOVLW 'T'
CALL    ESCREVE
MOVLW 'E'
CALL    ESCREVE
MOVLW 'C'
CALL    ESCREVE
MOVLW 'L'
CALL    ESCREVE
MOVLW 'A'
CALL    ESCREVE
MOVLW ''
CALL    ESCREVE
MOVLW '0'
CALL    ESCREVE

BSF    TELA_PRINCIPAL     ; SETA FLAG P/ MOSTRAR TELA PRINCIPAL

GOTO    VARRE             ; VOLTA P/ VARREDURA DOS BOTÕES

; ***** TRATAMENTO DO BOTÃO 1 *****
TRATA_BOTAO_1
MOVF   FILTRO_BOTOES,F
BTFS  STATUS,Z            ; FILTRO JÁ IGUAL A ZERO ?
; (FUNÇÃO JÁ FOI EXECUTADA?)

Conectando o PIC 16F877A - Recursos Avançados

```

GOTO VARRE ; SIM - VOLTA P/ VARREDURA DO TECLADO
; NÃO

DECFSZ FILTRO_BOTOES,F ; FIM DO FILTRO ? (RUIDO?)

GOTO VARRE ; NÃO - VOLTA P/ VARRE
; SIM - BOTÃO PRESSIONADO

; OS COMANDOS A SEGUIR SÃO PARA ESCREVER A FRASE RELACIONADA AO BOTÃO 1

BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS

MOVLW 0X01

CALL ESCREVE ; COMANDO P/ LIMPAR A TELA

MOVLW .1

CALL DELAY_MS ; DELAY DE 1MS

MOVLW 0X88 ; COMANDO PARA POSICIONAR O CURSOR

CALL ESCREVE ; LINHA 0 / COLUNA 8

BSF RS ; SELECCIONA O DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS

; LETRAS DE "TECLA 1"

MOVLW 'T'

CALL ESCREVE

MOVLW 'E'

CALL ESCREVE

MOVLW 'C'

CALL ESCREVE

MOVLW 'L'

CALL ESCREVE

MOVLW 'A'

CALL ESCREVE

MOVLW ''

CALL ESCREVE

MOVLW '1'

CALL ESCREVE

```

BSF      TELA_PRINCIPAL           ; SETA FLAG P/ MOSTRAR TELA PRINCIPAL

GOTO    VARRE                  ; VOLTA P/ VARREDURA DOS BOTÕES

; ***** TRATAMENTO DO BOTÃO 2 *****
TRATA_BOTAO_2

MOVF    FILTRO_BOTOES,F
BTFSZ   STATUS,Z              ; FILTRO JÁ IGUAL A ZERO ?
                                ; (FUNÇÃO JA FOI EXECUTADA?)

GOTO    VARRE                  ; SIM - VOLTA P/ VARREDURA DO TECLADO
                                ; NÃO

DECFSZ FILTRO_BOTOES,F       ; FIM DO FILTRO ? (RUIDO?)

GOTO    VARRE                  ; NÃO - VOLTA P/ VARRE
                                ; SIM - BOTÃO PRESSIONADO

; OS COMANDOS A SEGUIR SÃO PARA ESCREVER A FRASE RELACIONADA AO BOTÃO 2

BCF      RS                   ; SELECCIONA O DISPLAY P/ COMANDOS

MOVLW  0X01

CALL    ESCREVE               ; COMANDO P/ LIMPAR A TELA

MOVLW .1

CALL    DELAY_MS               ; DELAY DE 1MS

MOVLW 0XC0                   ; COMANDO PARA POSICIONAR O CURSOR

CALL    ESCREVE               ; LINHA 1 / COLUNA 0

BSF      RS                   ; SELECCIONA O DISPLAY P/ DADOS

                                ; COMANDOS PARA ESCREVER AS
                                ; LETRAS DE "TECLA 2"

MOVLW 'T'
CALL    ESCREVE
MOVLW 'E'

```

```

CALL    ESCREVE
MOVLW 'C'
CALL    ESCREVE
MOVLW 'L'
CALL    ESCREVE
MOVLW 'A'
CALL    ESCREVE
MOVLW ''
CALL    ESCREVE
MOVLW '2'
CALL    ESCREVE

BSF    TELA_PRINCIPAL      ; SETA FLAG P/ MOSTRAR TELA PRINCIPAL

GOTO   VARRE               ; VOLTA P/ VARREDURA DOS BOTÕES

; ***** TRATAMENTO DO BOTÃO 3 *****
TRATA_BOTAO_3
MOVF   FILTRO_BOTOES,F
BTFSC  STATUS,Z            ; FILTRO JÁ IGUAL A ZERO ?
                            ; (FUNÇÃO JA FOI EXECUTADA?)
GOTO   VARRE               ; SIM - VOLTA P/ VARREDURA DO TECLADO
                            ; NÃO

DECFSZ FILTRO_BOTOES,F    ; FIM DO FILTRO ? (RUIDO?)
GOTO   VARRE               ; NÃO - VOLTA P/ VARRE
                            ; SIM - BOTÃO PRESSIONADO

; OS COMANDOS A SEGUIR SÃO PARA ESCREVER A FRASE RELACIONADA AO BOTÃO 3

BCF    RS                  ; SELECCIONA O DISPLAY P/ COMANDOS
MOVLW 0X01
CALL    ESCREVE             ; COMANDO P/ LIMPAR A TELA
MOVLW .1

```

Conectando o PIC 16F877A - Recursos Avançados

```

CALL    DELAY_MS           ; DELAY DE 1MS

MOVLW 0XC8                 ; COMANDO PARA POSICIONAR O CURSOR

CALL    ESCREVE             ; LINHA 1 / COLUNA 8

BSF    RS                  ; SELEciona o DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS
; LETRAS DE "TECLA 3"

MOVLW 'T'
CALL    ESCREVE

MOVLW 'E'
CALL    ESCREVE

MOVLW 'C'
CALL    ESCREVE

MOVLW 'L'
CALL    ESCREVE

MOVLW 'A'
CALL    ESCREVE

MOVLW ''
CALL    ESCREVE

MOVLW '3'
CALL    ESCREVE

BSF    TELA_PRINCIPAL      ; SETA FLAG P/ MOSTRAR TELA PRINCIPAL

GOTO    VARRE              ; VOLTA P/ VARREDURA DOS BOTÕES

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

END                      ; FIM DO PROGRAMA

```

Dicas e comentários

Apesar da estrutura do sistema ficar muito simples com a implementação da rotina ESCREVE, nunca se esqueça de confirmar o estado da saída RS antes de utilizá-la. Conforme foi empregado neste exemplo, recomendamos deixar RS=1 como valor padrão, alterando-o para 0 sempre que desejar escrever algum comando. Em seguida, retornar RS=1 (padrão) para enviar dados.

Para enviar um caractere ao LCD, não se esqueça que deve ser especificado seu código ASCII. Para facilitar sua vida, o MpLab efetua essa conversão quando você digita um caractere entre aspas simples (Ex: 'A'). Neste caso, existe diferença entre maiúsculas e minúsculas. Para caracteres não "printáveis", seu código pode ser diretamente definido (Ex: 0x35). O MpLab, assim como a maioria dos programas para Windows, também aceita a especificação de um caractere que não aparece no teclado através da combinação da tecla ALT seguido do código ASCII, imprimindo o resultado na tela. O problema é que nem sempre o código ASCII é equivalente ao mesmo símbolo para o LCD e para a fonte do MpLab.

Apesar da placa proposta (McLab2), possuir ligação com o módulo de LCD através de oito vias de dados é possível utilizá-la para testar e implementar a comunicação com quatro vias. Basta modificar a rotina de inicialização e a de escrita de um byte.

Exercícios propostos

Agora que o exemplo já foi estudado e esclarecido, aproveite para gerar novos problemas e soluções, seguindo os exercícios propostos:

1. Altere a comunicação para 4 vias.
2. Mantenha a tela principal disponível, somente quando o sistema é ligado. Após alguns segundos, mostre uma tela com o nome das quatro teclas e indique a tecla pressionada através de um caractere de seta (\leftarrow) ou outro qualquer.

Conversor Analógico-Digital Interno

Introdução

Como já dissemos mais de uma vez, vivemos em um mundo analógico. Sendo assim, por mais digital que seja nosso sistema, é quase impossível não trabalharmos com variáveis analógicas. Seja o valor de uma tensão, ou de uma temperatura, seja uma posição ou o sinal de um sensor qualquer. Conforme aumenta a complexibilidade do sistema, maior será o número de entradas analógicas que devem ser monitoradas pelo microcontrolador. Mas como isso será possível? Uma vez que o PIC só pode processar dados digitais, será necessário converter qualquer sinal analógico para valores digitais. Este é o assunto que trataremos neste capítulo.

Teoria

Para começarmos a entender melhor a teoria da conversão dos sinais analógicos em dados digitais, nada melhor que partirmos para um exemplo prático: os sensores de temperatura normalmente fornecem uma informação analógica (como, por exemplo, uma tensão) proporcional à temperatura e, portanto, para que esta possa ser analisada pelo microcontrolador, necessitamos de um conversor analógico digital (CAD ou simplesmente A/D).

O menor passo, ou resolução, de um CAD é dado diretamente pelo seu número de bits e pode ser expresso por:

$$\text{resolução} = \frac{V_{\text{ref}}}{2^n}$$

Em que: V_{ref} é uma tensão de referência e n é o número de bits do conversor.

Cada um dos n bits que compõem a informação digital representa uma parcela do valor da tensão analógica a ser convertida, de forma que a soma de todas as contribuições de cada um dos n bits forma a tensão de entrada do conversor A/D. Assim, a parcela de tensão proporcional ao bit m do conversor A/D é dada por:

$$V_{\text{entrada}} = \frac{b_m 2^{(m-1)}}{2^n} V_{\text{ref}}$$

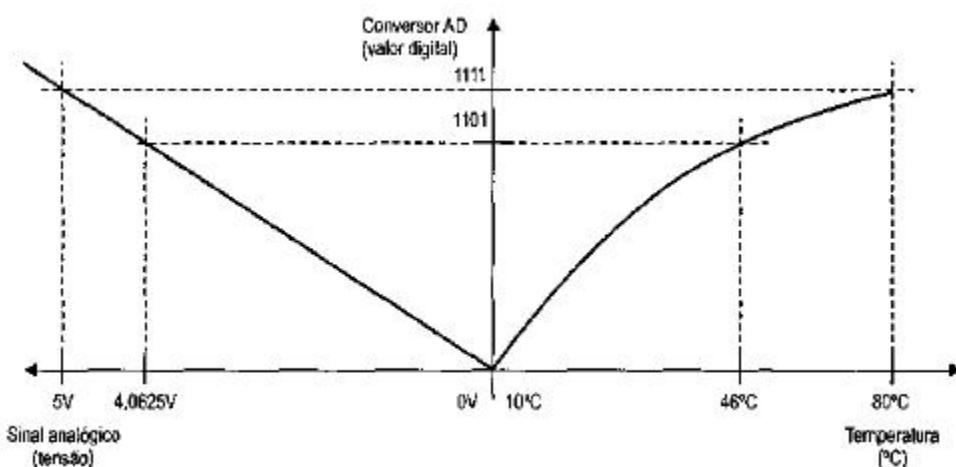
Em que: b_m é o valor do bit m, ou seja, 0 ou 1.

Veja, que apenas os bits em 1 representam algum valor em termos de tensão analógica, uma vez que os bits em zero não contribuem para formar a tensão de entrada. Quanto maior a quantidade de bits, maior a resolução e a precisão do conversor. O PIC 16F877A possui um conversor interno de 10 bits, mas existem outros modelos com 8 ou 12 bits.

Vamos supor que o CAD para o nosso exemplo da temperatura seja de quatro bits, a tensão de referência seja de 5V e o valor da conversão em binário seja 1101. A tensão de entrada, então, é:

$$V_{\text{entrada}} = \frac{(1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)}{2^4} (5) = 4,0625V$$

Este valor de tensão é equivalente a uma dada temperatura, e por isso o valor convertido (1101) é equivalente à mesma temperatura. Obviamente, para o exemplo da temperatura, assim como para a maioria dos demais casos práticos, não será necessária somente a conversão. Teremos também que nos preocupar com uma equação ou tabela para a adequação dos valores convertidos para a unidade desejada. Muitas vezes essa equação ou tabela será a responsável também pela linearização. Assim sendo, temos então duas conversões a serem feitas: a primeira de sinal analógico para valor digital, e a segunda de valor digital para a unidade realmente desejada, como por exemplo °C.



Existem diversas maneiras de implementarmos um conversor analógico/digital. Porém, como este livro trata do microcontrolador PIC16F877A, faremos um estudo aprofundado do sistema de conversão adotado pelo mesmo, que é denominado conversor de aproximação sucessiva.

Nesse tipo de conversor, a conversão é realizada do bit mais significativo para o menos significativo. Uma vez que o bit mais significativo (Msb) representa metade da tensão de referência, conhecer o estado deste bit (0 ou 1) já significa saber se a tensão de entrada é maior ou menor que a metade da referência. Conhecido o bit mais significativo, passa-se ao próximo bit, que representa a metade da metade da tensão de referência, ou seja, $\frac{1}{4}$ da tensão de referência. A conversão segue assim até o bit menos significativo (Lsb).

Vamos supor um CAD de quatro bits com tensão de referência de 5V:

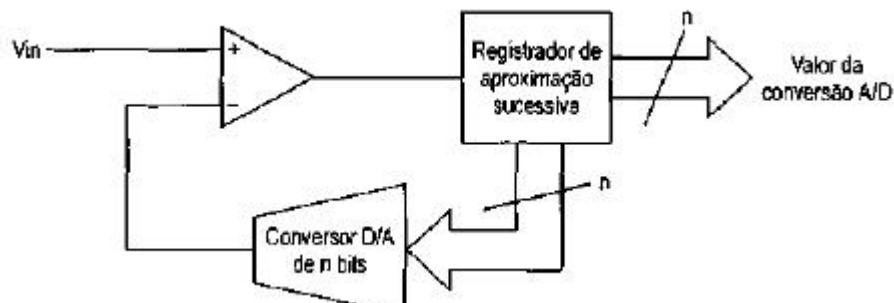
Bit	Tensão
4 (Msb)	2,5000 V
3	1,2500 V
2	0,6250 V
1 (Lsb)	0,325 V

Suponha que a tensão de entrada seja de 3,3V. Neste caso, o procedimento de conversão seria assim:

1. Testa-se o bit mais significativo, ou seja, a tensão de entrada é maior do que 2,5V? Sim, portanto, este bit vale 1.
2. Testa-se o próximo bit, ou seja, a tensão de entrada é maior do que 3,75V (2,5V + 1,25V)? Não, portanto, este bit é 0.
3. Testa-se o próximo bit, ou seja, a tensão de entrada é maior do que 3.125V (2,5V + 0.625V)? Sim, portanto, este bit é 1.
4. Finalmente testa-se o bit menos significativo, ou seja, a tensão de entrada é maior do que 3,4375V (2,5V + 0.625V + 0,3125V)? Não, portanto, este bit é 0 e o valor final da conversão em binário é 1010.

Essa forma de conversão é muito rápida, pois veja que para um conversor de n bits são necessárias " interações, independente do valor a ser convertido.

Em termos de hardware (diagrama de blocos), esse tipo de conversor pode ser representado por:



Recursos do PIC

Vamos agora estudar os modos de operação do conversor interno do PIC 16F877A, conhecendo seus recursos e os registradores de configuração e trabalho.

A primeira coisa que precisamos saber são as características desse conversor:

- Conversor interno de 10 bits, dando um total de 1024 pontos;
- Até oito canais de conversão, com diversas configurações entre analógicos e digitais;
- Quatro tipos de referência: V_{DD} (interna), V_{SS} (interna), V_{REF+} (externa) e V_{REF-} (externa);

- Freqüência de conversão baseada no clock da máquina ou em RC dedicado, possibilitando o funcionamento em modo SLEEP;
- Três ajustes de freqüência (divisores) para o clock de máquina;
- Dois tipos de justificação para o resultado da conversão: direita e esquerda;
- Um interrupção para término da conversão.

Bem, agora que já sabemos o resumo dos recursos analógicos do nosso PIC, aprenderemos a utilizá-los.

O primeiro conceito que deve ser entendido é que apesar do microcontrolador possuir diversos canais analógicos, internamente só existe um sistema de conversão. Por isso, somente um canal pode ser utilizado de cada vez. Vamos começar, então, aprendendo como configurar os canais corretamente.

Inicialmente devemos definir, conforme as necessidades do nosso projeto, qual a quantidade de canais analógicos que serão necessários. Como este PIC possui até oito canais analógicos, podemos utilizar todos ou só parte deles, deixando os demais pinos configurados como I/Os digitais. O problema é que, infelizmente, não é possível configurar individualmente cada canal, conforme nosso desejo. Existem valores padrões de configuração que devem ser respeitados.

Para configurarmos os canais analógicos precisamos alterar o valor dos bits existentes em **ADCON1<PCFG3:PCFG0>**:

PCFG3: PCFG0	AN7 RE2	AN6 RA1	AN5 RA0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF+ VDD	VREF- VSS	Canais Anal.	Ref. Ext.
0000	A	A	A	A	A	A	A	A	VDD	VSS	8	0
0001	A	A	A	A	VREF+	A	A	A	RA3	VSS	7	1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5	0
0011	D	D	D	A	VREF+	A	A	A	RA3	VSS	4	1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3	0
0101	D	D	D	D.	VREF.+	D	A	A	RA3	VSS	2	1
0110	D	D	D	D	D	D	D	D	VDD	VSS	0	0
0111	D	D	D	D	D	D	D	D	VDD	VSS	0	0
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2	6	2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6	0
1010	D	D	A	A	VREF+	A	A	A	RA3	VSS	5	1
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2	4	2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2	3	2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2	2	2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1	0
1111	D	D	D	D	VREF+	VREF-	D	A	RA3	RA2	1	2

Além da configuração dos pinos analógicos, conforme tabela apresentada, não pode ser esquecido de configurar corretamente os registradores **TRISA** e **TRISE**, de forma que os canais utilizados estejam ajustados para entrada (**TRIS = 1**). Caso um canal analógico esteja como saída, o sistema de conversão continuará funcionando, mas os valores convertidos serão equivalentes aos níveis alto ($1 = V_{DD}$) e baixo ($0 = V_{SS}$).

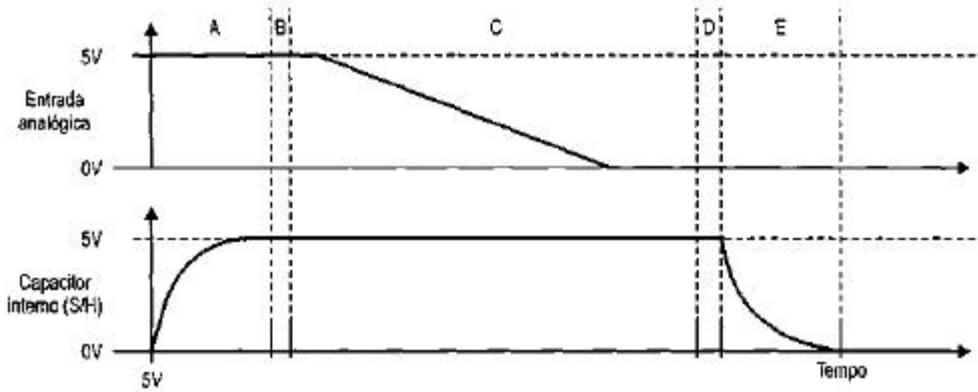
Outro ponto importante a ser observado nesta tabela diz respeito às tensões de referência. A conversão sempre será feita comparando-se a tensão no pino em relação as tensões de referência V_{REF+} e V_{REF-} . Desta forma, quando a tensão de entrada for igual à tensão V_{REF+} , a conversão resultará no valor máximo (1024) e quando a tensão de entrada for igual à V_{REF-} então o resultado da conversão será zero (0). Como já foi dito anteriormente, podemos ter quatro tipos de referências, sendo duas internas (V_{DD} e V_{SS}), e duas externas, ligadas aos pinos **RA2** (V_{REF+}) e **RA3** (V_{REF-}). Nossa entrada analógica poderá operar, por exemplo, com uma tensão variável de 0 a 5V. Para isso podemos utilizar as referências internas como $V_{REF+} = V_{DD}(5V)$ e $V_{REF-} = V_{SS}(0V)$. Em uma outra aplicação, nosso sensor pode variar de 1 a 4V. Para que não percamos resolução do A/D, podemos trabalhar com tensões de referência externas: V_{REF+} (**RA3**) = 4V e V_{REF-} (**RA2**) = 1V. O importante é respeitarmos os limites elétricos impostos a essas referências:

Referência	Mínimo (V)	Máximo (V)
V_{REF+}	$V_{DD} - 2,5$	$V_{DD} + 0,3$
V_{REF-}	$V_{SS} - 0,3$	$V_{REF+} - 2,0$
$(V_{REF+} - V_{REF-})$	2,0	$V_{DD} + 0,3$

O próximo ponto a ser aprendido diz respeito à velocidade e, consequentemente, aos tempos, de amostragem para a conversão A/D. Vamos começar entendendo como o sistema de conversão funciona internamente.

Para evitarmos problemas de ruído e variações da entrada analógica durante o processo de inversão (não podemos esquecer que nada é absolutamente instantâneo), o PIC utiliza internamente -n processo denominado Sample and Hold (S/H). Esse nome poderia ser traduzido como amostra e congela. Mas como isso é feito? Muito simples. Internamente o PIC possui um capacitor (120pF) que é ligado ao canal analógico em uso. Sendo assim, ele fica carregado com a tensão existente no canal amostra). Quando o processo de conversão é iniciado, este capacitor é desligado, automaticamente, do canal analógico, mantendo a tensão sobre o capacitor constante (congela). Por isso, durante todo o processo de conversão, a tensão utilizada é a existente no capacitor e não mais a do canal analógico. Mesmo que a tensão externa no canal varie, a conversão não será afetada.

Agora que já sabemos como o sistema funciona no âmbito do hardware, vamos conhecer quais são os tempos a serem respeitados de forma a não gerarmos erros durante a conversão. Para ilustrar o processo, mostraremos um gráfico resumido dos tempos de conversão:



Conforme o gráfico mostrado, podemos agora analisar cada um dos tempos envolvidos no sistema de conversão:

Código	Nome
A	Adequação do capacitor
B	Desligamento do capacitor
C	Conversão
D	Religamento do capacitor
E	Nova adequação do capacitor

Adequação do capacitor

Internamente o capacitor nunca fica desligado, exceto durante a conversão. Ele sempre encontra-se ligado ao sistema de entradas analógicas. Mas imaginemos que no momento o canal ativo esteja ligado ao GND. Desta forma o capacitor estará totalmente descarregado. No momento seguinte desejamos medir uma outra entrada analógica (outro canal) e por isso o sistema será chaveado internamente (veremos como isso é possível mais à frente). Caso o outro canal esteja com uma tensão de 5V, o capacitor interno terá que ser completamente carregado antes que possamos começar a efetuar a conversão. Obviamente essa carga não será instantânea. Para evitar problemas, sugerimos que o tempo padrão deixado para garantir a carga do capacitor seja de pelo menos 40µs. Obviamente, este é um tempo que garante a pior condição. Na verdade, o tempo de adequação pode ser bem menor que isso, chegando no mínimo a 10µs. Essa variação depende basicamente da diferença de tensão entre a entrada e o capacitor, da temperatura e da impedância de entrada. Essa impedância deve ser de no mínimo 50Q e no máximo 10 kΩ. Quanto menor a impedância, menor também o tempo de adequação. Para o cálculo exato desse tempo, consulte o *data sheet* do PIC.

Desligamento do capacitor

Depois de iniciada a conversão, o capacitor será desligado internamente. Isso é feito em aproximadamente 100ns, sendo um valor praticamente desprezível.

Conversão

O tempo de conversão já não é tão desprezível assim. Para entendermos o tempo de conversão precisaremos entender melhor sobre a freqüência de trabalho do A/D. Para que o sistema de conversão

funcione corretamente, um clock deve ser aplicado a ele. Cada período deste clock será chamado de T_{AD} e é equivalente ao tempo de conversão de 1 bit. Como nosso conversor é de 10 bits, o tempo total da conversão será de $10 T_{AD} + 2 T_{AD}$. Os dois períodos adicionais são para a adequação e o início da inversão. O valor de T_{AD} dependerá da freqüência empregada, e para isso o sistema possui um RC interno ou divisores para o oscilador da própria máquina, mas veremos isso mais adiante. Assim sendo, o tempo de conversão é $12 T_{AD}$.

Religamento do capacitor

Ao final da conversão, o valor dos registradores de resultado serão atualizados, o flag da interrupção será marcado e o capacitor será religado. O tempo mínimo recomendado para que tudo isso aconteça é $2 T_{AD}$.

Nova adequação do capacitor

Durante o tempo da conversão, o valor da tensão no capacitor interno não foi alterado; porém, ao término da conversão ele será religado à entrada analógica, que pode ter sofrido uma variação brusca (como no exemplo), seja por imposição do próprio sistema, seja por uma troca de canal. Por isso, entre uma conversão e outra é necessário uma nova adequação da carga do capacitor. Recomendamos novamente um tempo de $40\mu s$. As observações do item "Adequação do capacitor" também são válidas aqui para a diminuição deste tempo.

Agora que já entendemos um pouco mais as características e conceitos do A/D, vejamos como realmente devemos proceder para operar com ele.

Depois de termos configurado quais serão os canais utilizados (**ADCON1** e **TRIS**), teremos que configurar também a freqüência de trabalho. Para isso devemos ajustar 2 bits em **ADCON0<ADCS1:ADCS0>**:

ADCS1	ADCS0	Freqüência
0	0	Fosc /2
0	1	Fosc /8
1	0	Fosc /32
1	1	RC Interno

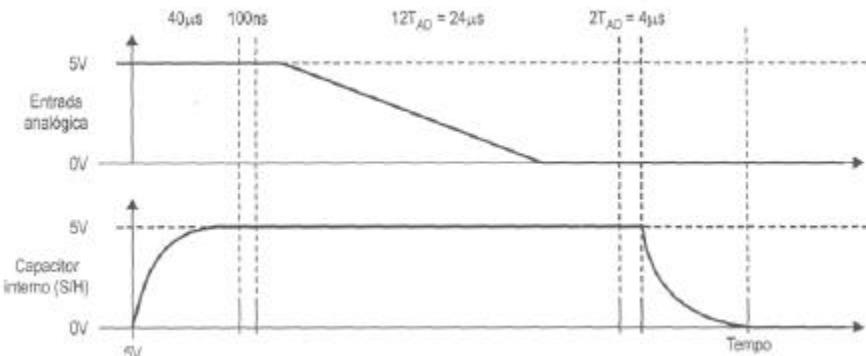
Para os três primeiros ajustes, a freqüência de trabalho do A/D será a freqüência do oscilador externo do PIC (Fosc) dividida por uma das opções (2, 8 ou 32). Desta forma, obteremos o valor de TAD. Por exemplo, para um cristal externo de 4MHz e uma opção de Fosc / 8 teremos:

$$T_{AD} = 8/4.000.000$$

$$T_{AD} = 0,000002s \text{ ou } 2\mu s$$

Neste caso, nosso T_{AD} é de $2\mu s$. O importante na escolha do cristal e da opção de divisão da freqüência é respeitar os valores mínimos de T_{AD} aceitos pelo PIC. No caso do PIC 16F877A (Standard) este valor deve ser maior que $1,6\mu s$. Valores muito altos para o T_{AD} também não são muito aconselháveis, e por isso recomendamos que não seja ultrapassado o limite de $20\mu s$.

Continuando com o exemplo dado, chequemos então a freqüência máxima de amostragem para a comutação entre canais:



Desconsiderando-se o tempo de desligamento (desprezível), teríamos um tempo total para a conversão de 68 μ s (40 + 24 + 4). Sendo assim, nossa freqüência de amostragem máxima seria de 14,7 kHz.

A última opção de escolha para o clock do A/D refere-se a um RC interno dedicado para essa finalidade. Neste caso o valor nominal para T_{AD} é de 4 μ s (pode variar de 2 a 6 μ s). Esta opção pode ser utilizada para que o sistema de conversão continue funcionando mesmo em modo SLEEP. Na verdade, este RC pode ser usado a qualquer momento, mas é recomendado que para sistemas que operem com freqüências superiores a 4 MHz este oscilador seja usado somente no modo SLEEP.

A última configuração necessária quanto às características de funcionamento do A/D diz respeito à forma como o resultado será armazenado nos registradores específicos. Como a conversão gera um resultado de 10 bits, serão necessários dois registradores para armazenar este valor. Para isso existem os registradores **ADRESH** e **ADRESL**, equivalentes à parte alta e à parte baixa do resultado. Acontece que a soma desses dois registradores resultam em 16 bits. Por isso, o resultado pode ser armazenado neles justificando pela esquerda ou direita. O bit de configuração **ADCON1<ADFM>** ajusta esta orientação:

ADFM	Freqüência
1	Justificado pela direita. Utiliza todos os bits de ADRESL<7:0> e somente 2 bits de ADRESH<1:0>
0 Justifi- cado pela direita Utiliza todos os bits de ADRE SH<1: 0> e some nte 2 bits de ADRE SH<1: 0>. 0 Justifi- cado pela esque- rda. Utiliza	Justificado pela esquerda. Utiliza todos os bits de ADRESL<7:0> e somente 2 bits de ADRESH<1:0>

Essa justificação é válida quando queremos, por exemplo, trabalhar com somente 8 bits do conversor. Neste caso, podemos justificar pela esquerda e acessarmos somente a parte mais significativa através de **ADRESH**. Com isso estaremos criando um filtro, onde jogamos fora os 2 bits menos significativos, que se encontram em **ADRESL**.

Agora que os ajustes da configuração já foram feitos, podemos finalmente efetuar uma conversão.

Começemos ligando o sistema de conversão. Isso é feito através de **SL<7:0>** e **ADCON0<ADON>**:

ADON	Estado do A/D
00	Sistema desligado
Sistema desliga do 1	Sistema ligado

Conectando ligado o pin 16 F877A - Recursos Avançados

O sistema pode ser mantido desligado sempre que não estiver sendo utilizado para redução do consumo.

A próxima ação é a escolha do canal a ser utilizado, entre os oito disponíveis. Essa escolha será feita através de **ADCON0<CHS2:CHS0>**:

CHS2 : CHS0	Canal Selecionado
000	Canal 0 (RA0/AN0)
001	Canal 1 (RA1/AN1)
010	Canal 2 (RA2/AN2)
011	Canal 3 (RA3/AN3)

CHS2 : CHS0	Canal Selecionado
100	Canal 4 (RA5/AN4)
101	Canal 5 (RE0/AN5)
110	Canal 6 (RE1/AN6)
111	Canal 7 (RE2/AN7)

O último passo é iniciar a conversão por meio do bit **ADCON0<GO/DONE>**:

GO/DONE	Estado do A/D
1	Inicia a conversão
1 Inicia a conversão	Indica que a conversão terminou. Caso seja forçado manualmente, cancela a conversão atual.

Q

Depois que iniciada a conversão, todo o processo será executado. O capacitor interno é desconectado. A conversão é efetuada (1TAD) e logo em seguida (no próximo ciclo da máquina), os registradores **ADRESH** e **ADRESL** são atualizados. Neste momento, o bit **ADCON0<GO/DONE>** volta automaticamente para 0 (zero) e poderá ser monitorado pelo programa para checar o término da conversão. Além disso, o flag da interrupção **PIR1<ADIF>** também é ativado. Caso a conversão seja cortada manualmente através de **ADCON0<GO/DONE>=0**, os registradores **ADRESH** e **ADRESL** não são alterados.

cancela a conversão

Para uso da interrupção de A/D, não se esqueça de efetuar os seguintes ajustes antes de iniciar a conversão:

- Limpar o flag **PIR1<ADIF>=0**;
- Ligar a interrupção de A/D em **PIE1 <ADIE>=1**;
- Ligar as interrupções de periféricos em **INTCON<PEIE>=1**;
- Ligar a chave geral das interrupções em **INTCON<GIE>=1**;

Resumo dos registradores associados ao A/D

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0Bh...	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
0ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	PSPIE	CCPIE	TMR2IE	TMR1IE
1Eh	ADRESH	Resultado da conversão (Parte alta)							
9Eh	ADRESL	Resultado da conversão (Parte baixa)							
1Fh	ACON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	-	ADON
9Fh	ADCON1	ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
85h	TRISA	-	-	Configuração do PORTA como Entrada(1) ou Saída(0)					
89h	TRISE	IBF	OBF	IBOV	PSP MODE	-	Configuração do PORTE como Entrada(1) ou Saída(0)		

Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Lógica do exemplo

Neste exemplo efetuaremos a leitura da tensão regulada pelo potenciômetro P2 da placa proposta (McLab2).

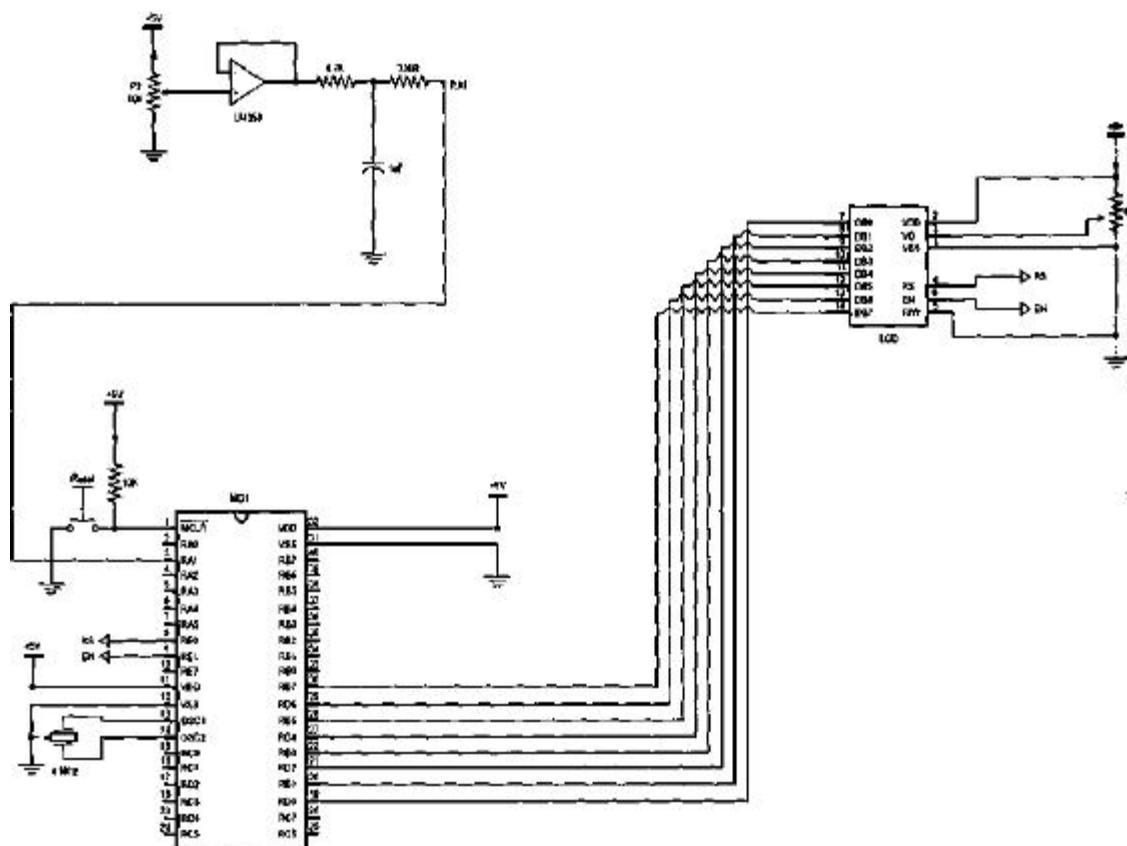
A conversão é feita diretamente no loop principal, sem a utilização de nenhuma interrupção, nem para checar o término da conversão, nem para definir uma freqüência de amostragem. Desta forma, a conversão será feita uma após a outra, na freqüência definida pelo período do loop principal.

Uma vez terminada a conversão, descartamos os 2 bits menos significativos e consideramos somente o resultado armazenado em **ADRESH**. Com isso já estaremos executando uma espécie de filtragem, evitando assim que o valor final fique oscilando.

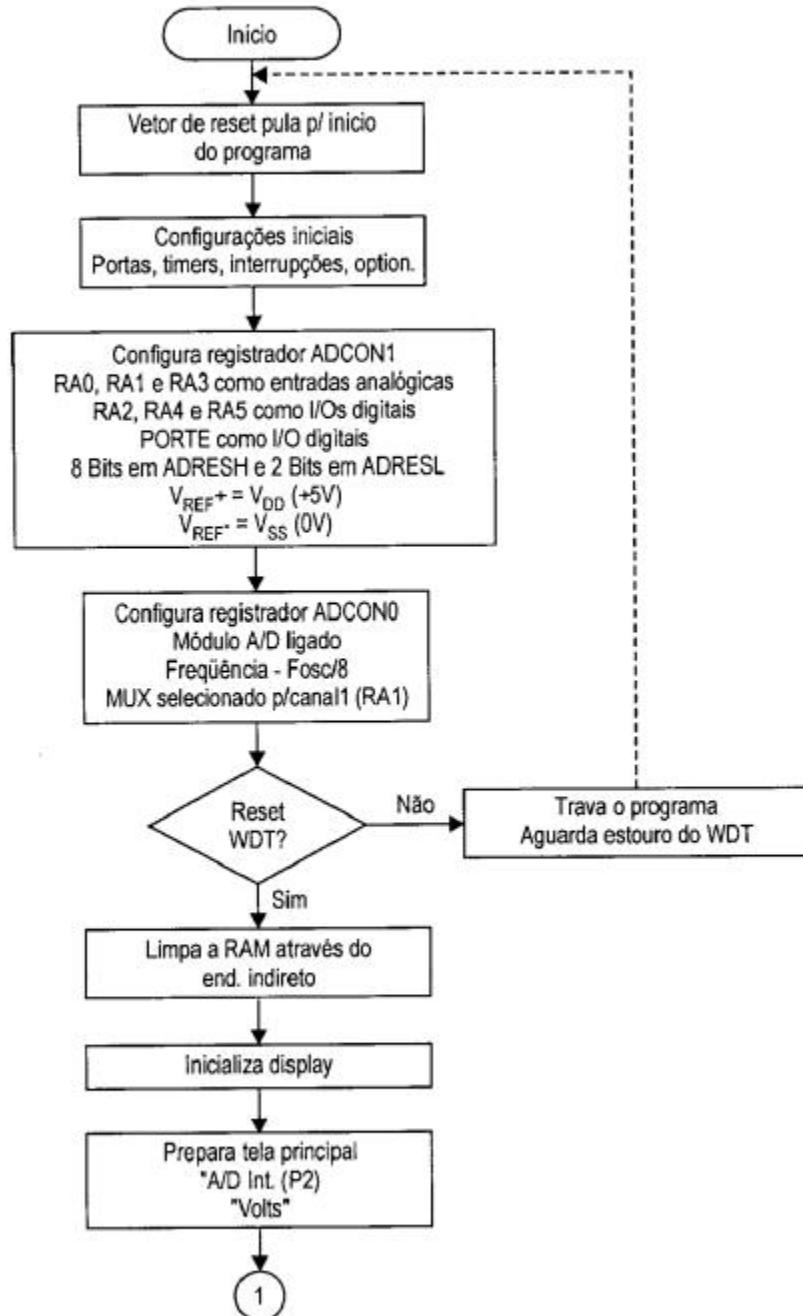
Aplicaremos então uma regra de três básica para converter o valor do A/D para a unidade desejada Volts. Conseguiremos isso considerando que, quando o A/D resultar em 0 (zero), a entrada possui 0,0V, e quando o A/D resultar em 255 a entrada é equivalente a 5,0V. Esta conta também diminuirá i resolução, aumentando ainda mais a estabilidade.

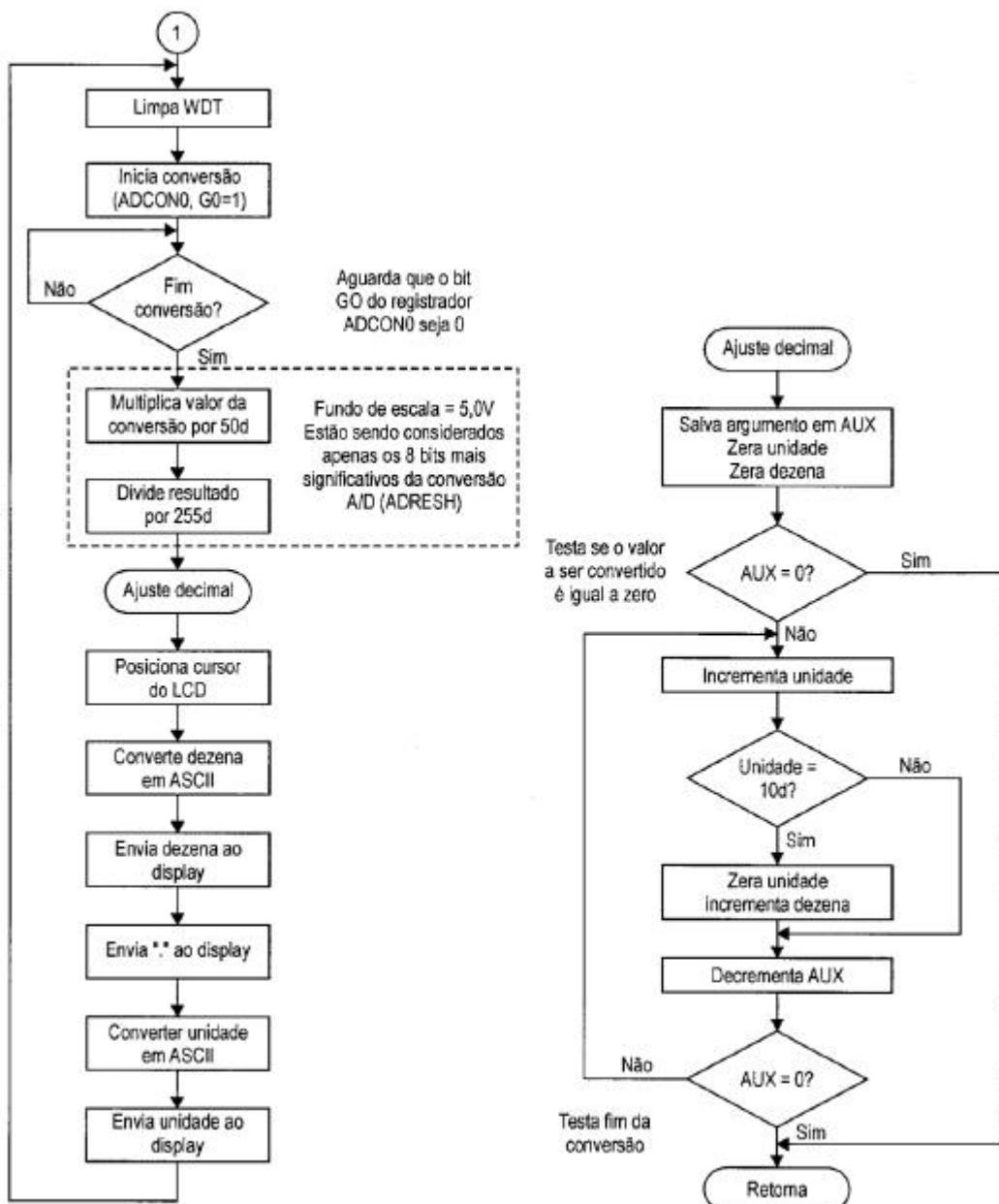
Por último escreveremos o valor, já em Volts, no LCD.

Esquema elétrico



Fluxograma





```

;*****
;*          CONECTANDO O PIC - RECURSOS AVANÇADOS      *
;*          EXEMPLO 4                                     *
;*
;*          NICOLÁS CÉSAR LAVINIA e DAVID JOSÉ DE SOUZA   *
;*
;*****
;*  VERSÃO : 2.0                                         *
;*  DATA : 24/02/2003                                    *
;*****


;*****
;*          DESCRIÇÃO GERAL                           *
;*****
;* ESTE EXEMPLO FOI ELABORADO PARA EXPLICAR O FUNCIONAMENTO DO MÓDULO DE
; CONVERSÃO ANALÓGICO DIGITAL INTERNO DO PIC. É CONVERTIDO O VALOR ANALÓGICO
; PRESENTE NO PINO RA2 DO MICROCONTROLADOR, SENDO QUE ESTE VALOR PODE SER
; ALTERADO ATRAVÉS DO POTENCIÔMETRO P2 DA PLACA MCLAB2. O VALOR DA CONVERSÃO
; A/D É AJUSTADO NUMA ESCALA DE 0 À 5V E MOSTRADO NO LCD.
; FORAM UTILIZADAS ROTINAS DE MULTIPLICAÇÃO DE 8x8 E DIVISÃO DE 16x16. ESTAS
; ROTINAS FORAM RETIRADAS DE APPLICATION NOTES DA PRÓPRIA MICROCHIP.
;

;*****
;*          CONFIGURAÇÕES PARA GRAVAÇÃO                 *
;*****


__CONFIG _CP_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &
_PWRTE_ON & _WDT_ON & _XT_OSC

;*****
;*          DEFINIÇÃO DAS VARIÁVEIS                   *
;*****
;* ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO LOGO NO INÍCIO DO BANCO 0

CBLOCK 0X20           ; POSIÇÃO INICIAL DA RAM

ACCaHI                ; ACUMULADOR a DE 16 BITS UTILIZADO
ACCaLO                ; NA ROTINA DE DIVISÃO

ACCbHI                ; ACUMULADOR b DE 16 BITS UTILIZADO
ACCbLO                ; NA ROTINA DE DIVISÃO

ACCcHI                ; ACUMULADOR c DE 16 BITS UTILIZADO
ACCcLO                ; NA ROTINA DE DIVISÃO

ACCdHI                ; ACUMULADOR d DE 16 BITS UTILIZADO
ACCdLO                ; NA ROTINA DE DIVISÃO

temp                  ; CONTADOR TEMPORÁRIO UTILIZADO
                      ; NA ROTINA DE DIVISÃO

H_byte                ; ACUMULADOR DE 16 BITS UTILIZADO
L_byte                ; P/ RETORNAR O VALOR DA ROTINA
                      ; DE MULTIPLICAÇÃO

mulplr               ; OPERADOR P/ ROTINA DE MULTIPLICAÇÃO
mulnd                ; OPERADOR P/ ROTINA DE MULTIPLICAÇÃO

TEMPO0                ; TEMPORIZADORES P/ ROTINA DE DELAY
TEMPO1                ; 

AUX                   ; REGISTRADOR AUXILIAR DE USO GERAL

UNIDADE              ; ARMAZENA VALOR DA UNIDADE DA
TENSÃO                ; 

DEZENA                ; ARMAZENA VALOR DA DEZENA DA TENSÃO

ENDC

;*****
;*          DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC    *
;*****
;* O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE
; OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE
; DE REDIGITAÇÃO.

#include <P16F877A.INC>           ; MICROCONTROLADOR UTILIZADO
Conectando o PIC 16F877A - Recursos Avançados

```

```

;*****
; *      DEFINIÇÃO DOS BANCOS DE RAM      *
;*****
; OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR
; ENTRE OS BANCOS DE MEMÓRIA.

#define BANK1 BSF      STATUS,RP0      ; SELEciona BANK1 DA MEMORIA RAM
#define BANK0 BCF      STATUS,RP0      ; SELEciona BANK0 DA MEMORIA RAM

;*****
; *      CONSTANTES INTERNAS      *
;*****
; A DEFINIÇÃO DE CONSTANTES FACILITA A PROGRAMAÇÃO E A MANUTENÇÃO.

; ESTE PROGRAMA NÃO UTILIZA NENHUMA CONSTANTE.

;*****
; *      DECLARAÇÃO DOS FLAGs DE SOFTWARE      *
;*****
; A DEFINIÇÃO DE FLAGs AJUDA NA PROGRAMAÇÃO E ECONOMIZA MEMÓRIA RAM.

; ESTE PROGRAMA NÃO UTILIZA NENHUM FLAG DE USUÁRIO

;*****
; *      ENTRADAS      *
;*****
; AS ENTRADAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

; ESTE PROGRAMA UTILIZA APENAS UMA ENTRADA P/ O CONVERSOR A/D.
; ESTA ENTRADA NÃO PRECISA SER DECLARADA, POIS O SOFTWARE NUNCA FAZ
; REFERÊNCIA A ELA DE FORMA DIRETA, POIS O CANAL A/D A SER CONVERTIDO É
; SELECIONADO NO REGISTRADOS ADCON0 DE FORMA BINÁRIA E NÃO ATRAVÉS DE
; DEFINES. PORÉM PARA FACILITAR O ENTENDIMENTO DO HARDWARE VAMOS DECLARAR
; ESTA ENTRADA NORMALMENTE.

#define CAD_P2 PORTA,1          ; ENTRADA A/D P/ O POTENCIÔMETRO P2

;*****
; *      SAÍDAS      *
;*****
; AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#define DISPLAY     PORTD      ; BARRAMENTO DE DADOS DO DISPLAY
#define RS          PORTE,0    ; INDICA P/ O DISPLAY UM DADO OU COMANDO
                           ; 1 -> DADO
                           ; 0 -> COMANDO
#define ENABLE      PORTE,1    ; SINAL DE ENABLE P/ DISPLAY
                           ; ATIVO NA BORDA DE DESCIDA

;*****
; *      VETOR DE RESET DO MICROCONTROLADOR      *
;*****
; POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA

org 0x0000      ; ENDEREÇO DO VETOR DE RESET
goto CONFIG      ; PULA PARA CONFIG DEVIDO A REGIÃO
                 ; DESTINADA AS ROTINAS SEGUINTEs

;*****
; *      ROTINA DE DELAY (DE 1MS ATÉ 256MS)      *
;*****
; ESTA É UMA ROTINA DE DELAY VARIÁVEL, COM DURAÇÃO DE 1MS X O VALOR PASSADO
; EM WORK (W).

DELAY_MS
    movwf TEMPO1      ; CARREGA TEMPO1 (UNIDADES DE MS)
    movlw .250
    movwf TEMPO0      ; CARREGA TEMPO0 (P/ CONTAR 1MS)

    clrwdt      ; LIMPA WDT (PERDE TEMPO)
    decfsz TEMPO0,f ; FIM DE TEMPO0 ?
    goto $-2        ; NÃO - VOLTA 2 INSTRUÇÕES

```

```

        ; SIM - PASSOU-SE 1MS
DECFSZ TEMPO1,F          ; FIM DE TEMPO1 ?
GOTO    $-6                ; NÃO - VOLTA 6 INSTRUÇÕES
                            ; SIM
RETURN                   ; RETORNA

;*****
; *      ROTINA DE ESCRITA DE UM CARACTER NO DISPLAY      *
;*****
; ESTA ROTINA ENVAI UM CARACTER PARA O MÓDULO DE LCD. O CARACTER A SER
; ESCRITO DEVE SER COLOCADO EM WORK (W) ANTES DE CHAMAR A ROTINA.

ESCREVE
    MOVWF DISPLAY           ; ATUALIZA DISPLAY (PORTD)
    NOP                     ; PERDE 1US PARA ESTABILIZAÇÃO
    BSF      ENABLE          ; ENVIA UM PULSO DE ENABLE AO DISPLAY
    GOTO    $+1
    BCF      ENABLE          ; ..
    MOVLW .1
    CALL    DELAY_MS         ; DELAY DE 1MS
    RETURN                  ; RETORNA

;*****
; *      AJUSTE DECIMAL          *
; *      W [HEX] = DEZENA [DEC] ; UNIDADE [DEC]          *
;*****
; ESTA ROTINA RECEBE UM ARGUMENTO PASSADO PELO WORK E RETORNA NAS VARIÁVEIS
; DEZENA E UNIDADE O NÚMERO BCD CORRESPONDENTE AO PARÂMETRO PASSADO.

AJUSTE_DECIMAL
    MOVWF AUX                ; SALVA VALOR A CONVERTER EM AUX
    CLRF     UNIDADE          ; RESETA REGISTRADORES
    CLRF     DEZENA
    MOVF    AUX,F             ; VALOR A CONVERTER = 0 ?
    BTFSC   STATUS,Z          ; SIM - RETORNA
    RETURN
    INCF    UNIDADE,F         ; NÃO
    INCF    UNIDADE,F         ; INCREMENTA UNIDADE
    MOVF    UNIDADE,W          ; UNIDADE = 10d ?
    XORLW  0X0A
    BTFSS   STATUS,Z          ; NÃO
    GOTO    $+3
    CLRF    UNIDADE            ; SIM
    INCF    DEZENA,F          ; RESETA UNIDADE
    INCF    DEZENA,F          ; INCREMENTA DEZENA
    DECFSZ AUX,F             ; FIM DA CONVERSÃO ?
    GOTO    $-.8
    RETURN                  ; NÃO - VOLTA P/ CONTINUAR CONVERSÃO
                            ; SIM

;*****
; *      ROTINA DE DIVISÃO          *
;*****
; Double Precision Division
;*****
; Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
;             Remainder in ACCc (16 bits)
; (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
; (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
; (c) CALL D_divF
; (d) The 16 bit result is in location ACCbHI & ACCbLO
; (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
;*****


D_divF
    MOVLW .16
    MOVWF temp                 ; CARREGA CONTADOR PARA DIVISÃO
    MOVF    ACCbHI,W
    MOVWF ACCdHI
    MOVF    ACCbLO,W
    MOVWF ACCdLO                ; SALVA ACCb EM ACCd

```

```

CLRF ACCbHI
CLRF ACCbLO ; LIMPA ACCb

CLRF ACCcHI
CLRF ACCcLO ; LIMPA ACCc

DIV
BCF STATUS,C
RLF ACCdLO,F
RLF ACCdHI,F
RLF ACCcLO,F
RLF ACCcHI,F
MOVF ACCaHI,W
SUBWF ACCcHI,W ;check if a>c
BTFS S STATUS,Z
GOTO NOCHK
MOVF ACCaLO,W
SUBWF ACCcLO,W ;if msb equal then check lsb

NOCHK
BTFS S STATUS,C ;carry set if c>a
GOTO NOGO
MOVF ACCaLO,W ;c-a into c
SUBWF ACCcLO,F
BTFS S STATUS,C
DEC F ACCcHI,F
MOVF ACCaHI,W
SUBWF ACCcHI,F
BSF STATUS,C ;shift a 1 into b (result)

NOGO
RLF ACCbLO,F
RLF ACCbHI,F

DECFSZ temp,F ; FIM DA DIVISÃO ?
GOTO DIV ; NÃO - VOLTA P/ DIV
; SIM
RETURN ; RETORNA

```

```

;*****
; *          ROTINA DE MULTIPLICAÇÃO          *
;*****
;
; 8x8 Software Multiplier
; ( Fast Version : Straight Line Code )
;*****


; The 16 bit result is stored in 2 bytes
; Before calling the subroutine " mpv ", the multiplier should
; be loaded in location " mulpl ", and the multiplicand in
; " mulcnd ". The 16 bit result is stored in locations
; H_byte & L_byte.
; Performance :
;     Program Memory : 37 locations
;     # of cycles   : 38
;     Scratch RAM   : 0 locations
;*****


; *****
; Define a macro for adding & right shifting
; *****

mult MACRO bit ; Begin macro

    BTFSC mulpl,bit
    ADDWF H_byte,F
    RRF H_byte,F
    RRF L_byte,F

ENDM ; End of macro

; *****
; Begin Multiplier Routine
; *****

mpy_F
    CLRF H_byte
    CLRF L_byte

```

```

MOVF mulcnd,W           ; move the multiplicand to W reg.
BCF STATUS,C             ; Clear carry bit in the status Reg.

mult 0
mult 1
mult 2
mult 3
mult 4
mult 5
mult 6
mult 7

RETURN                   ; RETORNA

;*****
; *      CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE
; *****
; NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS
; CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA A
; MÁQUINA E AGUARDA O ESTOIRO DO WDT.

CONFIG
    CLRF PORTA           ; GARANTE TODAS AS SAÍDAS EM ZERO
    CLRF PORTB
    CLRF PORTC
    CLRF PORTD
    CLRF PORTE

    BANK1                ; SELECCIONA BANCO 1 DA RAM

    MOVLW B'11111111'
    MOVWF TRISA          ; CONFIGURA I/O DO PORTA

    MOVLW B'11111111'
    MOVWF TRISB          ; CONFIGURA I/O DO PORTB

    MOVLW B'11111111'
    MOVWF TRISC          ; CONFIGURA I/O DO PORTC

    MOVLW B'00000000'
    MOVWF TRISD          ; CONFIGURA I/O DO PORTD
    MOVLW B'00000100'
    MOVWF TRISE          ; CONFIGURA I/O DO PORTE

    MOVLW B'11011011'
    MOVWF OPTION_REG     ; CONFIGURA OPTIONS
                        ; PULL-UPs DESABILITADOS
                        ; INTER. NA BORDA DE SUBIDA DO RB0
                        ; TIMER0 INCREM. PELO CICLO DE MÁQUINA
                        ; WDT - 1:8
                        ; TIMER - 1:1

    MOVLW B'00000000'
    MOVWF INTCON          ; CONFIGURA INTERRUPÇÕES
                        ; DESABILITA TODAS AS INTERRUPÇÕES

    MOVLW B'00000100'
    MOVWF ADCON1          ; CONFIGURA CONVERSOR A/D
                        ; RA0, RA1 E RA3 COMO ANALÓGICO
                        ; RA2, RA4 E RA5 COMO I/O DIGITAL
                        ; PORTE COMO I/O DIGITAL
                        ; JUSTIFICADO À ESQUERDA
                        ; 8 BITS EM ADRESH E 2 BITS EM ADRESL
                        ; Vref+ = VDD (+5V)
                        ; Vref- = GND ( 0V)

    BANK0                ; SELECCIONA BANCO 0 DA RAM

    MOVLW B'01001001'
    MOVWF ADCON0          ; CONFIGURA CONVERSOR A/D
                        ; VELOCIDADE -> Fosc/8
                        ; CANAL 1
                        ; MÓDULO LIGADO

; AS INSTRUÇÕES A SEGUIR FAZEM COM QUE O PROGRAMA TRAVE QUANDO HOUVER UM
; RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT. DESTA FORMA,
; SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA UM ESTOIRO DE WDT

```

; E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO START-UP DO PIC.

```
BTFS C STATUS,NOT_TO      ; RESET POR ESTOURO DE WATCHDOG TIMER ?
GOTO $                      ; NÃO - AGUARDA ESTOURO DO WDT
                                ; SIM
```

; * INICIALIZAÇÃO DA RAM *

; ESTA ROTINA IRÁ LIMPAR TODA A RAM DO BANCO 0, INDO DE 0X20 A 0X7F.
; EM SEGUIDA, AS VARIÁVEIS DE RAM DO PROGRAMA SÃO INICIALIZADAS.

```
MOVLW 0X20
MOVWF FSR                  ; APONTA O ENDEREÇAMENTO INDIRETO PARA
                            ; A PRIMEIRA POSIÇÃO DA RAM
LIMPA_RAM
CLRF INDF                  ; LIMPA A POSIÇÃO
INCF FSR,F                 ; INCREMENTA O PONTEIRO P/ A PRÓX. POS.
MOVF FSR,W
XORLW 0X80
BTFSZ STATUS,Z              ; COMPARA O PONTEIRO COM A ÚLT. POS. +1
GOTO LIMPA_RAM              ; JÁ LIMPOU TODAS AS POSIÇÕES?
                                ; NÃO - LIMPA A PRÓXIMA POSIÇÃO
                                ; SIM
```

; * CONFIGURAÇÕES INICIAIS DO DISPLAY *

; ESTA ROTINA INICIALIZA O DISPLAY P/ COMUNICAÇÃO DE 8 VIAS, DISPLAY PARA 2
; LINHAS, CURSOR APAGADO E DESLOCAMENTO DO CURSOR À DIREITA.

INICIALIZACAO_DISPLAY

```
BCF RS                     ; SELECCIONA O DISPLAY P/ COMANDOS
MOVLW 0X30
CALL ESCREVE                ; ESCREVE COMANDO 0X30 PARA
                            ; INICIALIZAÇÃO
MOVLW .3
CALL DELAY_MS               ; DELAY DE 3MS (EXIGIDO PELO DISPLAY)
MOVLW 0X30
CALL ESCREVE                ; ESCREVE COMANDO 0X30 PARA
                            ; INICIALIZAÇÃO
MOVLW 0X30
CALL ESCREVE                ; ESCREVE COMANDO 0X30 PARA
                            ; INICIALIZAÇÃO
MOVLW B'00111000'
CALL ESCREVE                ; ESCREVE COMANDO PARA
                            ; INTERFACE DE 8 VIAS DE DADOS
MOVLW B'00000001'
CALL ESCREVE                ; ESCREVE COMANDO PARA
                            ; LIMPAR TODO O DISPLAY
MOVLW .1
CALL DELAY_MS               ; DELAY DE 1MS
MOVLW B'00001100'
CALL ESCREVE                ; ESCREVE COMANDO PARA
                            ; LIGAR O DISPLAY SEM CURSOR
MOVLW B'00000110'
CALL ESCREVE                ; ESCREVE COMANDO PARA INCREM.
                            ; AUTOMÁTICO À DIREITA
```

; * ROTINA DE ESCRITA DA TELA PRINCIPAL *

; ESTA ROTINA ESCREVE A TELA PRINCIPAL DO PROGRAMA, COM AS FRASES:
; LINHA 1 - " A/D Int. (P2) "
; LINHA 2 - " Volts "

```
MOVLW 0X81
CALL ESCREVE                ; COMANDO PARA POSICIONAR O CURSOR
                            ; LINHA 0 / COLUNA 1
BSF RS                     ; SELECCIONA O DISPLAY P/ DADOS
                            ; COMANDOS PARA ESCREVER AS
                            ; LETRAS DE "A/D Int. (P2)"
MOVLW 'A'
CALL ESCREVE
MOVLW '/'
CALL ESCREVE
```

```

        MOVLW 'D'
        CALL    ESCREVE
        MOVLW "
        CALL    ESCREVE
        MOVLW 'I'
        CALL    ESCREVE
        MOVLW 'n'
        CALL    ESCREVE
        MOVLW 't'
        CALL    ESCREVE
        MOVLW '.'
        CALL    ESCREVE
        MOVLW "
        CALL    ESCREVE
        MOVLW "
        CALL    ESCREVE
        MOVLW '('
        CALL    ESCREVE
        MOVLW 'P'
        CALL    ESCREVE
        MOVLW '2'
        CALL    ESCREVE
        MOVLW ')'
        CALL    ESCREVE

        BCF    RS           ; SELECCIONA O DISPLAY P/ COMANDOS
        MOVLW 0XC7          ; COMANDO PARA POSICIONAR O CURSOR
                           ; LINHA 1 / COLUNA 7
        CALL    ESCREVE
        BSF    RS           ; SELECCIONA O DISPLAY P/ DADOS
                           ; COMANDOS PARA ESCREVER AS
                           ; LETRAS DE "Volts"

        MOVLW 'V'
        CALL    ESCREVE
        MOVLW 'o'
        CALL    ESCREVE
        MOVLW 'l'
        CALL    ESCREVE
        MOVLW 't'
        CALL    ESCREVE
        MOVLW 's'
        CALL    ESCREVE

;*****
; *          LOOP PRINCIPAL          *
;*****

; A ROTINA PRINCIPAL FICA CONVERTENDO O CANAL A/D, CALCULANDO O VALOR EM
; VOLTS E MOSTRANDO NO DISPLAY.

LOOP
        CLRWDAT          ; LIMPA WATCHDOG TIMER
        BSF    ADCON0,GO   ; INICIA CONVERSÃO A/D
        BTFSC ADCON0,GO   ; FIM DA CONVERSÃO ?
        GOTO  $-1          ; NÃO - VOLTA 1 INSTRUÇÃO
                           ; SIM
        MOVF   ADRESH,W   ; SALVA VALOR DA CONVERSÃO NO WORK
        MOVWF mulpr        ; CARREGA WORK EM mulpr

        MOVLW .50          ; CARREGA 50d EM mulnd
        MOVWF mulnd        ; CARREGA 50d EM mulnd

        CALL    mpy_F       ; CHAMA ROTINA DE MULTIPLICAÇÃO

        MOVF   H_byte,W   ; SALVA VALOR DA MULTIPLICAÇÃO
        MOVWF ACCbHI        ; EM ACCb PARA SER UTILIZADO NA
        MOVF   L_byte,W   ; ROTINA DE DIVISÃO
        MOVWF ACCbLO

        CLRF   ACCaHI      ; CARREGA ACCa COM 255d (FUNDO DE
        MOVLW .255         ; ESCALA DO CONVERSOR A/D)
        MOVWF ACCaLO        ; (ESTÃO SENDO UTILIZADOS 8 BITS)

        CALL    D_divF      ; CHAMA ROTINA DE DIVISÃO

        MOVF   ACCbLO,W   ; FAZ O AJUSTE DECIMAL PARA
        CALL    AJUSTE_DECIMAL ; MOSTRAR NO DISPLAY (LCD)

```

```

BCF      RS          ; SELECCIONA O DISPLAY P/ COMANDOS
MOVLW 0XC3          ; COMANDO PARA POSICIONAR O CURSOR
CALL    ESCREVE      ; LINHA 1 / COLUNA 3
BSF      RS          ; SELECCIONA O DISPLAY P/ DADOS

MOVF    DEZENA,W     ; CONVERTE BCD DA DEZENA EM ASCII
ADDLW 0X30          ; ENVIA AO LCD
CALL    ESCREVE

MOVLW ','
CALL    ESCREVE      ; ESCREVE UMA VIRGULA NO LCD

MOVF    UNIDADE,W    ; CONVERTE BCD DA UNIDADE EM ASCII
ADDLW 0X30          ; ENVIA AO LCD
CALL    ESCREVE

GOTO    LOOP         ; VOLTA PARA LOOP

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

END

```

Dicas e comentários

Inicialmente podemos comentar que toda a estrutura e rotinas utilizadas para a escrita no LCD são as mesmas já aplicadas no exemplo do capítulo 6.

Observe também que, conforme foi comentado anteriormente, não foi utilizada nenhuma interrupção nesse programa. Por isso, o programa permanece parado em um pequeno loop enquanto a conversão não termina. Isso é checado através do bit **ADCON0<GO/DONE>**. Outro ponto interessante é que nenhum delay é especificado antes do início da conversão. Isso não é necessário, pois o sistema analógico é lento (potenciômetro), só utilizamos um canal e, ainda por cima, o resto do loop principal dura mais de 40 μ s, garantindo o tempo mínimo entre uma conversão e outra.

A rotina de multiplicação (8 bits x 8 bits) e a de divisão (16 bits) foram obtidas dos Application Notes da própria Microchip e podem ser utilizadas em outros projetos.

Outra rotina bem interessante que aparece nesse sistema é a de conversão de um número qualquer (limitado entre 0 e 99) em dois dígitos separados, para a composição de números decimais, facilitando a escrita no LCD. Essa rotina devolve os dígitos nas variáveis UNIDADE e DEZENA. Não se esqueça de que antes de transmitir uma variável decimal para o LCD, deve-se convertê-la para ASCII, bastando para isso somar o valor 0x30 (Hex).

Para facilitar as contas e não utilizarmos números fracionários, a conversão para Volts é feita considerando-se 50 no lugar de 5,0. Na hora de enviar para o LCD, é simplesmente colocada uma vírgula entre os dois dígitos.

Exercícios propostos

Agora que o exemplo já foi estudado e esclarecido, aproveite para gerar novos problemas e soluções, seguindo os exercícios propostos:

1. Simule que a entrada analógica é um sensor de temperatura linear que deve marcar de 10 a 80° C.
2. Altere o exemplo para indicar a tensão entre 0 e 2,50V, utilizando 10 bits de resolução. Para isso, faça uso da tensão de referência externa existente na placa proposta (pino RA3).

Conversor Analógico-Digital por RC

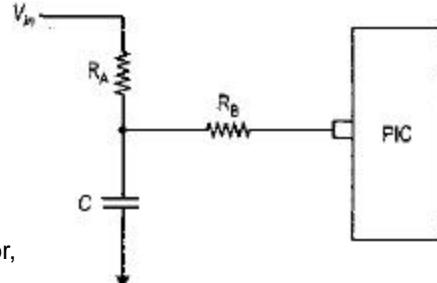
Introdução

Quando o microcontrolador não dispõe de um CAD interno é possível utilizarmos conversores externos que efetuarão a conversão e disponibilizarão o dado convertido através de uma comunicação serial ou paralela.

Mas existem alguns casos em que o custo ou os recursos do projeto não possibilitam a escolha de -n PIC com conversor interno e muito menos o uso de um conversor externo. Neste caso estamos Destinados a operar com valores meramente digitais? Nem sempre, pois existe uma solução que, apesar se não ser muito precisa, pode ser a salvação.

Teoria e recursos do PIC

Às vezes estamos interessados em quantizar apenas alguns níveis de tensão analógica, criando patamares de referenciais. Por exemplo, em um sistema alimentado à bateria é interessante sabermos se a mesma encontra-se carregada, normal ou descarregada e, portanto, acenas três níveis comparativos são necessários. O custo de um conversor A/D convencional para esse tipo de aplicação não se justifica. Nessas ocasiões pode-se recorrer a um sistema de conversão baseada na carga e descarga de um capacitor. A idéia consiste em medir, através do microcontrolador, o tempo de carga de um capacitor em um circuito RC. Veja o exemplo de hardware colocado a seguir:



Admitindo-se que o pino do PIC está configurado como entrada, o tempo de carga do capacitor C está relacionado com o valor de entrada (V_{in}), do resistor R_A e do próprio capacitor C. O resistor R_B não interfere no tempo de carga, pois o pino do PIC está em alta impedância (entrada). Já se o pino do PIC estiver configurado como saída em nível lógico 0, o capacitor tende a se descarregar pelo resistor R_B e carregar pelo resistor R_A . Porém, vamos admitir que o valor do resistor R_B seja muito menor do que o de R_A e, portanto, nesta configuração, podemos desprezar a carga proveniente do resistor R_A e admitir que o

capacitor C apenas se descarrega através de R_B . Em resumo, o capacitor se carrega através de R_A (pino como entrada) e se descarrega através de R_B (pino como saída em 0), foi que o tempo de carga/descarga depende do próprio valor do capacitor, da tensão de entrada (V_{in}) e do resistor em questão.

Como funciona então a conversão A/D?

1. O software deve configurar o pino do PIC como saída em 0.
2. Esperar o tempo de descarga do capacitor C. Este tempo deve ser garantido por software conforme os valores dos componentes utilizados.
3. Configurar o pino como entrada, ou seja, permitir a carga do capacitor.
4. Contar o tempo que o capacitor leva para que o PIC entenda nível lógico 1, ou seja, conte tempo de carga do capacitor.
5. Repetir o processo para uma nova conversão.

Como os valores do resistor e do capacitor são fixos e conhecidos, o tempo de carga do capacitor será proporcional à tensão de entrada V_{in} .

Admitindo-se que a tensão de entrada V_{in} não varia durante a conversão A/D, o modelo matemático aproximado para a curva de descarga do capacitor é

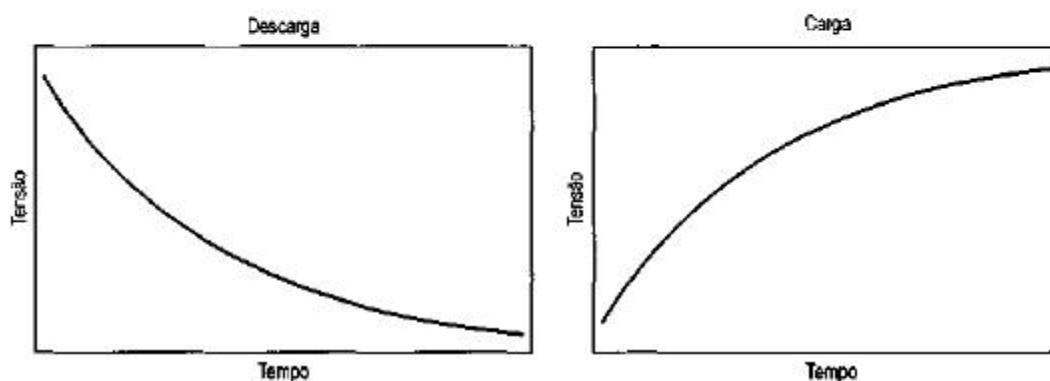
$$V_{cap}(t) = V_{in} e^{-\frac{t}{R_B C}},$$

enquanto a curva de carga segue o modelo

$$V_{cap}(t) = V_{in} \left(1 - e^{-\frac{t}{R_A C}} \right).$$

Para a curva de carga acima, temos o tempo de carga:

$$t = -R_A C \left(\ln \left(1 - \frac{V_{cap}(t)}{V_{in}} \right) \right)$$



Conforme foi comentado no início, esse tipo de conversor não apresenta uma boa precisão, além de apresentar uma série de inconvenientes:

- Nota-se nos gráficos que a tensão não varia linearmente no tempo e, portanto, esse tipo de conversor A/D não é linear.
- O valor da conversão, ou seja, o tempo de carga do capacitor está sujeito às variações dos componentes envolvidos. Para o caso do resistor, pode-se utilizar resistores de 1 %, porém para o capacitor o problema torna-se mais grave, já que geralmente existe muita variação de um lote de componentes para o outro.
- Normalmente o capacitor é muito suscetível a variações térmicas.
- A tensão de entrada deve ser suficientemente alta para que o PIC entenda nível lógico 1, por isso esse conversor não funciona no range completo de 0 a 5V.
- O valor de tensão necessário para que o PIC entenda nível lógico 1 pode variar em função da pastilha, da tensão da fonte (alimentação do PIC) e do tipo de pino (TTL/ST). Reavalie a teoria do capítulo 4.

Como dica, podemos sugerir:

- Utilizar RB pelo menos dez vezes menor que RA;
- Não utilizar capacitores maiores do que 1 μF ;
- Dê preferência ao uso de capacitores de tântalo ou cerâmico;
- Não discretizar mais do que oito níveis.

Lógica do exemplo

O exercício proposto para este capítulo mostrará no LCD o tempo de carga de um circuito RC disponível na placa do apêndice F (McLab2). Para isso, vamos colocar em prática a teoria vista posteriormente.

O pino de entrada do PIC (**RA1**) é do tipo TTL; portanto, o nível mínimo de tensão para indicar nível caco 1 é de aproximadamente 1,25V. Vamos admitir que a tensão de entrada varie entre 1,5V e 5V. A escolha de 1,5V e não 1,25V deve-se ao fato da curva do capacitor ser exponencial e, portanto, o capacitor tende a se carregar com uma tensão um pouco abaixo da tensão de entrada. Caso a tensão a; entrada fosse de 1,25V, nunca o PIC atingiria nível lógico 1.

Para o circuito da placa temos:

- $RA=4,7\text{k}\Omega$
- $RB = 330\Omega$
- $C = 1\mu\text{F}$

O tempo de carga do capacitor para uma tensão de entrada de 1,5V, considerando que o PIC entenda nível lógico 1 com uma tensão de entrada de 1,25V será então:

$$t = -(4,7k)(1\mu)\left(\ln\left(1 - \frac{(1,25)}{(1,5)}\right)\right)$$

$$t = 8,4\text{ms}$$

Já o tempo de carga do capacitor para uma tensão de entrada de 5V será:

$$t = -(4,7k)(1\mu) \left(\ln \left(1 - \frac{(1,25)}{5} \right) \right)$$

$$t = 1,4ms$$

Verifica-se, então, que para o range de tensão de entrada proposto (1,5V a 5,0V) o tempo de carga do capacitor varia entre 1,4ms e 8,4ms. Admitindo que a rotina que checa se o capacitor já está carregado têm um tempo de execução de 50 μ s e que a variável que armazenará o tempo (em múltiplos de 50 μ s) é de 8 bits (256 níveis), temos que o tempo máximo de carga do capacitor não pode ser maior do que:

$$t_{máx} = 256 * 50\mu s = 12,8ms$$

Veja que esse valor está acima dos 8,4ms calculados, o que garante que mesmo com um tempo de carga lento, o contador da rotina não irá estourar, a não ser no caso em que a tensão de entrada está abaixo da mínima admissível interpretada como nível lógico 1.

O tempo mínimo de carga do capacitor, ou seja, tensão máxima de entrada, foi calculado em 1,4ms. Como a rotina do software proposto executa uma contagem a cada 50 μ s, nesta condição, o valor mínimo medido pelo conversor A/D deve ser de:

$$A/D = \frac{1,4ms}{50\mu s} = 28d$$

Esses dados estão próximos aos reais e podem ser comprovados no software de exemplo. As diferenças encontradas podem ser explicadas levando-se em conta todos os comentários já discutidos anteriormente.

Além do tempo de carga do capacitor, devemos calcular também o tempo de descarga, uma vez que o ciclo de conversão deve ser iniciado com o capacitor descarregado. Considerando que o capacitor está totalmente carregado, ou seja, com 5V e que ele será considerado descarregado quando sua tensão for menor do que 50mV, podemos calcular o tempo máximo de descarga em:

$$t = -R_B C \ln \left(\frac{V_{cap}(t)}{V_{in}} \right)$$

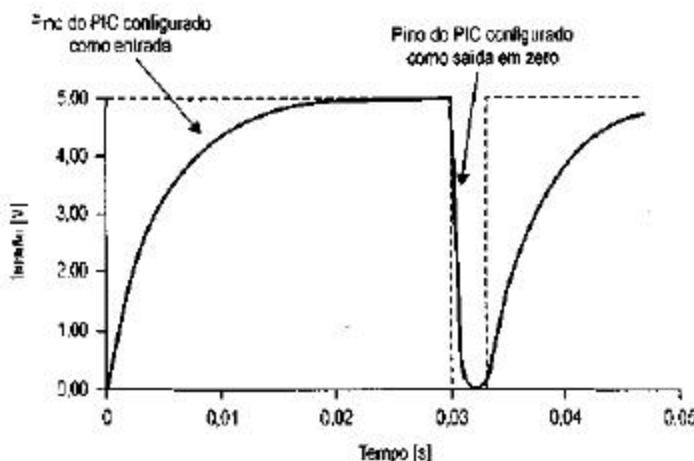
$$:= -(330)(1\mu) \ln \left(\frac{0,05}{5} \right)$$

$$t = 1,5ms$$

Na realidade, o capacitor nunca se descarregará tanto (50mV), pois o resistor RA (ligado à tensão de entrada) forma em conjunto com o resistor RB (ligado ao terra através do PIC) um divisor de tensão que não permitirá a descarga do capacitor até uma tensão tão baixa. De qualquer forma, o cálculo deve ser utilizado para estimar a ordem de grandeza do tempo de descarga do capacitor.

Para efeitos ilustrativos, segue o exemplo plotado da forma de onda no capacitor para uma tensão de entrada de 5,0V:

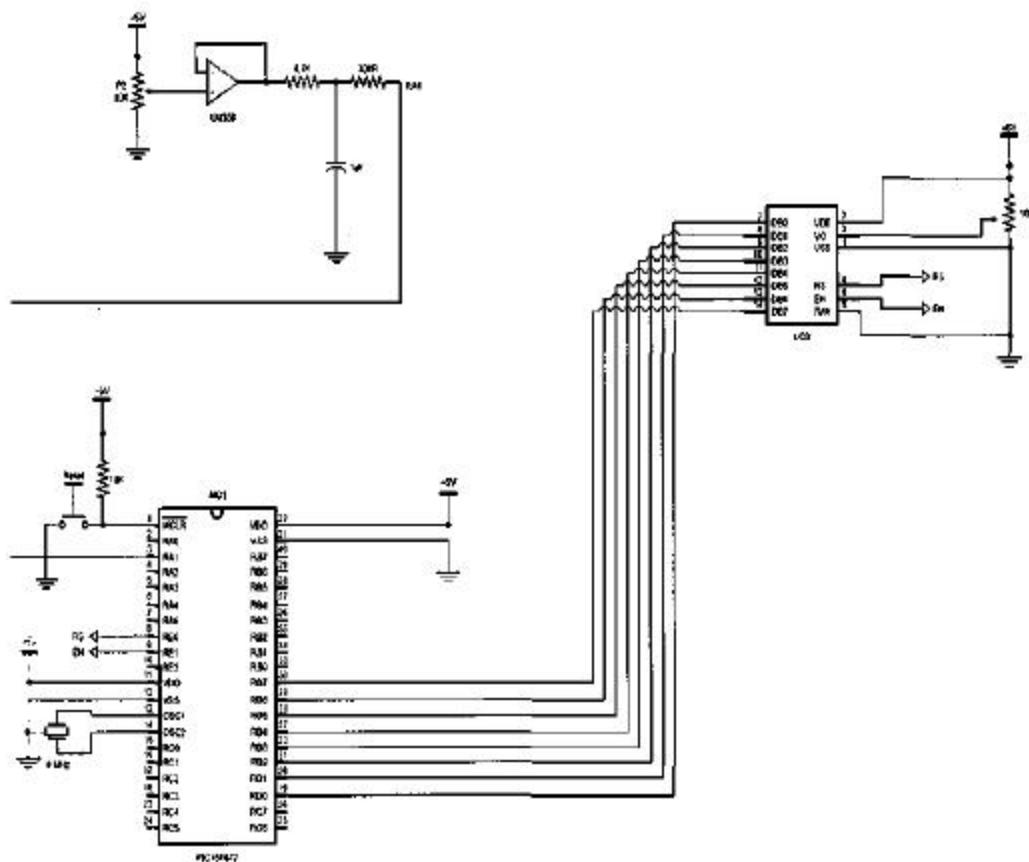
Forma de onda do capacitor



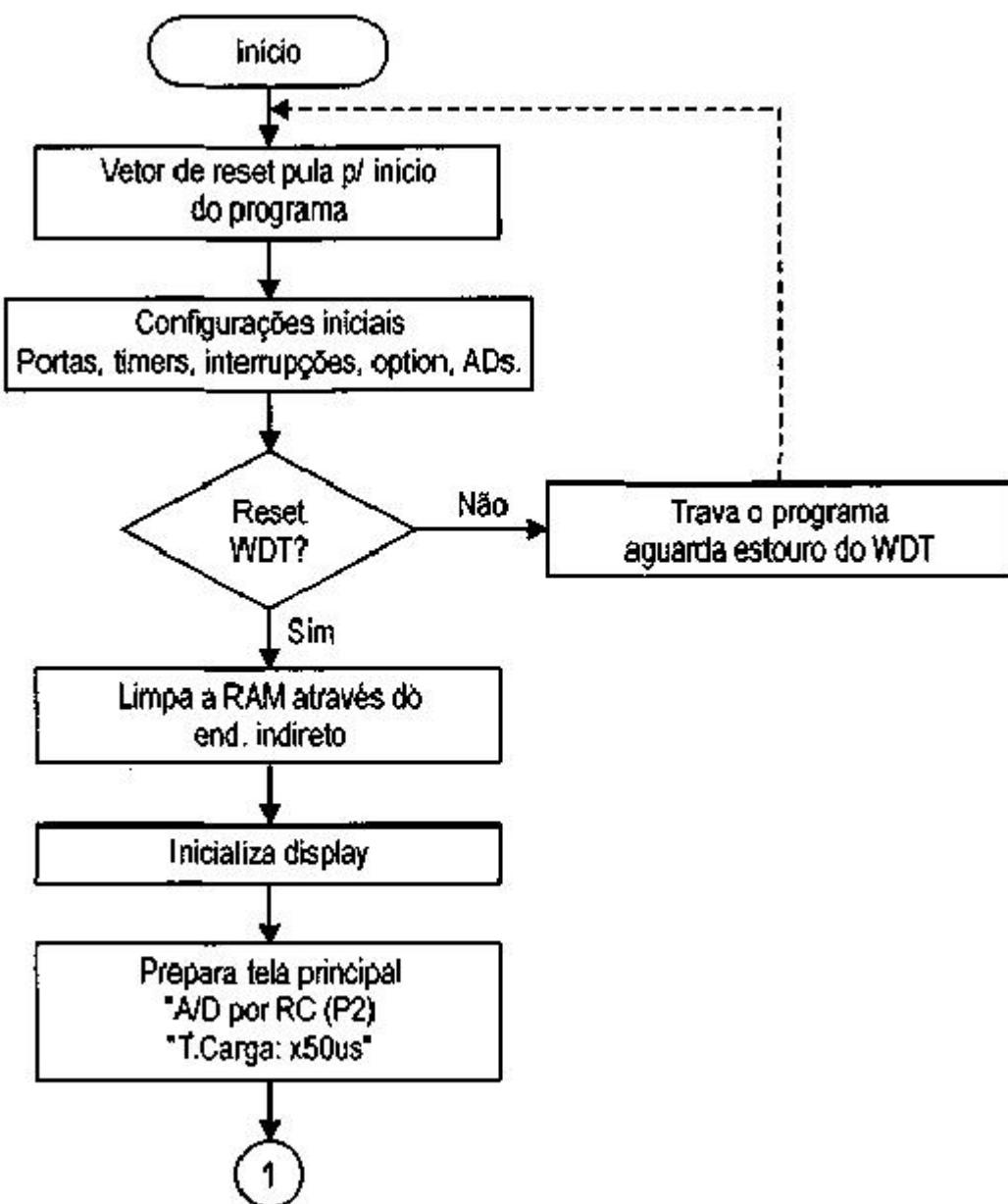
Levando-se em conta os cálculos apresentados, no software de exemplo foi criado um loop de 50us para a contagem do tempo de carga do capacitor. O valor do contador foi armazenado na variável denominada **CONTADOFLED**. Após a carga do capacitor, o valor desse contador é mostrado no LCD. Para garantir a descarga do capacitor foi deixado um delay de 3ms.

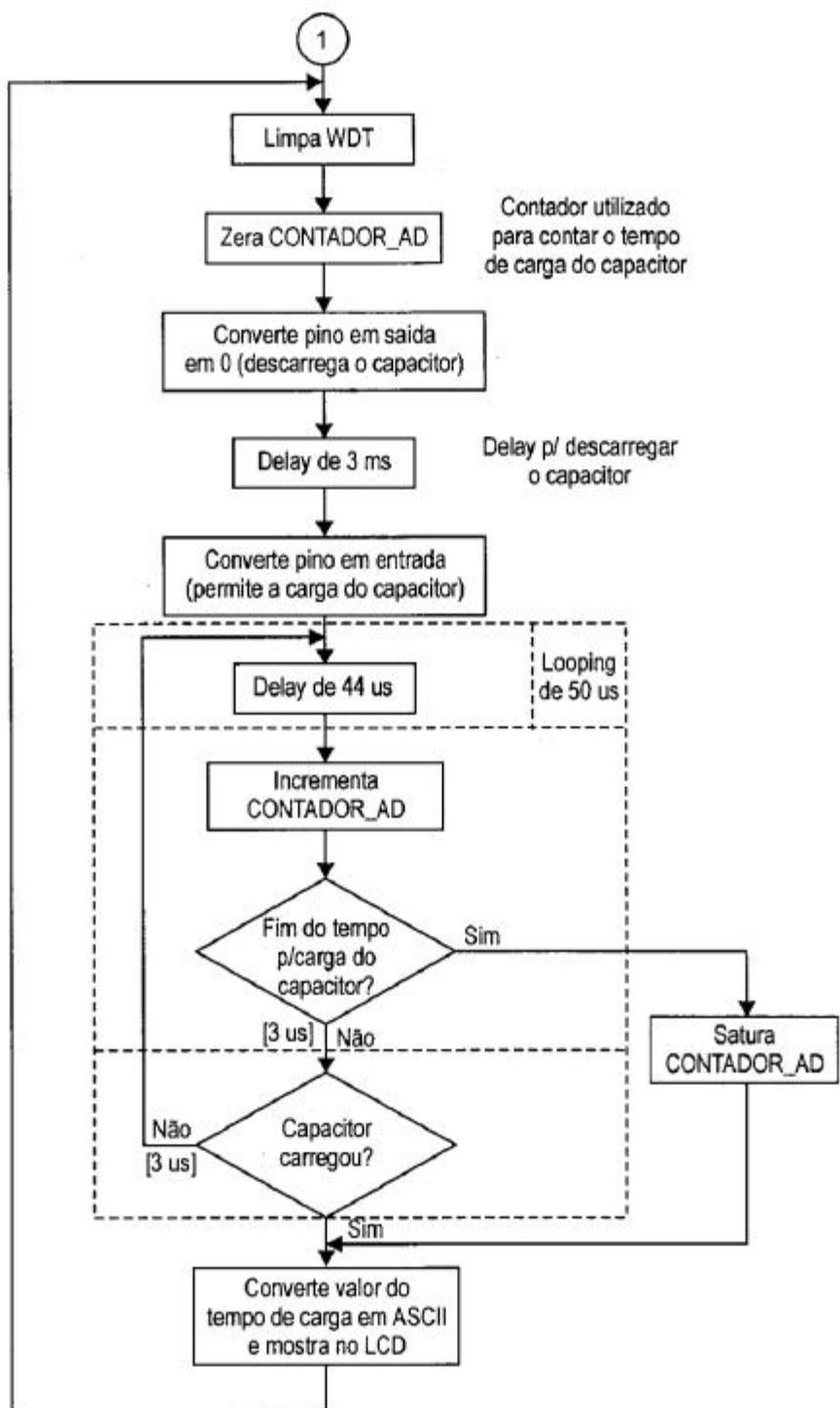
Nenhuma interrupção foi empregada neste exemplo.

Esquema elétrico



Fluxograma





```

;*****
;*      CONECTANDO O PIC - RECURSOS AVANÇADOS      *
;*      EXEMPLO 5          *
;*      NICOLÁS CÉSAR LAVINIA e DAVID JOSÉ DE SOUZA  *
;*      VERSÃO : 2.0          *
;*      DATA : 24/02/2003      *
;*****


;*****
;*      DESCRIÇÃO GERAL      *
;*****


; ESTE EXEMPLO FOI ELABORADO PARA EXPLICAR O FUNCIONAMENTO DE UM TIPO DE
; CONVERSOR A/D FUNDAMENTADO NO TEMPO DE CARGA DE UM CAPACITOR. O TEMPO DE
; CARGA DO CAPACITOR É MOSTRADO NO LCD E É INVERSAMENTE PROPORCIONAL À
; TENSÃO APLICADA ATRVÉS DO POTENCIÔMETRO P2.

;

;*****
;*      CONFIGURAÇÕES PARA GRAVAÇÃO      *
;*****


__CONFIG _CP_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &
_PWRTE_ON & _WDT_ON & _XT_OSC


;*****
;*      DEFINIÇÃO DAS VARIÁVEIS      *
;*****


; ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO LOGO NO INÍCIO DO BANCO 0

CBLOCK 0X20           ; POSIÇÃO INICIAL DA RAM

TEMPO1
TEMPO0           ; CONTADORES P/ DELAY


```

FILTRO_BOTOES ; FILTRO PARA RUIDOS DOS BOTÕES

CONTADOR_AD ; CONTADOR PARA CONVERSOR A/D

AUX ; REGISTRADOR AUXILIAR DE USO GERAL

ENDC

;*****

;* DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC *

;*****

; O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE

; OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE

; DE REDIGITAÇÃO.

#INCLUDE <P16F877A.INC> ; MICROCONTROLADOR UTILIZADO

;*****

;* DEFINIÇÃO DOS BANCOS DE RAM *

;*****

; OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR

; ENTRE OS BANCOS DE MEMÓRIA.

#DEFINE BANK1 BSF STATUS,RP0 ; SELECCIONA BANK1 DA MEMORIA RAM

#DEFINE BANK0 BCF STATUS,RP0 ; SELECCIONA BANK0 DA MEMORIA RAM

;*****

;* CONSTANTES INTERNAS *

;*****

; A DEFINIÇÃO DE CONSTANTES FACILITA A PROGRAMAÇÃO E A MANUTENÇÃO.

FILTRO_TECLA EQU .200 ; FILTRO P/ EVITAR RUIDOS DOS BOTÕES

;*****

```

;*          DECLARAÇÃO DOS FLAGs DE SOFTWARE          *
;*****;
; A DEFINIÇÃO DE FLAGs AJUDA NA PROGRAMAÇÃO E ECONOMIZA MEMÓRIA RAM.

; ESTE PROGRAMA NÃO UTILIZA NENHUM FLAG DE USUÁRIO

;*****;
;*          ENTRADAS          *
;*****;
; AS ENTRADAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#define PINO_AD      TRISA,1      ; PINO P/ LEITURA DO RC
; 0 -> FORÇA A DESCARGA DO CAPACITOR
; 1 -> LIBERA A CARGA DO CAPACITOR

#define CAD          PORTA,1      ; PINO P/ LEITURA DO CONV. A/D
; 0 -> CAPACITOR DESCARREGADO
; 1 -> CAPACITOR CARREGADO

;*****;
;*          SAÍDAS          *
;*****;
; AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#define DISPLAY      PORTD        ; BARRAMENTO DE DADOS DO DISPLAY

#define RS           PORTE,0      ; INDICA P/ DISPLAY UM DADO OU COMANDO
; 1 -> DADO
; 0 -> COMANDO

#define ENABLE       PORTE,1      ; SINAL DE ENABLE P/ DISPLAY
; ATIVO NA BORDA DE SUBIDA

```

```

;*****
;*          VETOR DE RESET DO MICROCONTROLADOR      *
;*****  

;  

; POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA  

;  

        ORG    0X000           ; ENDEREÇO DO VETOR DE RESET  

        GOTO   CONFIG          ; PULA PARA CONFIG DEVIDO A REGIÃO  

                                ; DESTINADA AS ROTINAS SEGUINTE  

;  

;*****
;  

;*          ROTINA DE DELAY (DE 1MS ATÉ 256MS)      *
;*****  

;  

; ESTA É UMA ROTINA DE DELAY VARIÁVEL, COM DURAÇÃO DE 1MS X O VALOR PASSADO  

; EM WORK (W).  

;  

DELAY_MS  

        MOVWF TEMPO1          ; CARREGA TEMPO1 (UNIDADES DE MS)  

        MOVLW .250  

        MOVWF TEMPO0          ; CARREGA TEMPO0 (P/ CONTAR 1MS)  

;  

        CLRWDT                ; LIMPA WDT (PERDE TEMPO)  

        DECFSZ TEMPO0,F       ; FIM DE TEMPO0 ?  

        GOTO    $-2             ; NÃO - VOLTA 2 INSTRUÇÕES  

                                ; SIM - PASSOU-SE 1MS  

        DECFSZ TEMPO1,F       ; FIM DE TEMPO1 ?  

        GOTO    $-6             ; NÃO - VOLTA 6 INSTRUÇÕES  

                                ; SIM  

        RETURN                ; RETORNA  

;  

;*****
;  

;*          ROTINA DE ESCRITA DE UM CARACTER NO DISPLAY      *
;*****  

;  

; ESTA ROTINA ENVIA UM CARACTER PARA O MÓDULO DE LCD. O CARACTER A SER  

; ESCRITO DEVE SER COLOCADO EM WORK (W) ANTES DE CHAMAR A ROTINA.  


```

ESCREVE

```
MOVWF DISPLAY          ; ATUALIZA DISPLAY (PORTD)
NOP                  ; PERDE 1US PARA ESTABILIZAÇÃO
BSF     ENABLE         ; ENVIA UM PULSO DE ENABLE AO DISPLAY
GOTO    $+1             ;.
BCF     ENABLE         ;.

MOVLW .1

CALL    DELAY_MS       ; DELAY DE 1MS
RETURN             ; RETORNA
```

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

CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE *

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

```
; NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS
; CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA AS
; VARIÁVEIS DE RAM E AGUARDA O ESTOURO DO WDT.
```

CONFIG

```
CLRF    PORTA          ; GARANTE AS SAÍDAS EM ZERO
CLRF    PORTB
CLRF    PORTC
CLRF    PORTD
CLRF    PORTE
```

```
BANK1              ; SELECCIONA BANCO 1 DA RAM
```

```
MOVLW B'11011111'
```

```
MOVWF TRISA        ; CONFIGURA I/O DO PORTA
```

```
MOVLW B'11111111'
```

```
MOVWF TRISB        ; CONFIGURA I/O DO PORTB
```

```
MOVLW B'11111111'
```

```
MOVWF TRISC        ; CONFIGURA I/O DO PORTC
```

Conectando o PIC 16F877A - Recursos Avançados

```

MOVLW B'00000000'
MOVWF TRISD ; CONFIGURA I/O DO PORTD

MOVLW B'00000100'
MOVWF TRISE ; CONFIGURA I/O DO PORTE

MOVLW B'11011111'
MOVWF OPTION_REG ; CONFIGURA OPTIONS
; PULL-UPS DESABILITADOS
; INTER. NA BORDA DE SUBIDA DO RB0
; TIMER0 INCREM. PELO CICLO DE MÁQUINA
; WDT - 1:128
; TIMER - 1:1

MOVLW B'00000000'
MOVWF INTCON ; CONFIGURA INTERRUPÇÕES
; DESABILITADA TODAS AS INTERRUPÇÕES

MOVLW B'00000111'
MOVWF ADCON1 ; CONFIGURA CONVERSOR A/D
; CONFIGURA PORTA COM I/O DIGITAL

BANK0 ; SELECCIONA BANCO 0 DA RAM

; AS INSTRUÇÕES A SEGUIR FAZEM COM QUE O PROGRAMA TRAVE QUANDO HOUVER UM
; RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT. DESTA FORMA,
; SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA UM ESTOURO DE WDT
; E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO START-UP DO PIC.

BTFS C STATUS,NOT_TO ; RESET POR ESTOURO DE WATCHDOG TIMER ?
GOTO $ ; NÃO - AGUARDA ESTOURO DO WDT
; SIM

```

```

;*          INICIALIZAÇÃO DA RAM      *
;*****  

;  

; ESTA ROTINA IRÁ LIMPARÁ TODA A RAM DO BANCO 0, INDO DE 0X20 A 0X7F  

;  

    MOVLW 0X20  

    MOVWF FSR           ; APONTA O ENDEREÇAMENTO INDIRETO PARA  

                        ; A PRIMEIRA POSIÇÃO DA RAM  

;  

    LIMPA_RAM  

        CLRF  INDF           ; LIMPA A POSIÇÃO  

        INCF  FSR,F          ; INCREMENTA O PONTEIRO P/ A PRÓX. POS.  

        MOVF  FSR,W  

        XORLW 0X80          ; COMPARA O PONTEIRO COM A ÚLT. POS. +1  

        BTFSS STATUS,Z       ; JÁ LIMPOU TODAS AS POSIÇÕES?  

        GOTO  LIMPA_RAM      ; NÃO - LIMPA A PRÓXIMA POSIÇÃO  

                        ; SIM  

;  

;*****  

;*          CONFIGURAÇÕES INICIAIS DO DISPLAY      *
;*****  

;  

; ESTA ROTINA INICIALIZA O DISPLAY P/ COMUNICAÇÃO DE 8 VIAS, DISPLAY PARA 2  

; LINHAS, CURSOR APAGADO E DESLOCAMENTO DO CURSOR À DIREITA.  

;  

    INICIALIZACAO_DISPLAY  

        BCF    RS           ; SELECCIONA O DISPLAY P/ COMANDOS  

;  

        MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA  

        CALL   ESCREVE       ; INICIALIZAÇÃO  

;  

        MOVLW .3  

        CALL   DELAY_MS      ; DELAY DE 3MS (EXIGIDO PELO DISPLAY)  

;  

        MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA  

        CALL   ESCREVE       ; INICIALIZAÇÃO  

;  

        MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA

```

```

CALL    ESCREVE           ; INICIALIZAÇÃO

MOVLW B'00111000'          ; ESCREVE COMANDO PARA
CALL    ESCREVE           ; INTERFACE DE 8 VIAS DE DADOS

MOVLW B'00000001'          ; ESCREVE COMANDO PARA
CALL    ESCREVE           ; LIMPAR TODO O DISPLAY

MOVLW .1

CALL    DELAY_MS           ; DELAY DE 1MS

MOVLW B'00001100'          ; ESCREVE COMANDO PARA
CALL    ESCREVE           ; LIGAR O DISPLAY SEM CURSOR

MOVLW B'00000110'          ; ESCREVE COMANDO PARA INCREM.
CALL    ESCREVE           ; AUTOMÁTICO A ESQUERDA

;*****
;*      ROTINA DE ESCRITA DA TELA PRINCIPAL      *
;*****

; ESTA ROTINA ESCREVE A TELA PRINCIPAL DO PROGRAMA, COM AS FRASES:
; LINHA 1 - "A/D por RC (P2)"
; LINHA 2 - "T.CARGA: x50us"

MOVLW 0X80                 ; COMANDO PARA POSICIONAR O CURSOR
CALL    ESCREVE           ; LINHA 0 / COLUNA 0

BSF     RS                 ; SELECCIONA O DISPLAY P/ DADOS
                            ; COMANDOS PARA ESCREVER AS
                            ; LETRAS DE "A/D por RC (P2)"

MOVLW 'A'
CALL    ESCREVE
MOVLW '/'
CALL    ESCREVE
MOVLW 'D'

```

CALL ESCREVE
MOVLW ''
CALL ESCREVE
MOVLW 'p'
CALL ESCREVE
MOVLW 'o'
CALL ESCREVE
MOVLW 'r'
CALL ESCREVE
MOVLW ''
CALL ESCREVE
MOVLW 'R'
CALL ESCREVE
MOVLW 'C'
CALL ESCREVE
MOVLW ''
CALL ESCREVE
MOVLW ''
CALL ESCREVE
MOVLW '('
CALL ESCREVE
MOVLW 'P'
CALL ESCREVE
MOVLW '2'
CALL ESCREVE
MOVLW ')'
CALL ESCREVE

BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS

MOVLW 0XC0 ; COMANDO PARA POSICIONAR O CURSOR

CALL ESCREVE ; LINHA 1 / COLUNA 0

BSF RS ; SELECCIONA O DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS
Conectando o PIC 16F877A - Recursos Avançados

; LETRAS DE "T.CARGA: x50us"

```
MOVlw 'T'
CALL    ESCREVE
MOVlw '.'
CALL    ESCREVE
MOVlw 'C'
CALL    ESCREVE
MOVlw 'A'
CALL    ESCREVE
MOVlw 'R'
CALL    ESCREVE
MOVlw 'G'
CALL    ESCREVE
MOVlw 'A'
CALL    ESCREVE
MOVlw ':'
CALL    ESCREVE
MOVlw ''
CALL    ESCREVE
MOVlw ''
CALL    ESCREVE
MOVlw 'x'
CALL    ESCREVE
MOVlw '5'
CALL    ESCREVE
MOVlw '0'
CALL    ESCREVE
MOVlw 'u'
CALL    ESCREVE
MOVlw 's'
CALL    ESCREVE
```

```

;*      ROTINA PARA DESCARREGAR O CAPACITOR DE LEITURA DO CONVERSOR A/D  *

;*****
;  

;ESTA ROTINA CONVERTE O PINO DO MICROCONTROLADOR EM SAÍDA COM NÍVEL LÓGICO 0
;E AGUARDA QUE O CAPACITOR SE DESCARREGUE. EM SEGUIDA O PINO É CONVERTIDO
;NOVAMENTE EM ENTRADA PARA PERMITIR QUE O CAPACITOR DE CARREGUE.

```

DESCARGA_CAPACITOR

```

CLRWDT           ; LIMPA WATCHDOG TIMER

CLRF  CONTADOR_AD    ; ZERA O CONTADOR DE TEMPO DE CARGA
                     ; DO CAPACITOR

BANK1            ; SELECCIONA BANCO 1 DA RAM

BCF   PINO_AD       ; TRANSFORMA PINO EM SAIDA

BANK0            ; VOLTA P/ BANCO 0 DA RAM

BCF   CAD           ; DESCARREGA O CAPACITOR

MOVLW .3

CALL  DELAY_MS      ; CHAMA ROTINA DE DELAY (3ms)
                     ; TEMPO NECESSÁRIO P/ DESCARGA
                     ; DO CAPACITOR

BANK1            ; SELECCIONA BANCO 1 DA RAM

BSF   PINO_AD       ; TRANSFORMA PINO EM ENTRADA

BANK0            ; VOLTA P/ BANCO 0 DA RAM

;*****
;  

;*      LOOP P/ ESPERAR CARGA DO CAPACITOR      *
;*****  

;  

;O TEMPO CONTA O TEMPO QUE O CAPACITOR LEVA PARA ATINGIR UM NÍVEL DE TENSÃO
;SUFICIENTE PARA QUE O MICROCONTROLADOR ENTENDA NÍVEL LÓGICO 1 NA ENTRADA TTL
;DO PINO RA1. CASO O CAPACITOR NUNCA SE DEMORE MAIS DO QUE 256 CICLOS DESTE
;LOOP, A ROTINA DESVIA PARA UMA ROTINA DE SATURAÇÃO.
;  

;O LOOP DA ROTINA É DE 50us (CRISTAL DE 4MHz).

```

LOOP_CAD

```
NOP ; [1us]

MOVLW .14 ; [2us]
MOVWF AUX ; [3us] - CARREGA AUX COM 14d
DECFSZ AUX,F
GOTO $-1 ; [4us] À [44us] - DELAY

INCFSZ CONTADOR_AD,F ; INCREM. CONTADOR E VERIFICA ESTOURO
GOTO $+2 ; NÃO HOUVE ESTOURO - PULA 1 INSTRUÇÃO
GOTO SATURACAO ; HOUVE ESTOURO - PULA P/ SATURAÇÃO

BTFS CAD ; CAPACITOR JÁ CARREGOU ?
GOTO LOOP_CAD ; NÃO - VOLTA P/ LOOP_CAD
GOTO MOSTRA_CONTADOR ; SIM - MOSTRA TEMPO DE CARGA
```

;*****

;* MOSTRA O TEMPO DE CARGA DO CAPACITOR NO LCD *

;*****

; ESTA ROTINA MOSTRA O TEMPO DE CARGA DO CAPACITOR EM HAXADECIMAL NO LCD.

; CASO O CAPACITOR NÃO TENHA SE CARREGADO, A ROTINA DE SATURAÇÃO GARANTE

; UM VALOR MÁXIMO PARA O TEMPO DE CARGA (0xFF).

SATURACAO

```
MOVLW 0XFF
MOVWF CONTADOR_AD ; SATURA O CONTADOR
; (CAPACITOR NÃO CARREGOU)
```

MOSTRA_CONTADOR

```
MOVLW 0XC9 ; COMANDO PARA POSICIONAR O CURSOR
CALL ESCREVE ; LINHA 1 / COLUNA 9

BSF RS ; SELECCIONA O DISPLAY P/ DADOS
```

```
SWAPF CONTADOR_AD,W ; INVERTE NIBLE DO CONTADOR_AD
```

Conectando o PIC 16F877A - Recursos Avançados

```

ANDLW B'00001111'          ; MASCARA BITS MAIS SIGNIFICATIVOS
MOVWF AUX                   ; SALVA EM AUXILIAR

MOVLW 0X0A

SUBWF AUX,W                ; AUX - 10d (ATUALIZA FLAG DE CARRY)
MOVLW 0X30                  ; CARREGA WORK COM 30h
BTFS C STATUS,C             ; RESULTADO E POSITIVO? (É UMA LETRA?)
MOVLW 0X37                  ; SIM - CARREGA WORK COM 37h
                                ; NÃO - WORK FICA COM 30h (NÚMERO)
ADDWF AUX,W                 ; SOMA O WORK AO AUXILIAR
                                ; (CONVERSÃO ASCII)
CALL    ESCREVE              ; MOSTRA NO DISPLAY

MOVF   CONTADOR_AD,W        ; CARREGA NO WORK O CONTADOR_AD
ANDLW B'00001111'          ; MASCARA BITS MAIS SIGNIFICATIVOS
MOVWF AUX                   ; SALVA EM AUXILIAR

MOVLW 0X0A

SUBWF AUX,W                ; AUX - 10d (ATUALIZA FLAG DE CARRY)
MOVLW 0X30                  ; CARREGA WORK COM 30h
BTFS C STATUS,C             ; RESULTADO E POSITIVO? (É UMA LETRA?)
MOVLW 0X37                  ; SIM - CARREGA WORK COM 37h
                                ; NÃO - WORK FICA COM 30h (NÚMERO)
ADDWF AUX,W                 ; SOMA O WORK AO AUXILIAR
                                ; (CONVERSÃO ASCII)
CALL    ESCREVE              ; MOSTRA NO DISPLAY
BCF    RS                    ; SELECCIONA O DISPLAY P/ COMANDOS

GOTO    DESCARGA_CAPACITOR ; VOLTA P/ DESCARREGAR O CAPACITOR
;*****
;*          FIM DO PROGRAMA          *
;*****


END                      ; FIM DO PROGRAMA

```

Dicas e comentários

Observe que foi feita uma verificação para garantir que, se o tempo de carga do capacitor ultrapassar o limite da variável CONTADOR_AD, essa variável terá seu valor assegurado em 0xFF através da rotina SATURAÇÃO.

Toda a estrutura e rotinas para escrita no LCD são as mesmas utilizadas no exemplo do capítulo 6.

Exercícios propostos

Agora que o exemplo já foi estudado e esclarecido, aproveite para gerar novos problemas e soluções, seguindo os exercícios propostos:

1. Seguindo o exemplo dado no começo do capítulo em relação ao monitoramento de uma bateria, estabeleça três níveis de comparação e altere a tela do LCD para mostrar as mensagens de bateria Fraca, Normal e Carregada.
2. A placa proposta (Apêndice G) possui um outro RC ligado ao PIC, só que com a tensão constante e o R variável. Trata-se dos jumpers A, B e C. A combinação desses jumpers cria sete níveis de resistência aplicadas ao RC que está ligado no pino RA4. Altere o programa para ter esses jumps e indicar seus estados no LCD. Os detalhes do esquema de ligação e valores dos componentes podem ser encontrados no Apêndice G.

Os Módulos CCP (Capture/Compare/PWM)

Introdução

Vamos estudar agora os módulos denominados CCPs, cujo nome é originado dos três tipos de recursos oferecidos por eles: Capture, Compare e PWM. Cada um desses recursos é empregado para uma finalidade diferente, que será conhecida a partir de agora.

Teoria e recursos do PIC

O PIC 16F877A possui dois módulos de CCP, denominados CCP1 e CCP2. Esses módulos são praticamente idênticos e, por isso, serão explicados ao mesmo tempo, sendo feito os comentários necessários quando houver algum tipo de diferença. Para facilitar o entendimento, cada um dos recursos (Capture, Compare e PWM) será descrito separadamente.

Para começar, entretanto, é bom esclarecermos a nomenclatura padrão que será utilizada para descrevermos características comuns aos dois módulos existentes:

Padrão	CCP1	CCP2 ;	Descrição
CCPxCON	CCP1CON	CCP2CON	Registrador de configuração
CCPRxH	CCPR1H	CCPR2H	Parte alta do valor de controle
CCPRxL	CCPR1L	CCPR2L	Parte baixa do valor de controle
CCPx	CCP1 (RC2)	CCP2 (RC1)	Pino relacionado

Outro dado interessante que devemos informar neste momento é quanto ao uso dos dois módulos conjuntamente. Como eles utilizam recursos compartilhados para suas bases de tempo (Timer 1 e Timer 2), podem existir algumas limitações ou conflitos, conforme a combinação de recursos desejada:

Modo	Base de tempo
Capture	Timer 1
Compare	Timer 1
PWM	Timer 2

Recursos Desejados		Observações
Capture	Capture	Sem conflitos, entretanto, ambos utilizarão a mesma base de tempo TMR1 e, por isso, serão sincronizados.
Capture	Compare	Caso o Compare esteja configurado para zerar o Timer 1 , isso poderá acarretar em um conflito com o outro modo.
Compare	Compare	Caso o Compare esteja configurado para zerar o Timer 1 , isso poderá acarretar em um conflito com o outro modo.
PWM	PWM	Ambos os PWM terão a mesma freqüência e serão sincronizados, devido ao uso da mesma base de tempo. Os <i>Duty Cycles</i> possuem controles independentes.
PWM	Capture	Ambos os modos são completamente independentes.
PWM	Compare	Ambos os modos são completamente independentes.

Modo Capture

Este módulo tem como objetivo a contagem de tempo entre dois eventos ocorridos em um dos pinos CCPx. Para isso será utilizado como base de tempo o Timer 1 e no momento do evento seu valor será capturado (dai o nome Capture). Como o Timer 1 é de 16 bits, a captura será feita em dois registradores: **CCPRxH** e **CCPRxL**.

Com esse recurso é possível então criarmos um periodímetro, contando o tempo gasto, por exemplo, entre duas bordas de subida da onda ligada ao pino **CCPx**. É importante observarmos, entretanto, que a captura do valor de Timer 1 não reseta este timer, e por isso, para definirmos o tempo real entre duas leituras será necessário uma conta de subtração entre a última leitura e a anterior. Esta conta deverá ser implementada pelo software.

Vejamos, então, como configurar o Capture para que possamos utilizá-lo corretamente.

O modo Capture opera com os pinos como entrada, mas essa configuração não é feita automaticamente. Por isso, antes de mais nada configure, através do **TRISC**, o pino **CCPx** como entrada. Caso este esteja configurado como saída, operações de escrita neste pino podem ser consideradas como mudança de borda, ativando a captura.

Quanto ao Timer 1, que será usado como base de tempo para este modo, ele não pode estar configurado para operação assíncrona, isto é, confirme a condição **T1CON<T1SYNC>=0**.

O Capture possui também quatro diferentes configurações (fora o desligamento), que podem ser escolhidas através de **CCPxCON<CCPxM3:CCPxM0>**:

CCPxM3: CCPxM0	Descrição
0000	CCPx desligado
0100	Capture ligado para borda de descida Prescale de 1:1
0101	Capture ligado para borda de subida Prescale de 1:1
0110	Capture ligado para borda de subida Prescale de 1 :4
0111	Capture ligado para borda de subida Prescale de 1:16

Obs. As demais opções de configuração dizem respeito aos outros modos (Compara/PWM).

As diferenças básicas entre essas configurações dizem respeito à borda que gerará o evento e ao prescaler adotado. Esse prescaler é um contador interno (não acessível) de bordas. Por exemplo: caso seja escolhida a opção 0111, a captura do TMR1 acontecerá a cada 16 bordas de subida. A finalidade desse prescaler é o aumento da precisão do sistema. “Quando optamos em trabalhar com prescaler de 1:1, a erro máximo em um período é de um ciclo de máquina (T_{CY}). Quando aumentamos o prescaler, esse erro diminui proporcionalmente, e teremos para 1:16 um erro máximo de $T_{CY}/16$.

Toda vez que o evento de Capture acontecer, o flag **CCPxIF** será ativado, e caso essa interrupção esteja ligada, ela irá acontecer. Esta é uma maneira fácil de implementarmos a conta de subtração dos tempos absolutos para chegarmos ao período correto. Não se esqueça de que, quando uma nova captura acontece, ela será gravada em **CCPRxH** e **CCPRxL**, sobrepondo os valores anteriores.

Para alterar entre as opções de configuração do modo, alguns cuidados devem ser tomados. Como o prescaler é um contador interno que não pode ser zerado manualmente, a alteração de configuração cederá gerar uma interrupção. Uma maneira de evitarmos isso seria o desligamento da interrupção **CCPxIE** antes dessa operação. Uma outra maneira, mais prática, é desligarmos o modo Capture tirando o registrador **CCPxCON**. Isso irá resetar o modo, limpando também o contador de prescaler. Depois, basta escolhermos a nova configuração, carregando **CCPxCON** com o valor desejado.

Quanto ao funcionamento do Capture em modo SLEEP, a história é um pouco confusa, pois nessa situação ou o Timer 1 não está funcionando ou está em modo assíncrono. Por isso, a interrupção de 3CP pode até acontecer, acordando o PIC, mas os registradores **CCPRxH** e **CCPRxL** não serão atualizados.

Resumo dos registradores associados ao Capture

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
0Ch	PIR1	PSP1IF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
8Ch	PIE1	PSP1IE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
0Dh	PIR2	-	-	-	EEIF	BCLIF	-	-	CCP2IF
8Dh	PIE2-	-	-	-	EEIE	BCLIE	-	-	CCP2IE
17h	CCP1CON	-	-	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
16h	CCPR1H	Captura de TMR1H (Parte alta)							
15h	CCPR1L	Captura de TMR1H (Parte baixa)							
1Dh	CCP2CON	-	-	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0
1Ch	CCPR2H	Captura de TMR1H (Parte alta)							
1Bh	CCPR2L	Captura de TMR1L (Parte baixa)							
87h	TRISC	Configuração do PORTC com Entrada(1) ou Saída(0)							

Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Modo Compare

Enquanto no modo Capture o valor de Timer 1 era capturado e armazenado, aqui ele será constantemente comparado (olha a origem do nome novamente) com os valores dos registradores **CCPRxH** e **CCPRxL**. Sempre que essa comparação de 16 bits resultar numa igualdade, o flag **CCPxIF** será ativado e a interrupção poderá acontecer, caso a mesma esteja devidamente ligada. Além disso, se desejado, podemos alterar automaticamente o estado do pino CCPx.

Aqui também vale a observação quanto ao funcionamento do Timer 1, que deve obrigatoriamente estar ajustado em modo síncrono (**T1CON<T1SYNC>=0**). Não esqueça também de configurar o pino **CCPx** como saída, através do **TRISC**.

A ativação do Compare e as opções para mudança no pino podem ser configuradas em **CCPxCON<CCPxM3:CCPxM0>**:

CCPxM3: CCPxM0	Descrição
0000	CCPx desligado
1000	Inicia com o pino em 0 (baixo) e altera para 1 (alto) quando a comparação
1001	Inicia com o pino em 1 (alto) e altera para 0 (baixo) quando a comparação
1010	Não altera o pino.
1011	Não altera o pino, mas reseta TMR1 .

Obs: As demais opções de configuração dizem respeitos aos outros modos (Compare/PWM).

Em todas as opções, o flag da interrupção sempre será ativado.

A última opção (1011) é chamada no Data Sheet do PIC de Special Event Trigger, e apesar dela não alterar o estado do pino **CCPx**, uma outra alteração muito importante acontece. Os registradores do Timer 1 (**TMR1H** e **TMR1L**) são zerados. Com isso podemos utilizar o Timer 1 de forma similar ao Timer 2 com o registrador de limite PR2. A vantagem aqui é que estamos trabalhando com 16 bits, e não mais com 8. Este é o único caso de diferença entre CCP1 e CCP2, pois CCP2 além de resetar o **TMR1**, irá também iniciar uma conversão analógica (**ADCON0<GO/DONE>**) se o conversor estiver ligado. Atenção ao escolher essa configuração quando os dois modos CCP1 e CCP2 estiverem em uso, pois tanto o Compare quanto o Capture utilizam o Timer 1 como base de tempo.

No modo SLEEP, ou Timer 1, está em modo assíncrono ou está paralisado. Em nenhum dos dois casos o Compare poderá funcionar. Por isso, este modo não opera durante o SLEEP. Só não se esqueça de que a saída **CCPx** é um pino como outro qualquer e, por isso, o nível de tensão da mesma será garantido mesmo em SLEEP.

Resumo dos registradores associados ao Compare

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0Bh...	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
OCh	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
ODh	PIR2	-	-	-	EEIF	BCLIF	-	-	CCP2IF
8Dh	PIE2	-	-	-	EEIE	BCLIE	-	-	CCP2IE
17h	CCP1CON	-	-	DC1B1	DC1B0	CCPM3	CCPM2	CCPM1	CCPM0
16	CCPR1H	Comparação com TMR1H (Parte alta)							
15h	CCPR1L	Comparação com TMR1L (Parte baixa)							
1Dh	CCP2CON	-	-	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0
1Ch	CCPR2H	Comparação com TMR1H Parte alta)							
1Bh	CCPR2L	Comparação com TMR1L (Parte baixa)							
87h	TRISC	Configuração do PORTC com Entrada(1) ou Saída(0)							

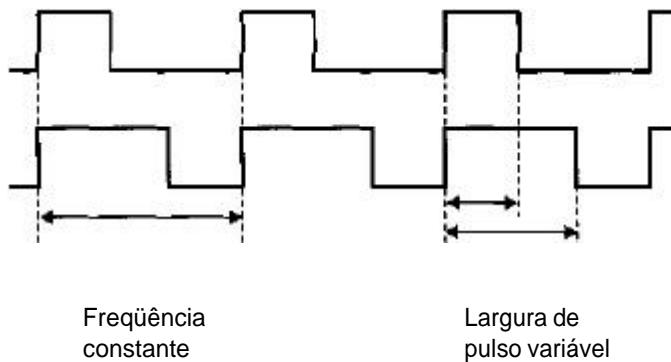
—
Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Modo PWM

O Modo PWM é provavelmente o recurso mais poderoso dos módulos CCPs, pois com ele podemos obter uma tensão analógica a partir de um sinal digital. Na verdade, esta saída é meramente digital, isto é, somente pode assumir os estados 0 e 1. Porém, pelo conceito aplicado ao PWM, podemos transformá-la em uma tensão variável. Obviamente isso exigirá um hardware complementar (filtros) depois da saída do PIC, mas isso é uma outra história. Por enquanto, vamos entender melhor o que é um PWM.

O nome PWM tem sua origem no inglês Pulse Width Modulation que em Português pode ser considerado como Modulação por Largura de Pulso. Mas o que exatamente significa isso? Trata-se de uma onda com freqüência constante (período fixo) e largura de pulso (duty cycle) variável.

Na figura seguinte temos a representação de duas formas de onda tipo PWM, cada uma delas com uma largura de pulso diferente:



Esse tipo de sinal é particularmente importante, já que a partir dele é possível implementar um conversor digital-analógico com um único pino do microcontrolador, uma vez que controlando a largura do pulso é possível obter uma tensão analógica variável.

Vejamos a teoria. Genericamente, a tensão média de uma forma de onda é dada por:

$$V_{dc} = \frac{1}{T} \int_0^T V(t) dt$$

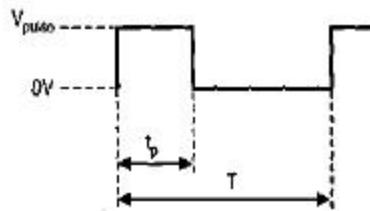
Onde T é o período da forma de onda e V(t) é a função da tensão no tempo.
Para o caso do PWM temos que:

$$V(t) = \begin{cases} V_{pulso} & \rightarrow 0 \leq t \leq t_p \\ 0 & \rightarrow t_p < t \leq T \end{cases}$$

Onde: T_p é a duração do pulso em nível lógico 1

V_{pulso} é a tensão de pulso do sinal PWM.

Então



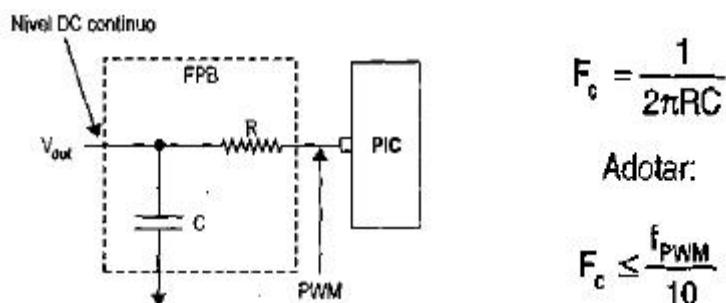
$$V_{dc} = \frac{1}{T} \left(\int_0^{t_p} V_{pulso} dt + \int_{t_p}^T 0 dt \right)$$

$$V_{dc} = \frac{t_p}{T} V_{pulso}$$

A razão entre a largura de pulso e o período da forma de onda recebe o nome de duty cycle, ou em português, ciclo ativo. O pulso da onda PWM apresenta tensão fixa, porém o valor médio da tensão tá forma de onda varia em função do duty cycle. A tensão média (V_{dc}) é diretamente proporcional ao duty cycle e como este varia entre 0 (quando $t_p = 0$) e 1 (quando $t_p = T$) temos que a tensão média dela pode variar entre 0 e V_{pulso} . No nosso caso a variação será de V_{ss} a V_{DD} , ou seja, de 0 a 5V.

Assim, para obtermos um conversor digital analógico a partir do pino **CCPx**, basta implementar um sinal tipo PWM e adicionar à saída um filtro que passa baixa freqüência de corte menor que a própria freqüência do PWM.

Cálculo da freqüência de corte do filtro (fc):



Quando não é necessário obter uma tensão média contínua, a implementação do filtro é descartada, no nos casos da placa proposta. Tanto o resistor de aquecimento quanto o ventilador trabalham com VMs sem filtro, pois a função desses componentes faz com que eles atuem como filtros, desde que a freqüência do PWM não seja muito baixa.

Assim sendo, a teoria continua válida, o que significa que podemos, através do PWM, regular a taxa i aquecimento do resistor e a velocidade do ventilador, variando a tensão média aplicada a eles.

Vamos aprender agora um pouco mais sobre o funcionamento dos PWMs dentro do 16F877A.

Esse PIC possui dois canais de PWMs (CCP1 e CCP2), cada um com resolução máxima de dez bits. Isso significa que nosso duty cycle poderá ser regulado de 0 a 100% com uma resolução máxima de 1024 pontos. No entanto, dependendo da configuração adotada, essa resolução não será atingida.

Vamos estudar como os tempos do PWM (pulso e período) são controlados internamente para podermos entender melhor esse problema.

O período do PWM (T) é controlado diretamente pelo Timer 2, através do registrador **PR2**. Como já foi visto no capítulo 4, sempre que **TMR2 = PR2**, o timer é zerado. Neste momento, um novo período do PWM é iniciado. Desta forma, podemos definir o período e a freqüência do PWM pelas seguintes formulas:

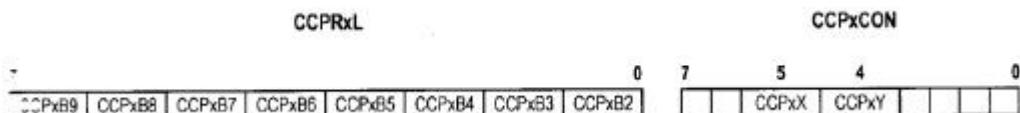
$$T = [(PR2) + 1] \times 4 \times T_{OSC} \times (\text{Prescale do TMR2})$$

$$\text{PWM}_{\text{Freq}} = 1 / T$$

Tudo bem quanto ao período, mas como definimos o duty cycle? Na realidade, no PIC não definimos : valor do duty cycle e sim o tempo do pulso em nível alto. Desta forma, o tempo do pulso pode ser calculado por:

$$t_p = \text{CCPRxL} : \text{CCPxCON} < \text{CCPxX} : \text{CCPxY} > \times T_{OSC} \times (\text{Prescale do TMR2})$$

Repare que a largura do pulso é ajustada em dois registradores: **CCPRxL** que armazena os 8 bits mais significativos, e **CCPxCON**, que armazena os dois bits menos significativos. Assim, temos os 10 bits que controlam o duty cycle do PWM alocados da seguinte maneira:



Para ficarmos de acordo com a teoria, calcularemos efetivamente o duty cycle dividindo o tempo do pulso em nível alto pelo período total do PWM.

$$\frac{t_p}{T} = \frac{\text{CCPRxL} : \text{CCPxCON} < \text{DCxB1} : \text{xB0} > \times T_{OSC} \times (\text{Prescale do TMR2})}{[(PR2) + 1] \times 4 \times T_{OSC} \times (\text{Prescale do TMR2})}$$

$$\frac{t_p}{T} = \frac{\text{CCPRxL} : \text{CCPxCON} < \text{DCxB1} : \text{dc} \times \text{b0} >}{[(PR2) + 1] \times 4}$$

Verifica-se então que apesar do período e o do tempo de pulso dependerem do cristal (T_{osc}) e do ajuste do prescaler do Timer 2, o duty cycle depende única e exclusivamente dos valores ajustados nos registradores **PR2**, **CCPRxL** e **CCPxCON** (bits 5 e 4).

Veja que o registrador **PR2** (8 bits) que controla o período do PWM é multiplicado por quatro para poder igualar-se aos 10 bits que controlam o duty cycle. É justamente esse o problema da resolução máxima atingida. Se o registrador **PR2** for ajustado com um valor menor que 8 bits, ou seja, menor do que 255, serão necessários menos do que 10 bits para atingir um PWM com 100% de duty cycle. Portanto, o número de pontos para ajuste do duty cycle é quatro vezes maior do que o valor ajustado em **(PR2+1)**. Em termos de bits, podemos dizer que a resolução do duty cycle é 2 bits maior do que o

número de bits que formam o valor ajustado em **PR2**. Repare também que, caso **PR2** seja ajustado com 255, nunca será atingido um duty cycle de 100%, pois o período atingirá o valor máximo de 1024 ($[PR2+1] \times 4$), enquanto o tempo do pulso em nível alto (**<DCxB9:DCxB0>**) será no máximo 1023.

É fácil notar também que a resolução para o ajuste do período depende do prescaíer do Timer 2. assim:

Prescale	Tempo do menor passo (resolução)
1	T_{osc}
4	$4T_{osc}$ ou T_{cy}
16	$16T_{osc}$ ou $4T_{cy}$

Porém, de qualquer forma, a menor resolução para o tempo do pulso (duty cycle) será sempre quatro vezes menor que a do período.

Note também que o postscale do Timer 2 não é utilizado para a construção dos tempos envolvidos no PWM.

Uma forma de calcular a quantidade máxima de bits que define a quantidade máxima de passos do nosso PWM é:

$$\frac{\log\left(\frac{F_{osc}}{F_{PWM}}\right)}{\log(2)}$$

Vamos a um exemplo prático.

Calculemos os valores para um PWM de 78,125 kHz, como um PIC rodando a 20 MHz e ajuste do prescaler do Timer2 em 1:1.

$$1 / 78,125 \text{ kHz} = [(PR2) + 1] \times 4 \times (1 / 20 \text{ MHz}) \times 1$$

$$12,8 \mu\text{s} = [(PR2) + 1] \times 4 \times 50 \text{ ns} \times 1$$

$$\mathbf{PR2 = 63}$$

Partimos agora para a conta da resolução:

$$1 / 78,125 \text{ kHz} = 2^{\text{PWMResolução}} \times (1 / 20 \text{ MHz}) \times 1$$

$$12,8 \mu\text{s} = 2^{\text{PWMResolução}} \times 50 \text{ ns} \times 1$$

$$2^{\text{PWMResolução}} = 256$$

$$\text{PWMResolução} \times \log(2) = \log(256)$$

$$\text{PWMResolução} = 8 \text{ (bits)}$$

Quanto à operação prática do PWM, já ficou claro que antes de mais nada é necessário definirmos a freqüência de trabalho, com base na freqüência de funcionamento do PIC e na resolução desejada. Com isso calculamos o valor para ser colocado em **PR2**.

Depois devemos configurar o pino **CCPx** para ser utilizado como saída. Essa configuração não é automática e deve ser feita através do **TRISC**. Em seguida, devemos calcular a largura de pulso 3 desejada.: O resultado deve ser armazenado em dois registradores, sendo os 8 bits mais significativos ir **CCPRxL** e os outros dois bits restantes em **CCPxCON<DCxB1:DCxB0>**.

Para o PWM, o registrador **CCPRxH** é do tipo somente leitura e ele é utilizado pelo sistema para armazenar uma cópia do **CCPRxL**. Essa cópia é utilizada para possibilitar que a largura de pulso seja superada durante o funcionamento do PWM. Assim sendo, a nova largura será adotada automaticamente ·c próximo período do PWM. Os dois bits adicionais também são armazenados internamente pelo sistema.

Quando **TMR2 = PR2**, as seguintes ações irão acontecer:

- **TMR2 = 0**, iniciando o próximo período;
- O pino **CCPx** é colocado em 1 (alto), a menos que a largura do pulso esteja definida para 0 (zero);
- O valor do registrador **CCPRxL** é copiado para **CCPRxH**, atualizando a largura de pulso.

O sistema passa, então, a monitorar o término do pulso, quando **TMR2 = CCPRxH**. Essa comparação irá considerar ainda os 2 bits menos significativos. Neste momento, a saída **CCPx** será colocada 0 (baixo), até que um novo período comece. Caso o tamanho do pulso seja especificado como sendo maior que o período total, a saída **CCPx** nunca será colocada em 0 (baixo), mas o sistema funcionará normalmente, como ajustado para 100% do PWM.

Para que, finalmente, a saída comece a operar, é necessário ainda ajustar o prescaler do Timer 2 e liga-lo, através do registrador **T2CON**. Por último, ligue também o módulo de PWM, através dos bits **CCPxCON<CCPxM3:CCPxM0>**:

CCPxM3:CCPxM0	Descrição
0000	CCPx desligado.
11XX	Ativa a saída PWM.

Os PWMs não funcionam em modo SLEEP nem geram interrupções; porém, não se esqueça de que a interrupção de Timer2 pode continuar acontecendo.

Resumo dos registradores associados ao PWM

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
17h	CCP1CON	-	-	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0
16h	CCPR1H	Cópia de CCPR1L (somente leitura)							
15h	CCPR1L	Largura do pulso, bits de 9 a 2 (Parte baixa)							
1Dh	CCP2CON	-	-	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0
1Ch	CCPPR2H	Cópia de CCPPR2L (somente leitura)							
1Bh	CCPR2L	Largura do pulso, bits de 9 a 2 (Parte baixa)							
87h	TRISC	Configuração do PORTC com Entrada(1) ou Saída(0)							

—
Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Lógica do exemplo

Infelizmente não é possível construirmos um exemplo que utilize todos os modos do CCP ao mesmo tempo. Por isso, optamos pela criação de um único exemplo que trabalhe com o recurso que nós consideramos o mais importante: o PWM.

Neste exemplo ativaremos a saída do módulo CCP2 para podermos controlar a rotação do ventilador que está ligado ao pino RC1.

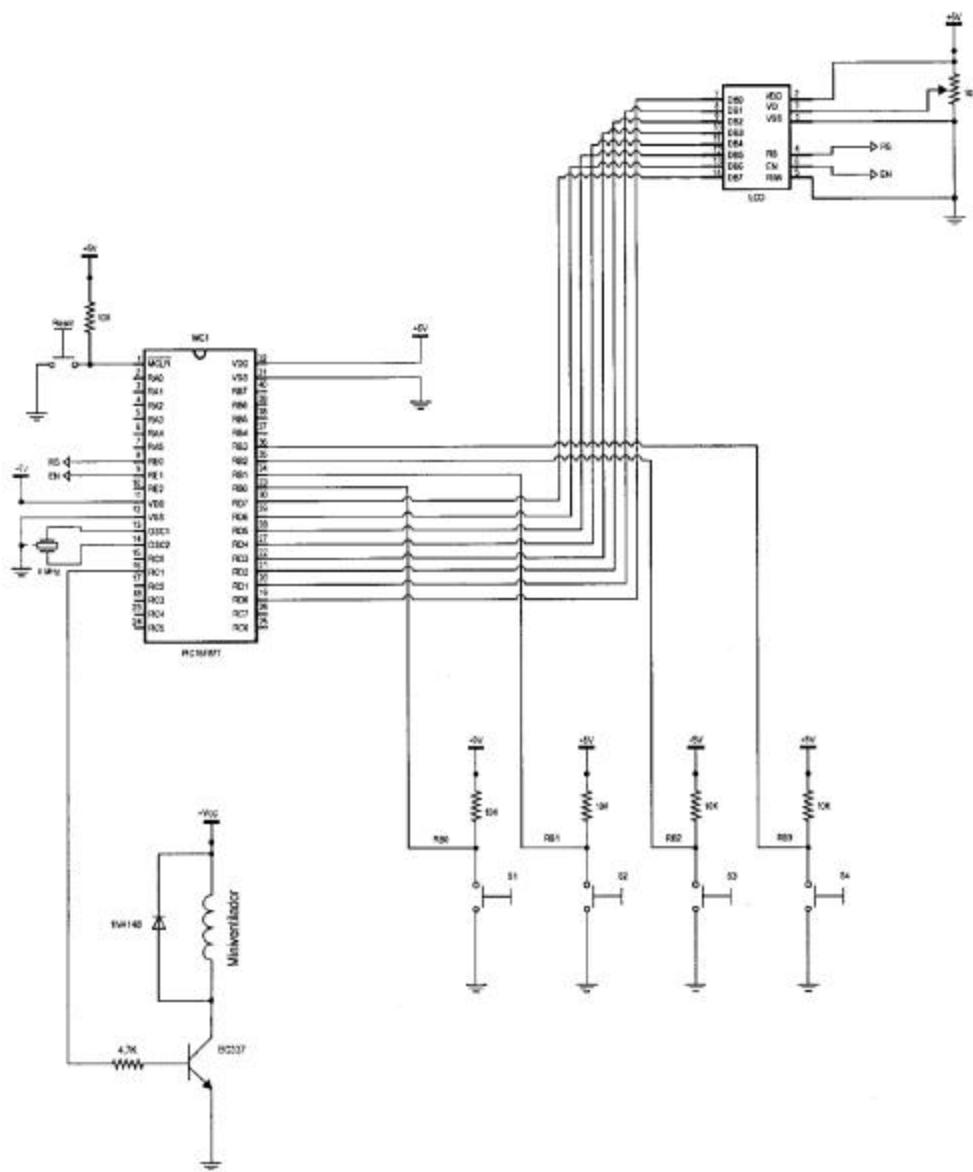
Optamos por trabalhar com PR2 no valor máximo (255) e o prescaler em 16. Com isso a freqüência do PWM será de 244,14 Hz ($\text{PWM}_{\text{período}} = 4,096\text{ms}$), considerando-se que na placa proposta o PIC está rodando a 4 MHz. Esta conta foi feita utilizando-se a fórmula dada na aula teórica.

Para ajuste do duty cycle, optamos pela implementação de quatro botões:

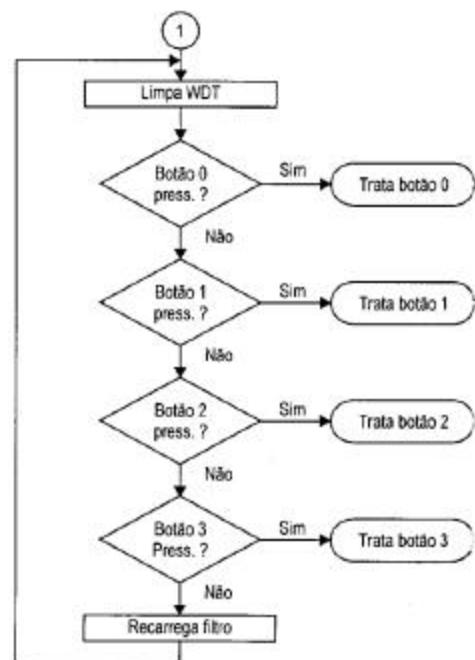
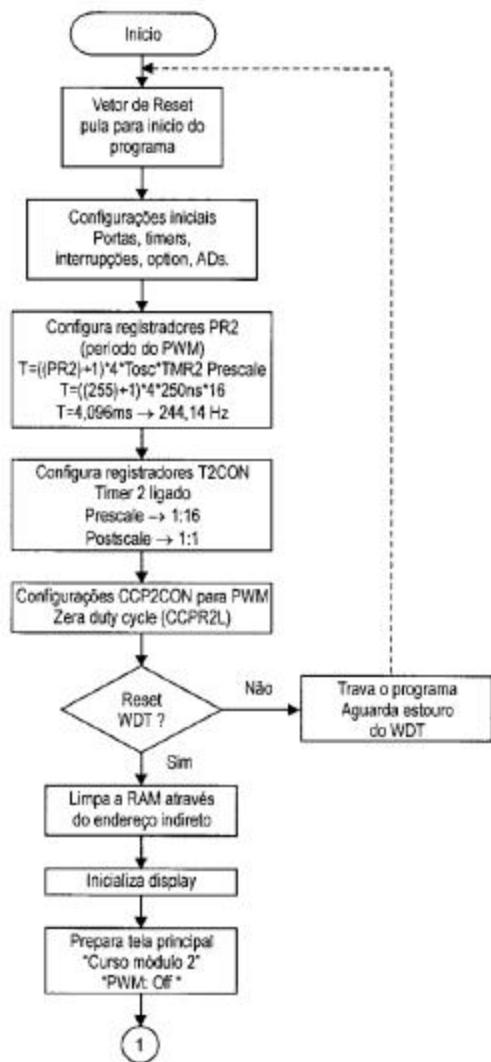
Botão	Duty Cicle
S1	0%
S2	50%
S3	75%
S4	100%

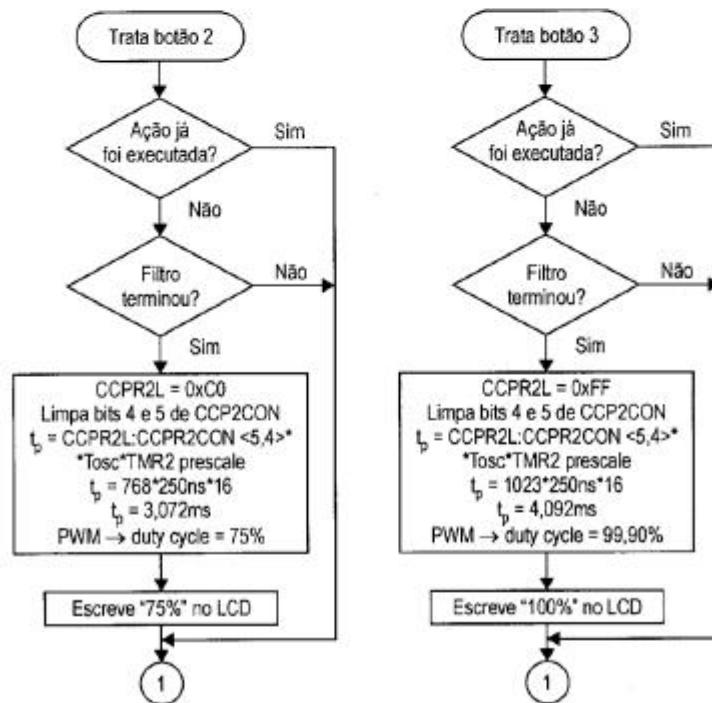
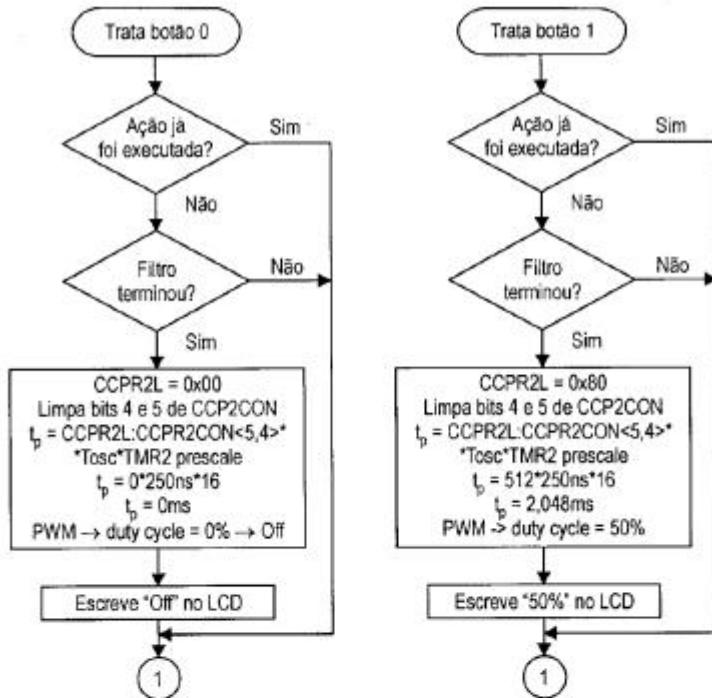
A fim de deixarmos nosso sistema mais interativo, utilizamos o LCD para mostrar o valor atua' ajustado para o PWM.

Esquema Elétrico



Fluxograma





```
;*****  
;*      CONECTANDO O PIC - RECURSOS AVANÇADOS      *  
;  
;*      EXEMPLO 6          *  
;  
;*      NICOLÁS CÉSAR LAVINIA e DAVID JOSÉ DE SOUZA    *  
;  
;*      *  
;*****  
;  
;* VERSÃO : 2.0          *  
;  
;* DATA : 24/02/2003      *  
;  
;*****
```

```
;*****  
;*      DESCRIÇÃO GERAL      *  
;  
;*      *  
;*****  
;  
; ESTE EXEMPLO FOI ELABORADO PARA EXPLICAR O FUNCIONAMENTO DO MÓDULO PWM  
;  
; DO PIC16F877. ELE MONITORA OS QUATRO BOTÕES E CONFORME O BOTÃO SELECIONADO  
;  
; APLICA UM VALOR DIFERENTE NO PWM, FAZENDO ASSIM UM CONTROLE SOBRE A  
;  
; VELOCIDADE DO VENTILADOR. NO LCD É MOSTRADO O VALOR ATUAL DO DUTY CYCLE.  
;  
;
```

```
;*****  
;*      CONFIGURAÇÕES PARA GRAVAÇÃO      *  
;  
;*****
```

```
_CONFIG _CP_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &  
_PWRTE_ON & _WDT_ON & _XT_OSC
```

```
;*****  
;*      DEFINIÇÃO DAS VARIÁVEIS      *  
;  
;*****  
;  
; ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO LOGO NO INÍCIO DO BANCO 0
```

```
CBLOCK 0X20           ; POSIÇÃO INICIAL DA RAM
```

```
FILTRO_BOTOES        ; FILTRO PARA RUIDOS
```

```
TEMPO1  
TEMPO0 ; CONTADORES P/ DELAY
```

```
ENDC
```

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

```
;% DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC *
```

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

```
; O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE  
; OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE  
; DE REDIGITAÇÃO.
```

```
#INCLUDE <P16F877A.INC> ; MICROCONTROLADOR UTILIZADO
```

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

```
;% DEFINIÇÃO DOS BANCOS DE RAM *
```

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

```
; OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR  
; ENTRE OS BANCOS DE MEMÓRIA.
```

```
#DEFINE BANK1 BSF STATUS,RP0 ; SELEciona BANK1 DA MEMORIA RAM  
#DEFINE BANK0 BCF STATUS,RP0 ; SELEciona BANK0 DA MEMORIA RAM
```

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

```
;% CONSTANTES INTERNAS *
```

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

```
; A DEFINIÇÃO DE CONSTANTES FACILITA A PROGRAMAÇÃO E A MANUTENÇÃO.
```

```
FILTRO_TECLA EQU .200 ; FILTRO P/ EVITAR RUIDOS DOS BOTÕES
```

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

```
;% DECLARAÇÃO DOS FLAGs DE SOFTWARE *
```

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

```
; A DEFINIÇÃO DE FLAGs AJUDA NA PROGRAMAÇÃO E ECONOMIZA MEMÓRIA RAM.  
Conectando o PIC 16F877A - Recursos Avançados
```

; ESTE PROGRAMA NÃO UTILIZA NENHUM FLAG DE USUÁRIO

;

;
*: ENTRADAS *;

;
; AS ENTRADAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#DEFINE BOTAO_0 PORTB,0 ; ESTADO DO BOTÃO 0

; 1 -> LIBERADO

; 0 -> PRESSIONADO

#DEFINE BOTAO_1 PORTB,1 ; ESTADO DO BOTÃO 1

; 1 -> LIBERADO

; 0 -> PRESSIONADO

#DEFINE BOTAO_2 PORTB,2 ; ESTADO DO BOTÃO 2

; 1 -> LIBERADO

; 0 -> PRESSIONADO

#DEFINE BOTAO_3 PORTB,3 ; ESTADO DO BOTÃO 3

; 1 -> LIBERADO

; 0 -> PRESSIONADO

;

;
*: SAÍDAS *;

;
; AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#DEFINE DISPLAY PORTD ; BARRAMENTO DE DADOS DO DISPLAY

#DEFINE RS PORTE,0 ; INDICA P/ O DISPLAY UM DADO OU COMANDO

; 1 -> DADO

Conectando o PIC 16F877A - Recursos Avançados

; 0 -> COMANDO

#DEFINE ENABLE PORTE,1 ; SINAL DE ENABLE P/ DISPLAY
 ; ATIVO NA BORDA DE DESCIDA

#DEFINE VENTILADOR PORTC,1 ; SAÍDA P/ O VENTILADOR
 ; 1 -> VENTILADOR LIGADO
 ; 0 -> VENTILADOR DESLIGADO

;
* VETOR DE RESET DO MICROCONTROLADOR *
;

;
* POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA

ORG 0X0000 ; ENDEREÇO DO VETOR DE RESET
GOTO CONFIG ; PULA PARA CONFIG DEVIDO A REGIÃO
 ; DESTINADA AS ROTINAS SEGUINTE

;
* ROTINA DE DELAY (DE 1MS ATÉ 256MS) *
;

;
* ESTA É UMA ROTINA DE DELAY VARIÁVEL, COM DURAÇÃO DE 1MS X O VALOR PASSADO
; EM WORK (W).

DELAY_MS

MOVWF TEMPO1 ; CARREGA TEMPO1 (UNIDADES DE MS)
MOVLW .250
MOVWF TEMPO0 ; CARREGA TEMPO0 (P/ CONTAR 1MS)

CLRWDT ; LIMPA WDT (PERDE TEMPO)

DECFSZ TEMPO0,F ; FIM DE TEMPO0 ?

GOTO \$-2 ; NÃO - VOLTA 2 INSTRUÇÕES

 ; SIM - PASSOU-SE 1MS

DECFSZ TEMPO1,F ; FIM DE TEMPO1 ?

GOTO \$-6 ; NÃO - VOLTA 6 INSTRUÇÕES
 Conectando o PIC 16F877A - Recursos Avançados

; SIM
RETURN ; RETORNA

;* ROTINA DE ESCRITA DE UM CARACTER NO DISPLAY *

; ESTA ROTINA ENVIA UM CARACTER PARA O MÓDULO DE LCD. O CARACTER A SER
; ESCRITO DEVE SER COLOCADO EM WORK (W) ANTES DE CHAMAR A ROTINA.

ESCREVE

MOVWF DISPLAY	; ATUALIZA DISPLAY (PORTD)
NOP	; PERDE 1US PARA ESTABILIZAÇÃO
BSF ENABLE	; ENVIA UM PULSO DE ENABLE AO DISPLAY
GOTO \$+1	;..
BCF ENABLE	;..

MOVlw .1
CALL DELAY_MS ; DELAY DE 1MS
RETURN ; RETORNA

;* CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE *

; NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS
; CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA A
; MÁQUINA E AGUARDA O ESTOURO DO WDT.

CONFIG

CLRF PORTA	; GARANTE TODAS AS SAÍDAS EM ZERO
CLRF PORTB	
CLRF PORTC	
CLRF PORTD	
CLRF PORTE	

BANK1 ; SELECCIONA BANCO 1 DA RAM
Conectando o PIC 16F877A - Recursos Avançados

```
MOVlw B'11111111'  
MOVwf TRISA ; CONFIGURA I/O DO PORTA  
  
MOVlw B'11111111'  
MOVwf TRISB ; CONFIGURA I/O DO PORTB  
  
MOVlw B'11111101'  
MOVwf TRISC ; CONFIGURA I/O DO PORTC  
  
MOVlw B'00000000'  
MOVwf TRISD ; CONFIGURA I/O DO PORTD  
  
MOVlw B'00000100'  
MOVwf TRISE ; CONFIGURA I/O DO PORTE  
  
MOVlw B'11011111'  
MOVwf OPTION_REG ; CONFIGURA OPTIONS  
; PULL-UPs DESABILITADOS  
; INTER. NA BORDA DE SUBIDA DO RB0  
; TIMER0 INCREM. PELO CICLO DE MÁQUINA  
; WDT - 1:128  
; TIMER - 1:1  
  
MOVlw B'00000000'  
MOVwf INTCON ; CONFIGURA INTERRUPÇÕES  
; DESABILITADA TODAS AS INTERRUPÇÕES  
  
MOVlw B'00000111'  
MOVwf ADCON1 ; CONFIGURA CONVERSOR A/D  
; CONFIGURA PORTA E PORTE COMO I/O DIGITAL  
  
MOVlw .255  
MOVwf PR2 ; CONFIGURA PERÍODO DO PWM  
; T=((PR2)+1)*4*Tosc*TMR2 Prescale  
Conectando o PIC 16F877A - Recursos Avançados
```

; T=((255)+1)*4*250ns*16

; T=4,096ms -> 244,14Hz

BANK0 ; SELECCIONA BANCO 0 DA RAM

MOVlw B'00000111'

MOVWF T2CON ; CONFIGURA TMR2

; TIMER 2 LIGADO

; PRESCALE - 1:16

; POSTSCALE - 1:1

MOVlw B'00001111'

MOVWF CCP2CON ; CONFIGURA CCP2CON PARA PWM

; (PINO RC1)

CLRF CCPR2L ; INICIA COM DUTY CYCLE EM ZERO

; AS INSTRUÇÕES A SEGUIR FAZEM COM QUE O PROGRAMA TRAVE QUANDO HOUVER UM

; RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT. DESTA FORMA,

; SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA UM ESTOURO DE WDT

; E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO START-UP DO PIC.

BTFSR STATUS,NOT_TO ; RESET POR ESTOURO DE WATCHDOG TIMER ?

GOTO \$; NÃO - AGUARDA ESTOURO DO WDT

; SIM

* INICIALIZAÇÃO DA RAM *

; ESTA ROTINA IRÁ LIMPAR TODA A RAM DO BANCO 0, INDO DE 0X20 A 0X7F.

; EM SEGUIDA, AS VARIÁVEIS DE RAM DO PROGRAMA SÃO INICIALIZADAS.

MOVlw 0X20

MOVWF FSR ; APONTA O ENDEREÇAMENTO INDIRETO PARA

; A PRIMEIRA POSIÇÃO DA RAM

Conectando o PIC 16F877A - Recursos Avançados

LIMPA_RAM

```
CLRF    INDF          ; LIMPA A POSIÇÃO
INCF    FSR,F         ; INCREMENTA O PONTEIRO P/ A PRÓX. POS.
MOVF    FSR,W
XORLW  0X80          ; COMPARA O PONTEIRO COM A ÚLT. POS. +1
BTFSS  STATUS,Z      ; JÁ LIMPOU TODAS AS POSIÇÕES?
GOTO   LIMPA_RAM     ; NÃO - LIMPA A PRÓXIMA POSIÇÃO
                      ; SIM
```

;*****

;* CONFIGURAÇÕES INICIAIS DO DISPLAY *

;*****

```
; ESTA ROTINA INICIALIZA O DISPLAY P/ COMUNICAÇÃO DE 8 VIAS, DISPLAY PARA 2
; LINHAS, CURSOR APAGADO E DESLOCAMENTO DO CURSOR À DIREITA.
```

INICIALIZACAO_DISPLAY

```
BCF    RS             ; SELECCIONA O DISPLAY P/ COMANDOS
```

```
MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
```

```
CALL   ESCREVE       ; INICIALIZAÇÃO
```

```
MOVLW .3
```

```
CALL   DELAY_MS      ; DELAY DE 3MS (EXIGIDO PELO DISPLAY)
```

```
MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
```

```
CALL   ESCREVE       ; INICIALIZAÇÃO
```

```
MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
```

```
CALL   ESCREVE       ; INICIALIZAÇÃO
```

```
MOVLW B'00111000'   ; ESCREVE COMANDO PARA
```

```
CALL   ESCREVE       ; INTERFACE DE 8 VIAS DE DADOS
```

```
MOVLW B'00000001'   ; ESCREVE COMANDO PARA
```

```
CALL   ESCREVE       ; LIMPAR TODO O DISPLAY
Conectando o PIC 16F877A - Recursos Avançados
```

```

MOVlw .1

CALL    DELAY_MS           ; DELAY DE 1MS

MOVlw B'00001100'          ; ESCREVE COMANDO PARA
CALL    ESCREVE            ; LIGAR O DISPLAY SEM CURSOR

MOVlw B'00000110'          ; ESCREVE COMANDO PARA INCREM.
CALL    ESCREVE            ; AUTOMÁTICO À DIREITA

;*****
;*      ROTINA DE ESCRITA DA TELA PRINCIPAL      *
;*****                                         ; ESTA ROTINA ESCREVE A TELA PRINCIPAL DO PROGRAMA, COM AS FRASES:
; LINHA 1 - "CURSO MODULO 2"
; LINHA 2 - " PWM: xx% "

MOSTRA_TELA_PRINCIPAL

MOVlw 0X81                 ; COMANDO PARA POSICIONAR O CURSOR
CALL    ESCREVE            ; LINHA 0 / COLUNA 1
BSF     RS                 ; SELECCIONA O DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS
; LETRAS DE "CURSO MODULO 2"

MOVlw 'C'
CALL    ESCREVE
MOVlw 'U'
CALL    ESCREVE
MOVlw 'R'
CALL    ESCREVE
MOVlw 'S'
CALL    ESCREVE
MOVlw 'O'
CALL    ESCREVE
MOVlw ''

```

CALL ESCREVE
MOVLW 'M'
CALL ESCREVE
MOVLW 'O'
CALL ESCREVE
MOVLW 'D'
CALL ESCREVE
MOVLW 'U'
CALL ESCREVE
MOVLW 'L'
CALL ESCREVE
MOVLW 'O'
CALL ESCREVE
MOVLW ''
CALL ESCREVE
MOVLW '2'
CALL ESCREVE

BCF RS ; SELECAO O DISPLAY P/ COMANDOS
MOVLW 0XC3 ; COMANDO PARA POSICIONAR O CURSOR
CALL ESCREVE ; LINHA 1 / COLUNA 3
BSF RS ; SELECAO O DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS
; LETRAS DE " PWM: OFF "
MOVLW 'P'
CALL ESCREVE
MOVLW 'W'
CALL ESCREVE
MOVLW 'M'
CALL ESCREVE
MOVLW ':'
CALL ESCREVE
MOVLW ''
CALL ESCREVE

```

MOVlw ''
CALL    ESCREVE
MOVlw 'O'
CALL    ESCREVE
MOVlw 'F'
CALL    ESCREVE
MOVlw 'F'
CALL    ESCREVE

;*****
;*          VARREDURA DOS BOTÕES
;*****          *
; ESTA ROTINA VERIFICA SE ALGUM BOTÃO ESTÁ PRESSIONADO E CASO AFIRMATIVO
; DESVIA PARA O TRATAMENTO DO MESMO.

VARRE
CLRWDT           ; LIMPA WATCHDOG TIMER

; ***** VERIFICA ALGUM BOTÃO PRESSIONADO *****
VARRE_BOTOES
BTFFS  BOTAO_0           ; O BOTÃO 0 ESTÁ PRESSIONADO ?
GOTO   TRATA_BOTAO_0     ; SIM - PULA P/ TRATA_BOTAO_0
                           ; NÃO

BTFFS  BOTAO_1           ; O BOTÃO 1 ESTÁ PRESSIONADO ?
GOTO   TRATA_BOTAO_1     ; SIM - PULA P/ TRATA_BOTAO_1
                           ; NÃO

BTFFS  BOTAO_2           ; O BOTÃO 2 ESTÁ PRESSIONADO ?
GOTO   TRATA_BOTAO_2     ; SIM - PULA P/ TRATA_BOTAO_2
                           ; NÃO

BTFFS  BOTAO_3           ; O BOTÃO 3 ESTÁ PRESSIONADO ?
GOTO   TRATA_BOTAO_3     ; SIM - PULA P/ TRATA_BOTAO_3
                           Conectando o PIC 16F877A - Recursos Avançados

```

; NÃO

; ***** FILTRO P/ EVITAR RUIDOS *****

```
MOVWF FILTRO_TECLA           ; CARREGA O VALOR DE FILTRO_TECLA  
MOVWF FILTRO_BOTOES          ; SALVA EM FILTRO_BOTOES  
                                ; RECARREGA FILTRO P/ EVITAR RUIDOS  
                                ; NOS BOTÕES
```

```
GOTO VARRE                  ; VOLTA PARA VARRER TECLADO
```

;*****

;* TRATAMENTO DOS BOTÕES *

;*****

; NESTE TRECHO DO PROGRAMA ESTÃO TODOS OS TRATAMENTOS DOS BOTÕES

;***** TRATAMENTO DO BOTÃO 0 *****

TRATA_BOTAO_0

```
MOVF FILTRO_BOTOES,F  
BTFS C STATUS,Z              ; FILTRO JÁ IGUAL A ZERO ?  
                                ; (FUNÇÃO JA FOI EXECUTADA?)  
GOTO VARRE                  ; SIM - VOLTA P/ VARREDURA DO TECLADO  
                                ; NÃO  
DECFSZ FILTRO_BOTOES,F      ; FIM DO FILTRO ? (RUIDO?)  
GOTO VARRE                  ; NÃO - VOLTA P/ VARRE  
                                ; SIM - BOTÃO PRESSIONADO
```

```
CLRF CCP2L                   ; ZERA CCP2L  
BCF CCP2CON,5                ; ZERA OS BITS 5 e 4  
BCF CCP2CON,4                ; (LSB DO DUTY CYCLE)  
                                ; Tp = CCP2L:CCP2CON<5,4>*Tosc*TMR2 Prescale  
                                ; Tp = 0 * 250ns * 16  
                                ; Tp = 0
```

; PWM -> DUTY CYCLE = 0% -> OFF
Conectando o PIC 16F877A - Recursos Avançados

```

BCF    RS           ; SELECCIONA O DISPLAY P/ COMANDOS
MOVlw 0XC8          ; COMANDO PARA POSICIONAR O CURSOR
CALL   ESCREVE      ; LINHA 1 / COLUNA 8
BSF    RS           ; SELECCIONA O DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS
; LETRAS DE " OFF"
MOVlw ''
CALL   ESCREVE
MOVlw 'O'
CALL   ESCREVE
MOVlw 'F'
CALL   ESCREVE
MOVlw 'F'
CALL   ESCREVE

GOTO  VARRE        ; VOLTA P/ VARREDURA DOS BOTÕES

```

; ***** TRATAMENTO DO BOTÃO 1 *****

TRATA_BOTAO_1

```

MOVf  FILTRO_BOTOES,F
BTFSZ STATUS,Z       ; FILTRO JÁ IGUAL A ZERO ?
; (FUNÇÃO JA FOI EXECUTADA?)
GOTO  VARRE        ; SIM - VOLTA P/ VARREDURA DO TECLADO
; NÃO
DECFSZ FILTRO_BOTOES,F ; FIM DO FILTRO ? (RUIDO?)
GOTO  VARRE        ; NÃO - VOLTA P/ VARRE
; SIM - BOTÃO PRESSIONADO

MOVlw 0X80
MOVWF CCPR2L         ; CARREGA CCPR2L COM 0X80
BCF    CCP2CON,5      ; LIMPA OS BITS 5 e 4
BCF    CCP2CON,4      ; LSB DO DUTY CYCLE

```

; Tp = CCPR2L:CCP2CON<5,4>*Tosc*TMR2 Prescale
 Conectando o PIC 16F877A - Recursos Avançados

; Tp = 512 * 250ns * 16

; Tp = 2,048ms

; PWM -> DUTY CYCLE = 50%

BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS
MOVlw 0XC8 ; COMANDO PARA POSICIONAR O CURSOR
CALL ESCREVE ; LINHA 1 / COLUNA 8
BSF RS ; SELECCIONA O DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS

; LETRAS DE " 50%"

MOVlw ''
CALL ESCREVE
MOVlw '5'
CALL ESCREVE
MOVlw '0'
CALL ESCREVE
MOVlw '%'
CALL ESCREVE

GOTO VARRE ; VOLTA P/ VARREDURA DOS BOTÕES

; ***** TRATAMENTO DO BOTÃO 2 *****

TRATA_BOTAO_2

MOVf FILTRO_BOTOES,F
BTFSZ STATUS,Z ; FILTRO JÁ IGUAL A ZERO ?
; (FUNÇÃO JA FOI EXECUTADA?)
GOTO VARRE ; SIM - VOLTA P/ VARREDURA DO TECLADO
; NÃO
DECFSZ FILTRO_BOTOES,F ; FIM DO FILTRO ? (RUIDO?)
GOTO VARRE ; NÃO - VOLTA P/ VARRE
; SIM - BOTÃO PRESSIONADO
MOVlw 0XC0
MOVWF CCPR2L ; CARREGA CCPR2L COM 0XC0
Conectando o PIC 16F877A - Recursos Avançados

```
BCF    CCP2CON,5           ; LIMPA OS BITS 5 e 4
BCF    CCP2CON,4           ; LSB DO DUTY CYCLE
                                ;Tp = CCPR2L:CCP2CON<5,4>*Tosc*TMR2 Prescale
                                ;Tp = 768 * 250ns * 16
                                ;Tp = 3,072ms
                                ;PWM -> DUTY CYCLE = 75%
```

```
BCF    RS                 ; SELECAO O DISPLAY P/ COMANDOS
MOVLW 0XC8                 ; COMANDO PARA POSICIONAR O CURSOR
CALL   ESCREVE             ; LINHA 1 / COLUNA 8
BSF    RS                 ; SELECAO O DISPLAY P/ DADOS
```

```
; COMANDOS PARA ESCREVER AS
; LETRAS DE "75%"
MOVLW ''
CALL   ESCREVE
MOVLW '7'
CALL   ESCREVE
MOVLW '5'
CALL   ESCREVE
MOVLW '%'
CALL   ESCREVE
```

```
GOTO  VARRE              ; VOLTA P/ VARREDURA DOS BOTÕES
```

```
;***** TRATAMENTO DO BOTÃO 3 *****
```

```
TRATA_BOTAO_3
MOVF  FILTRO_BOTOES,F
BTFSC STATUS,Z            ; FILTRO JÁ IGUAL A ZERO ?
                            ; (FUNÇÃO JA FOI EXECUTADA?)
GOTO  VARRE              ; SIM - VOLTA P/ VARREDURA DO TECLADO
                            ; NÃO
DECFSZ FILTRO_BOTOES,F   ; FIM DO FILTRO ? (RUIDO?)
GOTO  VARRE              ; NÃO - VOLTA P/ VARRE
Conecando o PIC 16F877A - Recursos Avançados
```

; SIM - BOTÃO PRESSIONADO

```
MOVlw 0xFF
MOVwf CCP2R2L           ; CARREGA CCP2R2L COM 0xFF
BSF    CCP2CON,5          ; SETA OS BITS 5 e 4
BSF    CCP2CON,4          ; LSB DO DUTY CYCLE
;Tp = CCP2R2L:CCP2CON<5,4>*Tosc*TMR2 Prescale
;Tp = 1023 * 250ns * 16
;Tp = 4,092ms
;PWM -> DUTY CYCLE = 99,90%
```

```
BCF    RS                ; SELECCIONA O DISPLAY P/ COMANDOS
MOVlw 0XC8               ; COMANDO PARA POSICIONAR O CURSOR
CALL   ESCREVE           ; LINHA 1 / COLUNA 8
BSF    RS                ; SELECCIONA O DISPLAY P/ DADOS
```

```
; COMANDOS PARA ESCREVER AS
; LETRAS DE "100%"
MOVlw '1'
CALL   ESCREVE
MOVlw '0'
CALL   ESCREVE
MOVlw '0'
CALL   ESCREVE
MOVlw '%'
CALL   ESCREVE
```

```
GOTO  VARRE            ; VOLTA P/ VARREDURA DOS BOTÕES
```

;
*: FIM DO PROGRAMA

*

```
END                  ; FIM DO PROGRAMA
```

Dicas e comentários

Como foi explicado na lógica do exemplo, deixamos o valor de **PR2** em 255 e o prescaler do Timer 2 ajustado para 16. Com isso nosso PWM possui um período de 4,096ms. Para podermos operar com o PWM em diversos níveis (0, 50, 75 e 100%) foi necessário calcular os valores corretos a serem carregados em **CCPR2**. Desta forma, foi efetuada a seguinte conta:

$$\text{CCPR2} = [(\text{PR2})+1] \times 4 \times \text{Porcentagem desejada}$$

No nosso caso:

- Porcentagem = 0%, 50%, 75% e 100%
- **PR2 = 255**

Com isso obtivemos os seguintes valores:

- **Para 0%: CCPR2 = 0**, sem arredondamento e sem erro;
- **Para 50%: CCPR2 = 512**. Largura do pulso em 2,048ms, resultando numa porcentagem final de 50,0%, também sem arredondamento e sem erro;
- **Para 75%: CCPR2 = 768**. Largura do pulso em 3,072ms, resultando novamente numa porcentagem final de 75,0% sem erros de arredondamento;
- **Para 100%: CCPR2 = 1023**. Já neste caso, a largura do pulso ficou em 4,092ms, resultando numa porcentagem final de 99,90%. Para 100%, CCPR2 deve ser arredondado para cima ultrapassando o valor do período e garantindo o pulso sempre em 1. Porém, isso não é possível, pois já estamos no limite máximo de 1023.

Exercícios propostos

Agora que o exemplo já foi estudado e esclarecido, aproveite para gerar novos problemas e soluções, seguindo os exercícios propostos:

1. Corrija o problema encontrado no nível 100%, evitando que a saída seja colocada em zero, mesmo que por um período de tempo muito curto.
2. Em vez de trabalhar com somente quatro níveis de PWM, altere o sistema para que um botão ligue e desligue a saída e outros dois botões incremente e decremente o PWM, de 50 a 100% com passos de 5%. Não recomendamos valores muito baixos para esta saída, pois o ventilador possui uma tensão mínima de trabalho.
3. Ative as duas saídas PWMs ao mesmo tempo, uma para o ventilador e outra para a resistência. Utilize dois botões para controlar o ajuste de cada uma delas.

Trabalhando com as Memórias Não-Voláteis

Introdução

Nosso estudo já está bem avançado e provavelmente o assunto deste capítulo você já conhece, pois os PICs 16F84 ou 16F628 (focos do livro "Desbravando o PIC") também possuem esse tipo de memória. Mas não custa nada relembrarmos a matéria e explicarmos as características diferentes do modelo 16F877A, sendo a principal delas o acesso direto à memória de programa.

Teoria e recursos do PIC

Quando uma memória é do tipo não-volátil, significa que ela é capaz de continuar armazenando seus dados mesmo quando não energizada, ao contrário da RAM, que perde todas as suas informações quando o PIC é desligado. 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.

A grande diferença do PIC 16F877A em relação a outros modelos mais simples, como o 16F84 ou 16F62X, é que esse modelo possui acesso a dois tipos de memórias não-voláteis:

Tipo	Tamanho	Observações
Dados (E ² PROM)	256 x 8 bits	A memória de dados é o tipo de memória convencional para armazenamento de informações do usuário. Esta memória é conhecida como E ² PROM.
Programa (FLASH)	8k x 14 bits	A memória de programa pode ser usada para alterar o próprio software ou mesmo como expansão da memória de dados. Por ser um tipo especial de E ² PROM, ela é conhecida como FLASH.

Muitas vezes uma memória E²PROM externa pode ser necessária, principalmente quando os 256 bytes disponíveis não são suficientes. A vantagem aqui é que a memória de programa também pode ser utilizada como expansão da memória de dados, com a vantagem de que ele ainda é de 14 bits, e não de 3. O acesso à memória externa existente na placa proposta será visto no próximo capítulo.

Muito interessante, não é? Quer dizer que podemos sair escrevendo nossos dados na memória de programa indefinidamente? Não é bem assim, alguns cuidados devem ser observados:

- A memória de dados (E²PROM) possui uma vida útil bem maior que a memória de programa (FLASH). Enquanto a primeira suporta 100.000 ciclos de escrita, a segunda possui uma vida útil assegurada de somente 1.000 ciclos. Quanto à velocidade, ambas respondem igualmente: 4ms nominais para um ciclo de escrita (máximo 8ms);
- Não podemos esquecer de que a memória FLASH continua sendo uma "Memória de programa". Isso significa que seu tamanho útil será somente a sobre em relação ao tamanho ocupado pelo programa propriamente dito;
- Por último, e o mais importante, é que o acesso a memória FLASH é irrestrito. Com isso, podemos acabar escrevendo onde não se deve. Caso seja utilizada, para armazenar um dado, uma posição da memória ocupada pelo programa, todo o sistema pode travar ou executar ações completamente aleatórias. Por isso, muita atenção ao utilizar essa memória, principalmente porque uma vez escrita, não adianta resetar o sistema para voltar ao normal.

Como já foi dito, a última observação é considerada a mais importante de todas. Por outro lado, ela possibilita também recursos extremamente avançados. Como assim? Bem, se temos acesso irrestrito a toda a área de programa, então podemos ler e escrever nela à vontade, certo? Isso mesmo. E com isso podemos criar sistemas capazes de se auto-atualizarem. Podemos, por exemplo, elaborar um projeto que recebe um comando via serial (ligação com PC ou modem) e com isso entrar em modo de reconfiguração. Depois, ele recebe, pela mesma via, partes do programa que devem ser alteradas. Nossa sistema pode então sofrer um *up-grade* pela porta serial. A grande vantagem disso é que não estamos limitados a alterar somente parâmetros gravados na E²PROM de dados. Podemos alterar as funções do sistema.

Voltemos agora ao nosso assunto principal. Como ter acessos a essas memórias, seja ela de dados ou de programa.

As diferenças para as rotinas existentes para os PICs mais simples são:

- Como a informação pode ser de 8 ou 14 bits, existem dois registradores para armazenar o dado: **EEDATH** (parte alta) e **EEDATA** (parte baixa);
- Para a memória de E2PROM existem 256 posições. Isso significa que um único byte é capaz de endereçar toda a área disponível. Entretanto, para a memória FLASH existe uma área com tamanho de 8K. Por isso, para o endereço também existem agora dois registradores: **EEADRH** (parte alta) e **EEADR** (parte baixa);
- Os registradores **EECON1** e **EECON2** continuam existindo e possuem as mesmas funções: controlar as operações de escrita e leitura. A única diferença é que foi criado mais um bit em **EECON1<EEPGD>**, que é responsável pela seleção da memória com a qual desejamos trabalhar:

EEP GD	Memória
0	E ² RROM (dados)
1	FLASH (programa)

- Atenção especial aos bancos onde se encontram os registradores:

Registrador Banco /Endereço	
EECON1	3/18Ch
EECON2	3/18Dh

Registrador	Banco / Endereço
EEDATA	2/10CH
EEDATH	2/10Eh
EEADR	2/10Dh
EEADRH	2/10Fh

Escrevendo na E²PROM (Dados)

A primeira ação que faremos em relação a E²PROM 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 necessária para garantirmos a proteção do sistema, para evitarmos escritas acidentais na memória. Por isso, a inicialização da escrita parecerá um pouco confusa e desnecessária, mas isso torna o sistema robusto e seguro.

A escrita da E²PROM deve seguir o roteiro:

1. O endereço para a escrita deve ser colocado em **EEADR**. Como existem 256 bytes disponíveis, este endereço deve estar entre 0 e 255. Não se esqueça de alterar para o **BANK2** antes de atualizar esse registrador.
2. O dado a ser escrito deve ser colocado em **EEDATA**. Só podemos escrever um byte de cada vez. Esse registrador também encontra-se em **BANK2**.
3. Devemos ajustar a opção de trabalho para memória de dados (**E PROM²**) através de **EECON1<EEPGD>=0**. Antes devemos alterar para **BANK3**.
4. A escrita deve ser habilitada através de **EECON1 <WREN>=1**. Continua em **BANKS**.
5. As interrupções devem ser desligadas para evitarmos conflitos, por meio de **INTCON<GIE>=0**. Aqui não existe problema quanto ao banco de memória.
6. O registrador **EECON2** deve ser carregado com os valores 0x55 e 0xAA, seqüencialmente. Esse procedimento é obrigatório e utilizado para a proteção da escrita. Essas operações também continuam sendo feitas no **BANK3**.
7. A escrita deve ser iniciada através do bit **EECON1<WR>=1** e **EECON1<WREN>=0**, nesta ordem. Mais uma vez não se altera o banco.
8. As interrupções podem ser novamente ligadas com **INTCON<GIE>=1**.
9. 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 essa ação aconteça. No caso de não podermos ficar esperando pelo fim da escrita, podemos ligar a interrupção relacionada a esse evento através do bit **PIE2<EEIE>** e esperar que ela aconteça para considerarmos finalizada a escrita.
10. 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).
11. Não se esqueça de ajustar o banco de memória novamente, conforme suas necessidades, para a continuação da execução do programa.

Lendo a E²PROM (Dados)

Agora que você já sabe escrever alguma coisa nos endereços disponíveis da E²PROM, poderemos ler essa informação de volta através de uma rotina de leitura. A leitura é muito mais simples, pois não necessita de tanta proteção. Também é muito mais rápida.

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

1. O endereço para a escrita deve ser colocado em **EEADR**. Como existem 256 bytes disponíveis, este endereço deve estar entre 0 e 255. Não se esqueça de alterar para o **BANK2** antes de atualizar esse registrador.
2. Devemos ajustar a opção de trabalho para memória de dados (E PROM²) através de **EECON1<EEPGD>=0**. Antes devemos alterar para **BANK3**.
3. A leitura deve ser ligada através do bit **EECON1 <RD>=1**.
4. O dado lido será colocado em **EEDATA**. Para acessá-lo não se esqueça de alterar antes para **BANK2**.

Escrevendo na FLASH (Programa)

Esta operação é muito semelhante à escrita da memória de dados, com a diferença básica do endereço e do dado ser de 14 bits. Os PIC mais novos, como a versão 16F877A possuem uma limitação em relação à escrita na memória FLASH. Agora, os dados só podem ser escritos em blocos de 4, começando obrigatoriamente nos endereços terminados em 00 (binário). Por causa disso, para escrever em um único endereço, será necessário antes ler e armazenar temporariamente o valor existente nos outros 3 endereços, para que os mesmos não sejam perdidos. A complexibilidade é a mesma para garantir a segurança em relação à escrita acidental.

O roteiro para esta operação fica então da seguinte maneira:

1. O endereço para a escrita deve ser colocado em **EEADRH** (parte alta) e **EEADR** (parte baixa). Como existem 8K disponíveis, este endereço deve estar entre 0 e 8.191 (1FFFh), mas deve ser utilizado obrigatoriamente um endereço terminado em 00 (binário). Não se esqueça de alterar para o **BANK2** antes de atualizar estes registradores.
2. O primeiro dado a ser escrito deve ser colocado em **EEDATH** (parte alta) e **EEDATA** (parte baixa). Como só podemos escrever uma palavra de 14 bits de cada vez, os 2 bits mais significativos de **EEDATH** serão ignorados. Esses registradores também encontram-se em **BANK2**.
3. Devemos ajustar a opção de trabalho para memória de programa através de **EECON1 <EEPGD>=1**. Antes devemos alterar para **BANK3**.
4. A escrita deve ser habilitada através de **EECON1 <WREN>=1**. Continua em **BANK3**.
5. As interrupções devem ser desligadas para evitarmos conflitos, através de **INTCON<GIE>=0**. Aqui não existe problema quanto ao banco de memória.
6. O registrador **EECON2** deve ser carregado com os valores **0x55** e **0xAA**, seqüencialmente. Este procedimento é obrigatório e utilizado para a proteção da escrita. Essas operações também continuam sendo feitas no **BANK3**.

7. A escrita deve ser iniciada através do bit **EECON1<WR>=1** e **EECON1<WREN>=0**, nesta ordem. Mais uma vez não se altera o banco.
8. Duas instruções **NOP** devem ser colocadas no programa, obrigatoriamente, devido a lógica interna do sistema. Não podem ser substituídas por GOTO \$+1.
9. As interrupções podem ser novamente ligadas com **INTCON<GIE>=1**.
10. 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 essa ação aconteça. No caso de não podermos ficar esperando pelo fim da escrita, podemos ligar a interrupção relacionada a este evento através do bit **PIE2<EEIE>** e esperar que ela aconteça para considerarmos finalizada a escrita.
11. O próximo dado deve ser colocado em **EEDATAH** e **EEDATAL** e o processo repetido a partir do passo 3. Isto deve ser feito 4 vezes seguidas. Durante as 3 primeiras passagens, os dados serão armazenados em um buffer temporário. Somente ao término da 4ª é que todos os dados serão gravados de fato. Caso a rotina não seja completa, os dados serão perdidos.
12. Caso algum erro ocorra durante a operação de escrita, o bit **EECON1<WRERR>** será selado (1). No caso de sucesso na operação este bit será mantido em zero (0).
13. Não se esqueça de ajustar o banco de memória novamente, conforme suas necessidades, para a continuação da execução do programa.

Lendo a FLASH (Programa)

A relação entre a leitura da memória de dados e a de programa é a mesma que a comentada para =s operações de escrita.

O roteiro para a criação da rotina de leitura fica sendo da seguinte maneira:

1. O endereço para a leitura deve ser colocado em **EEADRH** (parte alta) e **EEADR** (parte baixa). Como existem 8 K disponíveis, este endereço deve estar entre 0 e 8191 (1FFFh). Não se esqueça de alterar para o **BANK2** antes de atualizar estes registradores.
2. Devemos ajustar a opção de trabalho para memória de programa através de **EECON1 <EEPGD>=1**. Antes devemos alterar para **BANK3**.
3. A leitura deve ser ligada através do bit **EECON1 <RD>=1**.
4. Duas instruções **NOP** devem ser colocadas no programa, obrigatoriamente, devido a lógica, interna do sistema. Não podem ser substituídas por GOTO \$+1.
5. O dado lido será colocado em **EEDATH** (parte alta) e **EEDATA** (parte baixa). Para acessá-los não se esqueça de alterar antes para **BANK2**.

Tratando a interrupção de final de escrita na E²PROM e FLASH

Esta 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 **EECON1<WR>** como no exemplo anterior). Quando a escrita for terminada, uma interrupção irá ocorrer e o sistema poderá tomar as ações pertinentes.

Para que esta interrupção possa ocorrer, sua chave individual **PIE2<EEIE>** deve estar ligada (1). Dentro da rotina de tratamento devemos testar o flag **PIR2<EEIF>** para sabermos se foi esta interrupção que ocorreu. Antes de sairmos do tratamento devemos limpar este mesmo flag manualmente.

Resumo dos registradores associados a E PROM² / FLASH

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	T0IF	INTF	RBIF
ODh	PIR2	-	-	-	EEIF	BCLIF	-	-	CCP2IF
8Dh	PIE2	-	-	-	EEIE	BCLIE	-	-	CCP2IE
180Ch	EECON1	EEPGD	-	-	-	WRERR	WREN	WR	RD
18Dh	EECON2	Registrador de segurança para operações de escrita na E PROM/FLASH							
10Ch	EEDATA	Dado a ser escrito na E PROM/FLASH (Parte baixa)							
10Eh	EEDATH	Dado a ser escrito na FLASH (Parte alta)							
10Dh	EEADR	Endereço da escrita da E PROM/FLASH (Parte baixa)							
10Fh	EEADRH	Endereço da escrita da FLASH (Parte alta)							

Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Lógica do exemplo

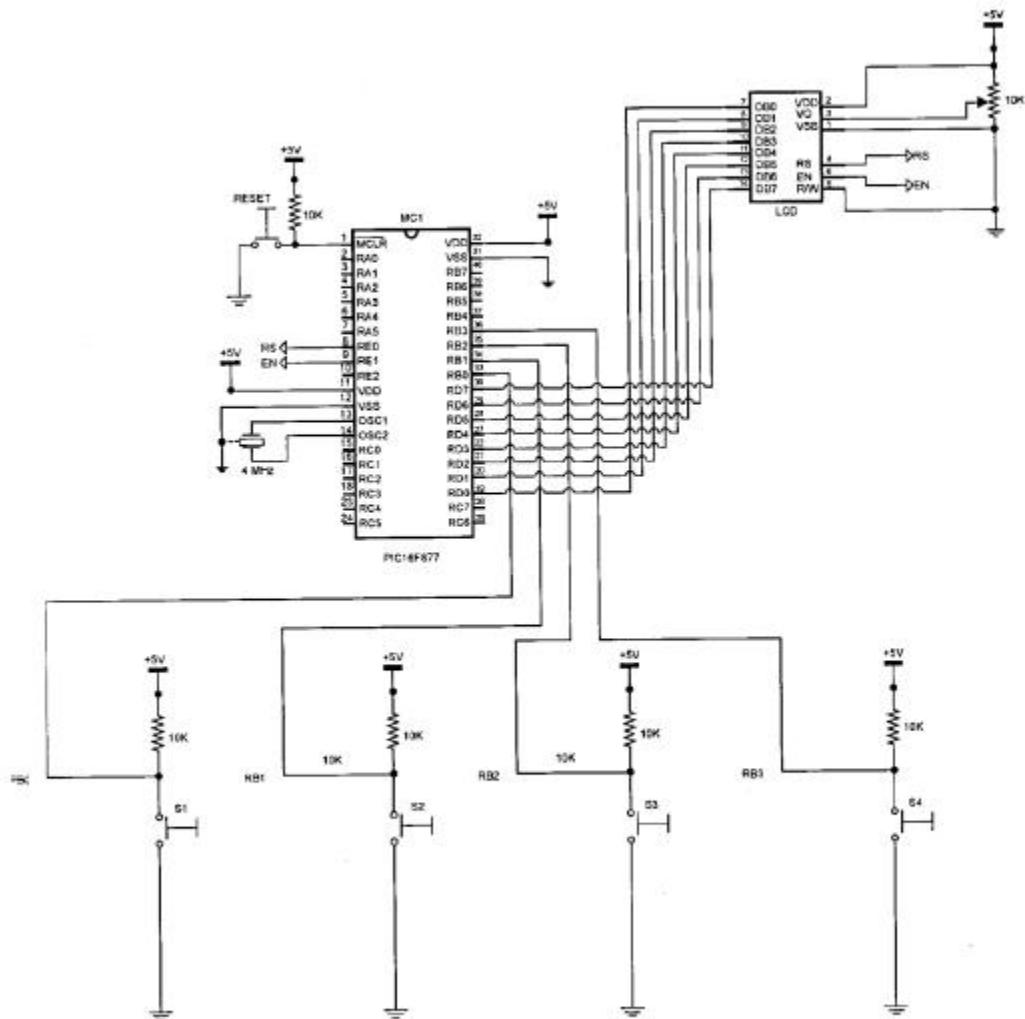
Nosso sistema dividirá o LCD em duas partes. Do lado esquerdo teremos um valor relativo à memória de dados, variável de 0 a FFh (8-bits), com incremento e decremento rotativo através dos botões S2 e S3. Do lado direito o valor será para a memória de programa, também com incremento e decremento rotativo através dos botões S2 e S3, podendo ir de 0 a 3FFFh (14-bits).

Para alterar o controle dos botões S2 e S3 entre o lado esquerdo e o lado direito deve ser usado o botão S1. Para o lado ativo, no momento, o valor será indicado entre os sinais > e <.

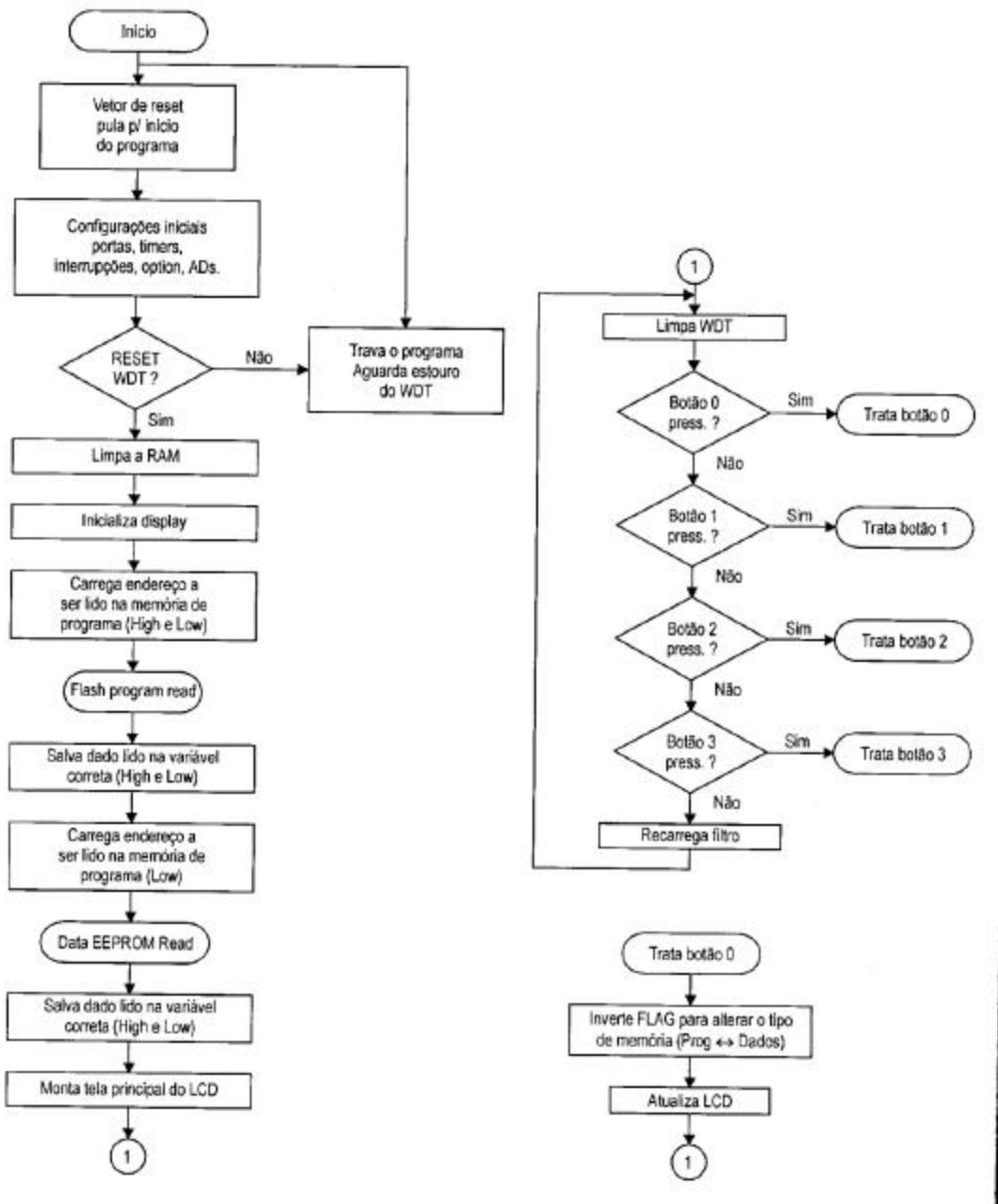
Depois de ajustados os valores desejados, basta pressionar o botão S4 para que ambos sejam gravados, cada um na memória correspondente.

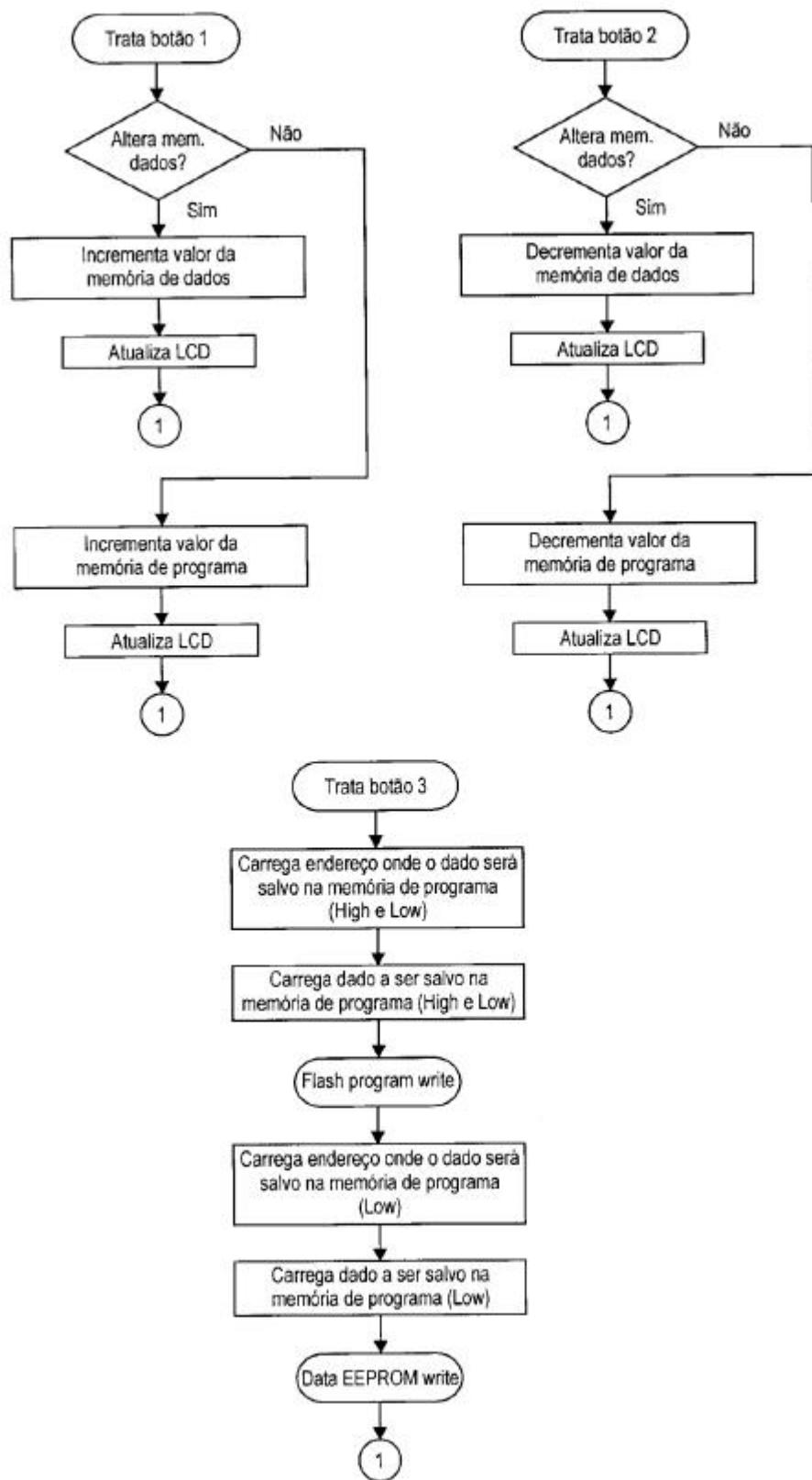
Para checar a gravação, altere os valores e reset o sistema. Os valores gravados serão recuperados na inicialização e mostrados no LCD.

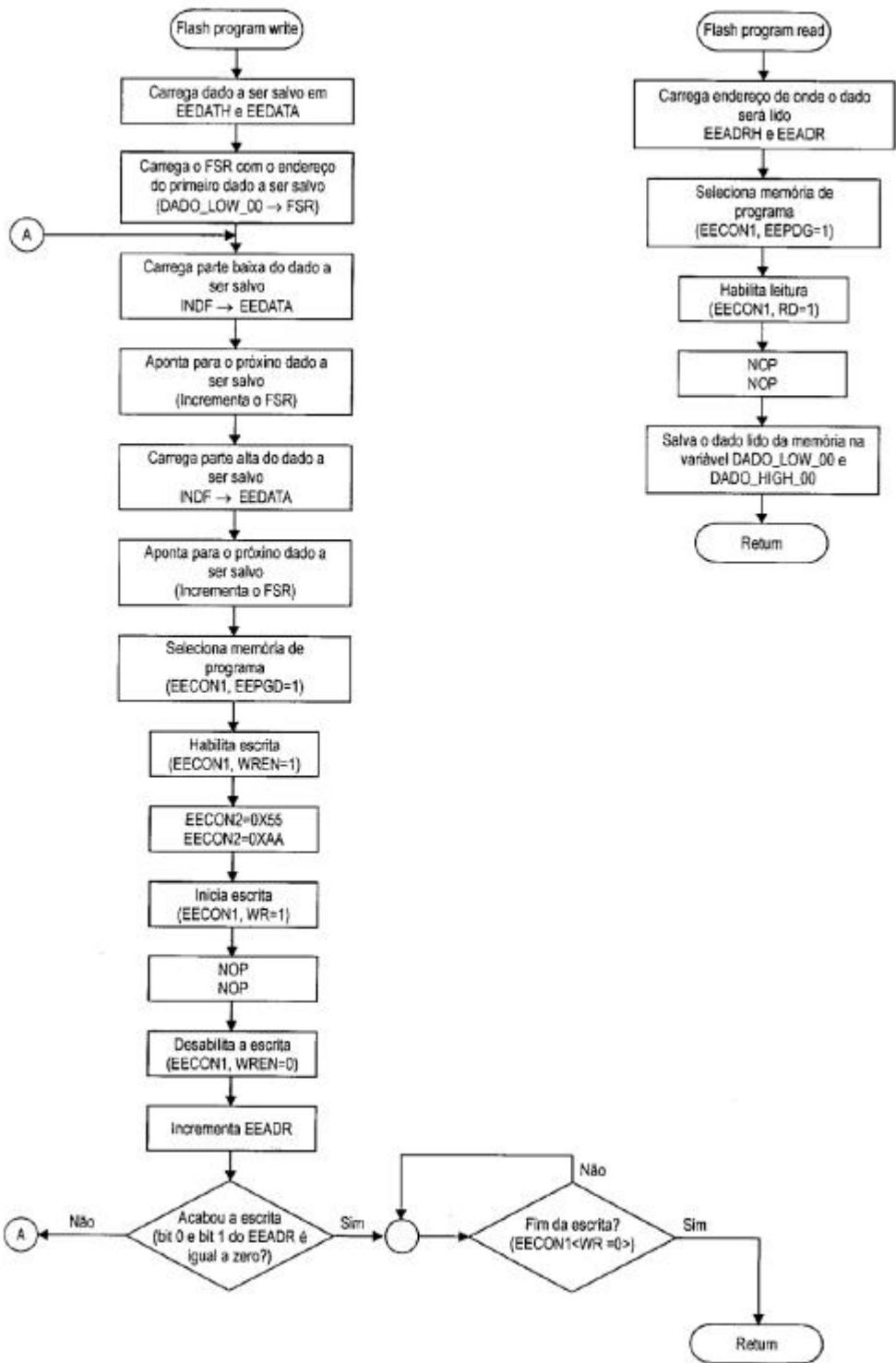
Esquema elétrico

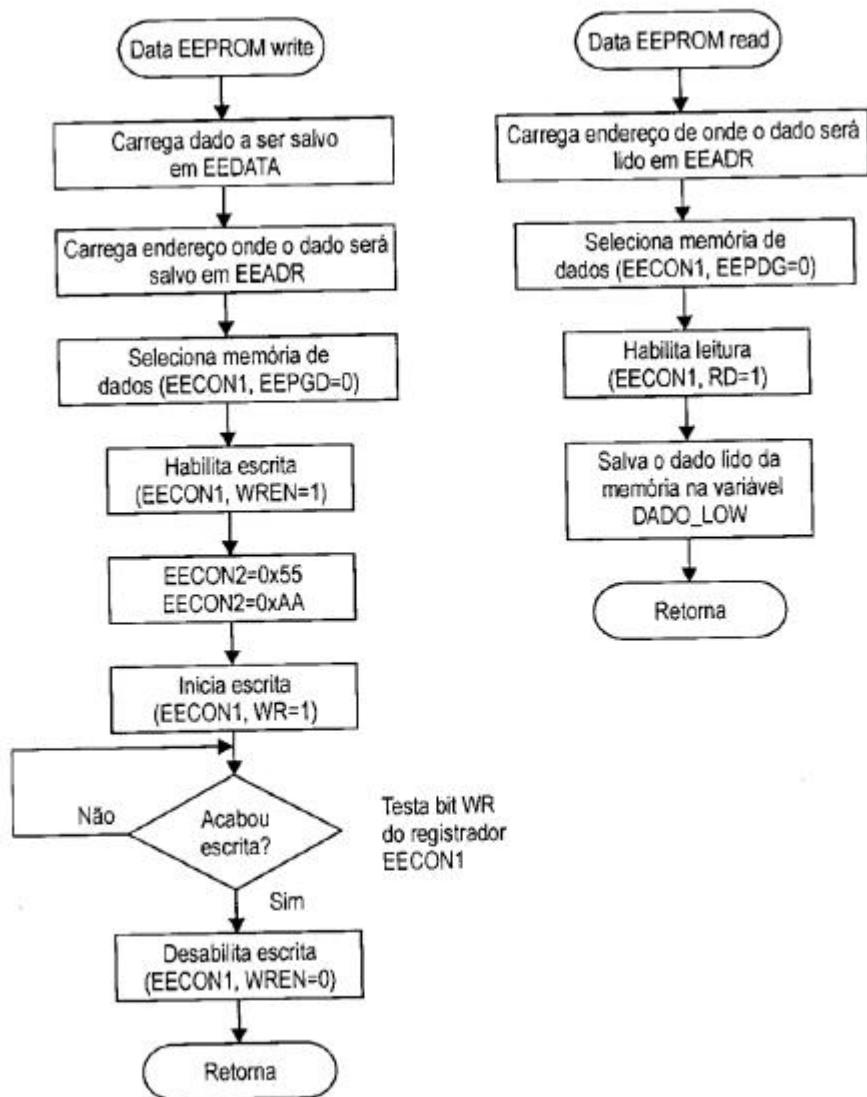


Fluxograma









```

;*****
;*          CONECTANDO O PIC - RECURSOS AVANÇADOS      *
;
;*          EXEMPLO 7                                     *
;
;*          NICOLÁS CÉSAR LAVINIA e DAVID JOSÉ DE SOUZA  *
;
;*          VERSÃO : 2.0                                     *
;*          DATA : 24/02/2003                               *
;*****
```

```

;*****
;*          DESCRIÇÃO GERAL      *
;*          Conectando o PIC 16F877A - Recursos Avançados*
```

```

;*
*  

;*****  

;  

; ESTE EXEMPLO FOI ELABORADO PARA EXPLICAR O FUNCIONAMENTO DA LEITURA/ESCRITA  

; TANTO NA MEMÓRIA DE DADOS QUANTO NA MEMÓRIA DE PROGRAMA.  

;  

;*****  

;  

;*          CONFIGURAÇÕES PARA GRAVAÇÃO           *  

;*****  

;  

;  

;*****  

; _CONFIG_CP_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &  

; _PWRTE_ON & _WDT_ON & _XT_OSC  

;  

;*****  

;  

;*          DEFINIÇÃO DAS VARIÁVEIS           *  

;*****  

;  

; ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO LOGO NO INÍCIO DO BANCO 0  

;  

CBLOCK 0X20           ; POSIÇÃO INICIAL DA RAM  

;  

FILTRO_BOTOES        ; FILTRO PARA RUIDOS  

TEMPO_TURBO          ; TEMPORIZADOR P/ TURBO DAS TECLAS  

;  

TEMPO1  

TEMPO0           ; CONTADORES P/ DELAY  

;  

FLAG              ; FLAG DE USO GERAL  

;  

VALOR_DADOS         ; VALOR ARMAZENADO NA MEMÓRIA  

; DE DADOS (8 BITS)  

;  

VALOR_PROG_HIGH      ; VALOR ARMAZENADO NA MEMÓRIA  

VALOR_PROG_LOW       ; DE PROGRAMAS (14 BITS)  

;  

AUX                ; REGISTRADOR AUXILIAR DE USO GERAL

```

```
        ENDERECO_HIGH      ; REGISTRADORES DE ENDEREÇO PARA
        ENDERECO_LOW       ; ACESSO À MEMÓRIA DE DADOS E PROGRAMA
                           ; MAPEADOS NO BANCO 0 DA RAM

        DADO_LOW_00         ; REGISTRADORES DE DADOS PARA
        DADO_HIGH_00         ; ACESSO À MEMÓRIA DE DADOS E PROGRAMA
                           ; MAPEADOS NO BANCO 0 DA RAM

        DADO_LOW_01         ; REGISTRADORES DE DADOS PARA
        DADO_HIGH_01         ; ACESSO À MEMÓRIA DE DADOS E PROGRAMA
                           ; MAPEADOS NO BANCO 0 DA RAM

        DADO_LOW_10         ; REGISTRADORES DE DADOS PARA
        DADO_HIGH_10         ; ACESSO À MEMÓRIA DE DADOS E PROGRAMA
                           ; MAPEADOS NO BANCO 0 DA RAM

        DADO_LOW_11         ; REGISTRADORES DE DADOS PARA
        DADO_HIGH_11         ; ACESSO À MEMÓRIA DE DADOS E PROGRAMA
                           ; MAPEADOS NO BANCO 0 DA RAM

ENDC
```

```
;*****
;*      DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC      *
;*****  
;  
; O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE
; OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE
; DE REDIGITAÇÃO.
```

```
#INCLUDE <P16F877A.INC>      ; MICROCONTROLADOR UTILIZADO
```

```
;*****
;*      DEFINIÇÃO DOS BANCOS DE RAM      *
;*****  
;  
; OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR
; ENTRE OS BANCOS DE MEMÓRIA.
```

```

#define BANK1  BSF      STATUS,RP0      ; SELEciona BANK1 DA MEMORIA RAM
#define BANK0  BCF      STATUS,RP0      ; SELEciona BANK0 DA MEMORIA RAM

;*****
;*          CONSTANTES INTERNAS          *
;*****

; A DEFINIÇÃO DE CONSTANTES FACILITA A PROGRAMAÇÃO E A MANUTENÇÃO.

FILTRO_TECLA EQU .200           ; FILTRO P/ EVITAR RUIDOS DOS BOTÕES

TURBO_TECLA EQU .60            ; TEMPORIZADOR P/ TURBO DAS TECLAS

END_MEM_DADO EQU 0X10          ; ENDEREÇO P/ LEITURA E GRAVAÇÃO
                                ; NA MEMÓRIA DE DADOS

END_MEM_PROG_H     EQU 0X08      ; ENDEREÇO P/ LEITURA E GRAVAÇÃO
END_MEM_PROG_L     EQU 0X00      ; NA MEMÓRIA DE PROGRAMA

;*****
;*          DECLARAÇÃO DOS FLAGs DE SOFTWARE          *
;*****


; A DEFINIÇÃO DE FLAGs AJUDA NA PROGRAMAÇÃO E ECONOMIZA MEMÓRIA RAM.

#define TIPO_MEMORIA FLAG,0        ; DEFINE A MEMORIA QUE ESTA SENDO
                                ; UTILIZADA
                                ; 1 -> MEMORIA DE PROGRAMA
                                ; 0 -> MEMORIA DE DADOS

;*****
;*          ENTRADAS          *
;*****


; AS ENTRADAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

```

#DEFINE BOTAO_0 PORTB,0 ; ESTADO DO BOTÃO 0
; 1 -> LIBERADO
; 0 -> PRESSIONADO

#DEFINE BOTAO_1 PORTB,1 ; ESTADO DO BOTÃO 1
; 1 -> LIBERADO
; 0 -> PRESSIONADO

#DEFINE BOTAO_2 PORTB,2 ; ESTADO DO BOTÃO 2
; 1 -> LIBERADO
; 0 -> PRESSIONADO

#DEFINE BOTAO_3 PORTB,3 ; ESTADO DO BOTÃO 3
; 1 -> LIBERADO
; 0 -> PRESSIONADO

;*****

;* SAÍDAS *

;*****

; AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#DEFINE DISPLAY PORTD ; BARRAMENTO DE DADOS DO DISPLAY

#DEFINE RS PORTE,0 ; INDICA P/ O DISPLAY UM DADO OU COMANDO
; 1 -> DADO
; 0 -> COMANDO

#DEFINE ENABLE PORTE,1 ; SINAL DE ENABLE P/ DISPLAY
; ATIVO NA BORDA DE DESCIDA

;*****

;* VETOR DE RESET DO MICROCONTROLADOR *

;*****

; POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA
Conectando o PIC 16F877A - Recursos Avançados

```
ORG 0X0000 ; ENDEREÇO DO VETOR DE RESET
GOTO CONFIG ; PULA PARA CONFIG DEVIDO A REGIÃO
; DESTINADA AS ROTINAS SEGUINTE
```

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

```
;%* ROTINA DE DELAY (DE 1MS ATÉ 256MS) *
;*****  
; ESTA É UMA ROTINA DE DELAY VARIÁVEL, COM DURAÇÃO DE 1MS X O VALOR PASSADO
; EM WORK (W).
```

DELAY_MS

```
MOVWF TEMPO1 ; CARREGA TEMPO1 (UNIDADES DE MS)
MOVLW .250
MOVWF TEMPO0 ; CARREGA TEMPO0 (P/ CONTAR 1MS)

CLRWDT ; LIMPA WDT (PERDE TEMPO)
DECFSZ TEMPO0,F ; FIM DE TEMPO0 ?
GOTO $-2 ; NÃO - VOLTA 2 INSTRUÇÕES
; SIM - PASSOU-SE 1MS
DECFSZ TEMPO1,F ; FIM DE TEMPO1 ?
GOTO $-6 ; NÃO - VOLTA 6 INSTRUÇÕES
; SIM
RETURN ; RETORNA
```

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

```
;%* ROTINA DE ESCRITA DE UM CARACTER NO DISPLAY *
;*****  
; ESTA ROTINA ENVIA UM CARACTER PARA O MÓDULO DE LCD. O CARACTER A SER
; ESCRITO DEVE SER COLOCADO EM WORK (W) ANTES DE CHAMAR A ROTINA.
```

ESCREVE

```
MOVWF DISPLAY ; ATUALIZA DISPLAY (PORTD)
NOP ; PERDE 1US PARA ESTABILIZAÇÃO
BSF ENABLE ; ENVIA UM PULSO DE ENABLE AO DISPLAY
Conectando o PIC 16F877A - Recursos Avançados
```

```

GOTO    $+1          ;.

BCF     ENABLE        ;.

MOVlw .1

CALL    DELAY_MS      ; DELAY DE 1MS

RETURN             ; RETORNA

;*****ROTINA DE ESCRITA LINHA 1 DO LCD*****
; ESTA ROTINA ESCREVE A LINHA 1 DA TELA PRINCIPAL DO LCD, COM A FRASE:
; LINHA 1 - "M.DADOS M.PROG."

ATUALIZA_TELA_LINHA_1

BCF     RS            ; SELECCIONA O DISPLAY P/ COMANDOS

MOVlw 0X80           ; COMANDO PARA POSICIONAR O CURSOR

CALL    ESCREVE       ; LINHA 0 / COLUNA 0

BSF     RS            ; SELECCIONA O DISPLAY P/ DADOS

; COMANDOS PARA ESCREVER AS
; LETRAS DE "M.DADOS M.PROG."

MOVlw 'M'
CALL    ESCREVE
MOVlw '.'

CALL    ESCREVE
MOVlw 'D'
CALL    ESCREVE
MOVlw 'A'
CALL    ESCREVE
MOVlw 'D'
CALL    ESCREVE
MOVlw 'O'
CALL    ESCREVE
MOVlw 'S'
CALL    ESCREVE

```

Conectando o PIC 16F877A - Recursos Avançados

```

MOVlw ""
CALL    ESCREVE
MOVlw ""
CALL    ESCREVE
MOVlw 'M'
CALL    ESCREVE
MOVlw '!'
CALL    ESCREVE
MOVlw 'P'
CALL    ESCREVE
MOVlw 'R'
CALL    ESCREVE
MOVlw 'O'
CALL    ESCREVE
MOVlw 'G'
CALL    ESCREVE
MOVlw '!'
CALL    ESCREVE

RETURN           ; RETORNA

;*****
;*      ROTINA DE ESCRITA LINHA 2 DO LCD
*      *****
; ESTA ROTINA ESCREVE A LINHA 2 DA TELA PRINCIPAL DO LCD.
; A ROTINA LEVA EM CONTA TODAS AS VARIÁVEIS PERTINENTES P/ FORMAR A LINHA 2.

ATUALIZA_TELA_LINHA_2

BCF    RS           ; SELECCIONA O DISPLAY P/ COMANDO
MOVlw 0XC1           ; COMANDO PARA POSICIONAR O CURSOR
CALL   ESCREVE        ; LINHA 1 / COLUNA 1

BSF    RS           ; SELECCIONA O DISPLAY P/ DADOS

MOVlw '>'
```

```

BTFSC  TIPO_MEMORIA ; ESTÁ UTILIZANDO A MEMÓRIA DE DADOS ?

MOVLW  '' ; NÃO - ESCREVE ESPAÇO EM BRANCO

CALL    ESCREVE ; SIM - ESCREVE ">" NO DISPLAY

SWAPF  VALOR_DADOS,W ; INVERTE NIBLE DO VALOR_DADOS

ANDLW  B'00001111' ; MASCARA BITS MAIS SIGNIFICATIVOS

MOVWF  AUX ; SALVA EM AUXILIAR

MOVLW  0X0A

SUBWF  AUX,W ; AUX - 10d (ATUALIZA FLAG DE CARRY)

MOVLW  0X30 ; CARREGA WORK COM 30h

BTFSC  STATUS,C ; RESULTADO É POSITIVO? (É UMA LETRA?)

MOVLW  0X37 ; SIM - CARREGA WORK COM 37h

; NÃO - WORK FICA COM 30h (NÚMERO)

ADDWF  AUX,W ; SOMA O WORK AO AUXILIAR

; (CONVERSÃO ASCII)

CALL    ESCREVE ; ENVIA CARACTER AO DISPLAY LCD

MOVF   VALOR_DADOS,W ; CARREGA WORK COM VALOR_DADOS

ANDLW  B'00001111' ; MASCARA BITS MAIS SIGNIFICATIVOS

MOVWF  AUX ; SALVA EM AUXILIAR

MOVLW  0X0A

SUBWF  AUX,W ; AUX - 10d (ATUALIZA FLAG DE CARRY)

MOVLW  0X30 ; CARREGA WORK COM 30h

BTFSC  STATUS,C ; RESULTADO É POSITIVO? (É UMA LETRA?)

MOVLW  0X37 ; SIM - CARREGA WORK COM 37h

; NÃO - WORK FICA COM 30h (NÚMERO)

ADDWF  AUX,W ; SOMA O WORK AO AUXILIAR

; (CONVERSÃO ASCII)

CALL    ESCREVE ; ENVIA CARACTER AO DISPLAY LCD

MOVLW  'h'

CALL    ESCREVE ; ESCREVE "h" NO DISPLAY

```

MOVLW '<'
BTFSC TIPO_MEMORIA ; ESTÁ UTILIZANDO A MEMÓRIA DE DADOS ?
MOVLW '' ; NÃO - ESCREVE ESPAÇO EM BRANCO
CALL ESCREVE ; SIM - ESCREVE "<" NO DISPLAY

MOVLW ''
CALL ESCREVE
MOVLW ''
CALL ESCREVE
MOVLW ''
CALL ESCREVE ; ESPAÇOS EM BRANCO

MOVLW '>'
BTFSS TIPO_MEMORIA ; ESTÁ UTILIZANDO A MEMÓRIA DE PROGRAMA?
MOVLW '' ; NÃO - ESCREVE ESPAÇO EM BRANCO
CALL ESCREVE ; SIM - ESCREVE ">" NO DISPLAY

SWAPF VALOR_PROG_HIGH,W ; INVERTE NIBLE DO VALOR_PROG_HIGH
ANDLW B'00001111' ; MASCARA BITS MAIS SIGNIFICATIVOS
MOVWF AUX ; SALVA EM AUXILIAR

MOVLW 0X0A
SUBWF AUX,W ; AUX - 10d (ATUALIZA FLAG DE CARRY)
MOVLW 0X30 ; CARREGA WORK COM 30h
BTFSC STATUS,C ; RESULTADO É POSITIVO? (É UMA LETRA?)
MOVLW 0X37 ; SIM - CARREGA WORK COM 37h
; NÃO - WORK FICA COM 30h (NÚMERO)
ADDWF AUX,W ; SOMA O WORK AO AUXILIAR
; (CONVERSÃO ASCII)
CALL ESCREVE ; ENVIAM CARACTER AO DISPLAY LCD

MOVF VALOR_PROG_HIGH,W ; CARREGA WORK COM VALOR_PROG_HIGH
ANDLW B'00001111' ; MASCARA BITS MAIS SIGNIFICATIVOS
MOVWF AUX ; SALVA EM AUXILIAR

```

MOVlw 0x0A
SUBwf AUX,W           ; AUX - 10d (ATUALIZA FLAG DE CARRY)
MOVlw 0x30             ; CARREGA WORK COM 30h
BTfsc STATUS,C        ; RESULTADO É POSITIVO? (É UMA LETRA?)
MOVlw 0x37             ; SIM - CARREGA WORK COM 37h
                                      ; NÃO - WORK FICA COM 30h (NÚMERO)
ADDwf AUX,W           ; SOMA O WORK AO AUXILIAR
                                      ; (CONVERSÃO ASCII)
CALL    ESCREVE        ; ENVIA CARACTER AO DISPLAY LCD

SWAPF VALOR_PROG_LOW,W ; INVERTE NIBLE DO VALOR_PROG_LOW
ANDlw B'00001111'       ; MASCARA BITS MAIS SIGNIFICATIVOS
MOVwf AUX              ; SALVA EM AUXILIAR

MOVlw 0x0A
SUBwf AUX,W           ; AUX - 10d (ATUALIZA FLAG DE CARRY)
MOVlw 0x30             ; CARREGA WORK COM 30h
BTfsc STATUS,C        ; RESULTADO É POSITIVO? (É UMA LETRA?)
MOVlw 0x37             ; SIM - CARREGA WORK COM 37h
                                      ; NÃO - WORK FICA COM 30h (NÚMERO)
ADDwf AUX,W           ; SOMA O WORK AO AUXILIAR
                                      ; (CONVERSÃO ASCII)
CALL    ESCREVE        ; ENVIA CARACTER AO DISPLAY LCD

MOVf  VALOR_PROG_LOW,W ; CARREGA WORK COM VALOR_PROG_LOW
ANDlw B'00001111'       ; MASCARA BITS MAIS SIGNIFICATIVOS
MOVwf AUX              ; SALVA EM AUXILIAR

MOVlw 0x0A
SUBwf AUX,W           ; AUX - 10d (ATUALIZA FLAG DE CARRY)
MOVlw 0x30             ; CARREGA WORK COM 30h
BTfsc STATUS,C        ; RESULTADO É POSITIVO? (É UMA LETRA?)
MOVlw 0x37             ; SIM - CARREGA WORK COM 37h
                                      ; NÃO - WORK FICA COM 30h (NÚMERO)
ADDwf AUX,W           ; SOMA O WORK AO AUXILIAR

```

Conectando o PIC 16F877A - Recursos Avançados

; (CONVERSÃO ASCII)

CALL	ESCREVE	; ENVIA CARACTER AO DISPLAY LCD
MOVlw 'h'		
CALL	ESCREVE	; ESCREVE "h" NO DISPLAY
MOVlw '<'		
BTFS	TIPO_MEMORIA	; ESTÁ UTILIZANDO A MEMÓRIA DE PROGRAMA?
MOVlw ''		; NÃO - ESCREVE ESPAÇO EM BRANCO
CALL	ESCREVE	; SIM - ESCREVE "<" NO DISPLAY

RETURN

;*: ROTINA DE ESCRITA NA MEMÓRIA DE DADOS *

; ESTA ROTINA ESCREVE UM DADO (8 BITS) NA MEMÓRIA DE DADOS (E2PROM).
; O DADO A SER GRAVADO DEVE SER PASSADO PELO REGISTRADOR DADO_LOW_00.
; O REGISTRADOR DADO_HIGH_00 NÃO É UTILIZADO POIS A MEMÓRIA É DE 8 BITS.
; O ENDEREÇO DEVE SER PASSADO PELO REGISTRADOR ENDERECO_LOW.
; O REGISTRADOR ENDERECO_HIGH NÃO É UTILIZADO, POIS A MEMÓRIA TEM 256 ENDER.

DATA_EEPROM_WRITE

MOVF	DADO_LOW_00,W	; CARREGA NO WORK DADO P/ SER GRAVADO
BANKSEL	EEDATA	; ALTERA P/BANK DO REGISTRADOR
EEDATA		
MOVWF	EEDATA	; SALVA DADO A SER GRAVADO EM EEDATA
; (CARREGA DADO NO REGISTRADOR		
; CORRETO DO BANCO 2 DA RAM A PARTIR		
; DO REGISTRADOR DE USUÁRIO MAPEADO		
; NO BANCO 0 DA RAM)		

BANKSEL	ENDERECO_LOW	; ALTERA P/BANK DO REGIST.
ENDERECO_LOW		

MOVF	ENDERECO_LOW,W	; CARREGA NO WORK O ENDEREÇO DE
DESTINO		

BANKSEL EEADR	EEADR	; ALTERA P/ BANK DO REGISTRADOR
MOVWF EEADR ; SALVA ENDEREÇO EM EEADR		
; (CARREGA ENDEREÇO NO REGISTRADOR		
; CORRETO DO BANCO 2 DA RAM A PARTIR		
; DO REGISTRADOR DE USUÁRIO MAPEADO		
; NO BANCO 0 DA RAM)		
BANKSEL EECON1 ; ALTERA P/ BANK DO REGISTRADOR		
EECON1		
BCF EECON1,EEPGD	; APONTA P/ MEMÓRIA DE DADOS	
BSF EECON1,WREN	; HABILITA ESCRITA	
MOVLW 0X55		
MOVWF EECON2 ; ESCREVE 0X55 EM EECON2 (OBIGATÓRIO)		
MOVLW 0XAA		
MOVWF EECON2 ; ESCREVE 0XAA EM EECON2 (OBIGATÓRIO)		
BSF EECON1,WR	; INICIA ESCRITA	
BTFSC EECON1,WR ; ACABOU ESCRITA ?		
GOTO \$-1	; NÃO - AGUARDA FIM DA ESCRITA	
; SIM		
BCF EECON1,WREN	; DESABILITA ESCRITAS NA MEMÓRIA	
BANKSEL 0X20 ; VOLTA P/ BANK0		
RETURN ; RETORNA		

;*****

.* ROTINA DE LEITURA NA MEMÓRIA DE DADOS *

;*****

; ESTA ROTINA LÊ UM DADO (8 BITS) DA MEMÓRIA DE DADOS (E2PROM).

; O DADO A SER LIDO É RETORNADO NO REGISTRADOR DADO_LOW_00.

; O REGISTRADOR DADO_HIGH_00 NÃO É UTILIZADO POIS A MEMÓRIA É DE 8 BITS.

; O ENDEREÇO DEVE SER PASSADO PELO REGISTRADOR ENDERECO_LOW.

; O REGISTRADOR ENDERECO_HIGH NÃO É UTILIZADO, POIS A MEMÓRIA TEM 256 ENDER.

DATA_EEPROM_READ

```
        MOVF    ENDERECO_LOW,W           ; CARREGA NO WORK O ENDEREÇO DE  
DESTINO  
  
        BANKSEL      EEADR            ; ALTERA P/BANK DO REGISTRADOR  
EEADR  
  
        MOVWF  EEADR                ; SALVA ENDEREÇO EM EEADR  
  
                                ; (CARREGA ENDEREÇO NO REGISTRADOR  
                                ; CORRETO DO BANCO 2 DA RAM A PARTIR  
                                ; DO REGISTRADOR DE USUÁRIO MAPEADO  
                                ; NO BANCO 0 DA RAM)  
  
        BANKSEL      EECON1           ; ALTERA P/BANK DO REGISTRADOR  
EECON1  
  
        BCF     EECON1,EEPGD          ; APONTA P/MEMÓRIA DE DADOS  
  
        BSF     EECON1,RD             ; HABILITA LEITURA  
  
        BANKSEL      EEDATA           ; ALTERA P/BANK DO REGISTRADOR  
EEDATA  
  
        MOVF    EEDATA,W              ; SALVA DADO LIDO NO WORK  
  
        BANKSEL DADO_LOW_00          ; ALTERA P/BANK DO REGIST. DADO_LOW_00  
  
        MOVWF  DADO_LOW_00            ; SALVA DADO LIDO EM DADO_LOW_00  
  
                                ; (SALVA DADO LIDO NO REGISTRADOR  
                                ; DE USUÁRIO MAPEADO NO BANCO 0 DA RAM  
                                ; A PARTIR DO REGISTRADOR UTILIZADO  
                                ; PELO MICROCONTROLADOR MAPEADO  
                                ; NO BANCO 2 DA RAM)  
  
        RETURN                   ; RETORNA
```

;*****

;* ROTINA DE ESCRITA NA MEMÓRIA DE PROGRAMA *

;*****

; A ESCRITA NA MEMÓRIA DE PROGRAMA É FEITA DE 4 EM 4 WORDS OU DE 8 EM 8 BYTES
; OBRIGATORIAMENTE. O ENDEREÇO DEVE OBRIGATORIAMENTE ESTAR ALINHADO, OU SEJA,
; O ENDEREÇO INICIAL DEVERÁ SEMPRE TER OS ÚLTIMOS DOIS BITS EM 00. DESTA FORMA,
; SEMPRE A ESCRITA NA MEMÓRIA DE PROGRAMA É FEITA NOS ENDEREÇOS COM FINAIS 00,
; 01, 10 E 11, COMPLETANDO ASSIM 4 WORDS.

; ESTA ROTINA ESCREVE QUATRO WORDS (14 BITS) NA MEMÓRIA DE PROGRAMA.
 ; OS VAORES A SEREM SALVOS DEVEM SER PASSADOS PELOS REGISTRADORES
 ; DADO_HIGH_00:DADO_LOW_00, DADO_HIGH_01:DADO_LOW_01,
 ; DADO_HIGH_10:DADO_LOW_10 E DADO_HIGH_11:DADO_LOW_11.
 ; O ENDEREÇO DEVE SER PASSADO PELOS REGIST. ENDERECO_HIGH E ENDERECO_LOW.

FLASH_PROGRAM_WRITE

```

      MOVF  ENDERECO_HIGH,W           ; CARREGA NO WORK O ENDEREÇO DE
DESTINO

      BANKSEL      EEADRH          ; ALTERA P/ BANK DO REGISTRADOR
EEADH

      MOVWF EEADRH                ; SALVA ENDEREÇO EM EEADH

      BANKSEL      ENDERECO_LOW       ; ALTERA P/BANK DO REGIST.
ENDERECO_LOW

      MOVF  ENDERECO_LOW,W          ; CARREGA NO WORK O ENDEREÇO DE
DESTINO

      ANDLW  B'11111100'          ; MASCARA PARA ZERAR OS ÚLTIMOS DOIS BIT

      BANKSEL      EEADR           ; ALTERA P/ BANK DO REGISTRADOR
EEADR

      MOVWF EEADR                ; SALVA ENDEREÇO EM EEADR

                                ; (CARREGA ENDEREÇO NOS REGISTRADOS
                                ; CORRETOS DO BANCO 2 DA RAM A PARTIR
                                ; DOS REGISTRADORES DE USUÁRIO MAPEADOS
                                ; NO BANCO 0 DA RAM)

      MOVLW  DADO_LOW_00           ; CARREGA NO W ENDEREÇO DO REGISTRADOR
DADO_HIGH_00

      MOVWF FSR                  ; SALVA O ENDEREÇO DO REGISTRADOR NO FSR

```

FLASH_PROGRAM_WRITE_2

```

      BANKSEL      EEDATA          ; ALTERA P/ BANK DO REGISTRADOR
EEDATA

      MOVF  INDF,W               ; CARREGA NO W O VALOR A SER SALVO

      MOVWF EEDATA              ; SALVA DADO A SER GRAVADO EM EEDATA

      INCF   FSR,F               ; INCREMENTA PONTEIRO

      MOVF  INDF,W               ; CARREGA NO W O VALOR A SER SALVO

      MOVWF EEDATH              ; SALVA DADO A SER GRAVADO EM EEDATH

      INCF   FSR,F               ; INCREMENTA PONTEIRO

```

BANKSEL EECON1 ; ALTERA P/ BANK DO REGISTRADOR
EECON1

BSF EECON1,EEPGD ; APONTA P/ MEMÓRIA DE PROGRAMA
BSF EECON1,WREN ; HABILITA ESCRITA

MOVLW 0X55

MOVWF EECON2 ; ESCREVE 0X55 EM EECON2 (OBRIGATÓRIO)

MOVLW 0XAA

MOVWF EECON2 ; ESCREVE 0XAA EM EECON2 (OBRIGATÓRIO)

BSF EECON1,WR ; INICIA ESCRITA

NOP

NOP ; NÃO OPERA

BCF EECON1,WREN ; DESABILITA ESCRITAS NA MEMÓRIA

BANKSEL EEADR ; ALTERA P/ BANK DO REGISTRADOR
EEADR

INCF EEADR,F ; INCREMENTA ENDEREÇO

MOVLW B'00000011' ; CARREGA MASCARA NO WORK

ANDWF EEADR,W ; WORK FICA COM APENAS OS ÚLTIMOS
DOIS BITS DO ENDEREÇO

BTFSZ STATUS,Z ; DEVE ESCREVER MAIS ALGUM DADO ? (WORK
DIFERENTE DE ZERO?)

GOTO FLASH_PROGRAM_WRITE_2 ; SIM - VOLTA PARA ESCRITA

; NÃO

BANKSEL EECON1 ; ALTERA P/ BANK DO REGISTRADOR
EECON1

BTFSZ EECON1,WR ; ACABOU ESCRITA ?

GOTO \$-1 ; NÃO - AGUARDA FIM DA ESCRITA
; SIM

BANKSEL 0X20 ; VOLTA P/ BANK0

RETURN ; RETORNA DA SUBROTINA

;*****

/* ROTINA DE LEITURA NA MEMÓRIA DE DADOS */

;*****

; ESTA ROTINA LÊ UM DADO (14 BITS) DA MEMÓRIA DE PROGRAMA.

; O DADO LIDO É RETORNADO NOS REGISTRADORES DADO_HIGH_00 E DADO_LOW_00

; O ENDEREÇO DEVE SER PASSADO PELOS REGIST. ENDERECO_HIGH E ENDERECO_LOW.

FLASH_PROGRAM_READ

MOVF ENDERECO_HIGH,W ; CARREGA NO WORK O ENDEREÇO DE DESTINO

BANKSEL EEADRH ; ALTERA P/ BANK DO REGISTRADOR EEADH

MOVWF EEADRH ; SALVA ENDEREÇO EM EEADH

BANKSEL ENDERECO_LOW ; ALTERA P/BANK DO REGIST. ENDERECO_LOW

MOVF ENDERECO_LOW,W ; CARREGA NO WORK O ENDEREÇO DE DESTINO

BANKSEL EEADR ; ALTERA P/ BANK DO REGISTRADOR EEADR

MOVWF EEADR ; SALVA ENDEREÇO EM EEADR

; (CARREGA ENDEREÇO NOS REGISTRADOS

; CORRETOS DO BANCO 2 DA RAM A PARTIR

; DOS REGISTRADORES DE USUÁRIO MAPEADOS

; NO BANCO 0 DA RAM)

BANKSEL EECON1 ; ALTERA P/ BANK DO REGISTRADOR EECON1

BSF EECON1,EEPGD ; APONTA P/ MEMÓRIA DE PROGRAMA

BSF EECON1,RD ; HABILITA LEITURA

NOP

NOP

BANKSEL EEDATH ; ALTERA P/ BANK DO REGISTRADOR EEDATH

MOVF EEDATH,W ; SALVA DADO LIDO NO WORK

```
BANKSEL      DADO_HIGH_00           ; ALTERA P/ BANK DO REGIST. DADO_HIGH
MOVWF DADO_HIGH_00          ; SALVA DADO LIDO EM DADO_HIGH_00
BANKSEL      EEDATA              ; ALTERA P/ BANK DO REGISTRADOR
EEDATA_00
MOVF   EEDATA,W               ; SALVA DADO LIDO NO WORK
BANKSEL      DADO_LOW_00          ; ALTERA P/ BANK DO REGIST.
DADO_LOW_00
MOVWF DADO_LOW_00            ; SALVA DADO LIDO EM DADO_LOW_00
; (SALVA DADO LIDO NOS REGISTRADORES
; DE USUÁRIO MAPEADOS NO BANCO 0 DA RAM
; A PARTIR DOS REGISTRADORES UTILIZADOS
; PELO MICROCONTROLADOR MAPEADOS
; NO BANCO 2 DA RAM)

RETURN          ; RETORNA
```

```
;*****
;*          CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE          *
;*****
; NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS
; CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA A
; MÁQUINA E AGUARDA O ESTOURO DO WDT.
```

CONFIG

```
CLRF  PORTA                ; GARANTE TODA AS SAÍDAS EM ZERO
CLRF  PORTB
CLRF  PORTC
CLRF  PORTD
CLRF  PORTE
```

```
BANK1          ; SELECCIONA BANCO 1 DA RAM
```

```
MOVLW B'11111111'
```

```
MOVWF TRISA          ; CONFIGURA I/O DO PORTA
```

```
MOVLW B'11111111'
```

Conectando o PIC 16F877A - Recursos Avançados

MOVWF TRISB ; CONFIGURA I/O DO PORTB

MOVLW B'11111101'

MOVWF TRISC ; CONFIGURA I/O DO PORTC

MOVLW B'00000000'

MOVWF TRISD ; CONFIGURA I/O DO PORTD

MOVLW B'00000100'

MOVWF TRISE ; CONFIGURA I/O DO PORTE

MOVLW B'11011111'

MOVWF OPTION_REG ; CONFIGURA OPTIONS

; PULL-UPs DESABILITADOS

; INTER. NA BORDA DE SUBIDA DO RB0

; TIMER0 INCREM. PELO CICLO DE MÁQUINA

; WDT - 1:128

; TIMER - 1:1

MOVLW B'00000000'

MOVWF INTCON ; CONFIGURA INTERRUPÇÕES

; DESABILITADA TODAS AS INTERRUPÇÕES

MOVLW B'00000111'

MOVWF ADCON1 ; CONFIGURA CONVERSOR A/D

; CONFIGURA PORTA E PORTE COMO I/O DIGITAL

BANK0 ; SELECCIONA BANCO 0 DA RAM

; AS INSTRUÇÕES A SEGUIR FAZEM COM QUE O PROGRAMA TRAVE QUANDO HOUVER UM

; RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT. DESTA FORMA,

; SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA UM ESTOURO DE WDT

; E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO START-UP DO PIC.

BTFSR STATUS,NOT_TO ; RESET POR ESTOURO DE WATCHDOG TIMER ?

Conectando o PIC 16F877A - Recursos Avançados

GOTO \$; NÃO - AGUARDA ESTOURO DO WDT

; SIM

.* INICIALIZAÇÃO DA RAM *

; ESTA ROTINA IRÁ LIMPAR TODA A RAM DO BANCO 0, INDO DE 0X20 A 0X7F.

MOVLW 0X20

MOVWF FSR ; APONTA O ENDEREÇAMENTO INDIRETO PARA
; A PRIMEIRA POSIÇÃO DA RAM

LIMPA_RAM

CLRF INDF ; LIMPA A POSIÇÃO
INCF FSR,F ; INCREMENTA O PONTEIRO P/ A PRÓX. POS.
MOVF FSR,W
XORLW 0X80 ; COMPARA O PONTEIRO COM A ÚLT. POS. +1
BTFSS STATUS,Z ; JÁ LIMPOU TODAS AS POSIÇÕES?
GOTO LIMPA_RAM ; NÃO - LIMPA A PRÓXIMA POSIÇÃO
; SIM

.* CONFIGURAÇÕES INICIAIS DO DISPLAY *

; ESTA ROTINA INICIALIZA O DISPLAY P/ COMUNICAÇÃO DE 8 VIAS, DISPLAY PARA 2
; LINHAS, CURSOR APAGADO E DESLOCAMENTO DO CURSOR À DIREITA.

INICIALIZACAO_DISPLAY

BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS

MOVLW 0X30 ; ESCREVE COMANDO 0X30 PARA

CALL ESCREVE ; INICIALIZAÇÃO

MOVLW .3

CALL DELAY_MS ; DELAY DE 3MS (EXIGIDO PELO DISPLAY)

```

MOVlw 0x30          ; ESCREVE COMANDO 0X30 PARA
CALL    ESCREVE      ; INICIALIZAÇÃO

MOVlw 0x30          ; ESCREVE COMANDO 0X30 PARA
CALL    ESCREVE      ; INICIALIZAÇÃO

MOVlw B'00111000'   ; ESCREVE COMANDO PARA
CALL    ESCREVE      ; INTERFACE DE 8 VIAS DE DADOS

MOVlw B'00000001'   ; ESCREVE COMANDO PARA
CALL    ESCREVE      ; LIMPAR TODO O DISPLAY

MOVlw .1
CALL    DELAY_MS     ; DELAY DE 1MS

MOVlw B'00001100'   ; ESCREVE COMANDO PARA
CALL    ESCREVE      ; LIGAR O DISPLAY SEM CURSOR

MOVlw B'00000110'   ; ESCREVE COMANDO PARA INCREM.
CALL    ESCREVE      ; AUTOMÁTICO À DIREITA

BSF     RS           ; SELECCIONA O DISPLAY P/ DADOS

```

```

;*****
;*          INICIALIZAÇÃO DA RAM
;* *****
; ESTE TRECHO DO PROGRAMA LÊ OS DADOS DAS MEMÓRIAS (E2PROM E FLASH) E
; ATUALIZA A RAM.

```

```

LE_MEMORY_PROGRAMA

MOVlw END_MEM_PROG_H
MOVWF ENDERECO_HIGH
MOVlw END_MEM_PROG_L
MOVWF ENDERECO_LOW       ; CARREGA ENDEREÇO P/ LEITURA

```

```

CALL    FLASH_PROGRAM_READ ; CHAMA ROTINA P/ LER DADO

MOVF   DADO_HIGH_00,W
MOVWF  VALOR_PROG_HIGH
MOVF   DADO_LOW_00,W
MOVWF  VALOR_PROG_LOW           ; SALVA O DADO LIDO EM
; VALOR_PROG_HIGH E VALOR_PROG_LOW

LE_MEMORY_DADOS

MOVLW END_MEM_DADO
MOVWF ENDERECO_LOW           ; CARREGA ENDERECHO P/ LEITURA

CALL    DATA EEPROM_READ      ; CHAMA ROTINA P/ LER DADO

MOVF   DADO_LOW_00,W
MOVWF  VALOR_DADOS           ; SALVA DADO LIDO EM VALOR_DADOS

;*****
;*          ROTINA DE ESCRITA DA TELA PRINCIPAL          *
;***** 

; ESTA ROTINA ESCREVE A TELA PRINCIPAL DO PROGRAMA, COM AS FRASES:
; LINHA 1 - "M.DADOS M.PROG."
; LINHA 2 - ">xxh< xxxxh"


```

```
CALL    ATUALIZA_TELA_LINHA_1 ; ATUALIZA TELA LINHA 1 DO LCD
```

```
CALL    ATUALIZA_TELA_LINHA_2 ; ATUALIZA TELA LINHA 2 DO LCD
```

```
;*****
;*          VARREDURA DOS BOTÕES          *
;***** 

; ESTA ROTINA VERIFICA SE ALGUM BOTÃO ESTÁ PRESSIONADO E CASO AFIRMATIVO
; DESVIA PARA O TRATAMENTO DO MESMO.
```

VARRE

Conectando o PIC 16F877A - Recursos Avançados

CLRWDT ; LIMPA WATCHDOG TIMER

; ***** VERIFICA ALGUM BOTÃO PRESSIONADO *****

VARRE_BOTOES

BTFS S BOTAO_0 ; O BOTÃO 0 ESTA PRESSIONADO ?

GOTO TRATA_BOTAO_0 ; SIM - PULA P/ TRATA_BOTAO_0

; NÃO

BTFS S BOTAO_1 ; O BOTÃO 1 ESTA PRESSIONADO ?

GOTO TRATA_BOTAO_1 ; SIM - PULA P/ TRATA_BOTAO_1

; NÃO

BTFS S BOTAO_2 ; O BOTÃO 2 ESTA PRESSIONADO ?

GOTO TRATA_BOTAO_2 ; SIM - PULA P/ TRATA_BOTAO_2

; NÃO

BTFS S BOTAO_3 ; O BOTÃO 3 ESTA PRESSIONADO ?

GOTO TRATA_BOTAO_3 ; SIM - PULA P/ TRATA_BOTAO_3

; NÃO

; ***** FILTRO P/ EVITAR RUIDOS *****

MOVLW FILTRO_TECLA ; CARREGA O VALOR DE FILTRO_TECLA

MOVWF FILTRO_BOTOES ; SALVA EM FILTRO_BOTOES

; RECARREGA FILTRO P/ EVITAR RUIDOS

; NOS BOTÕES

MOVLW .1

MOVWF TEMPO_TURBO ; CARREGA TEMPO DO TURBO DAS TECLAS

; COM 1 - IGNORA O TURBO A PRIMEIRA

; VEZ QUE A TECLA É PRESSIONADA

GOTO VARRE ; VOLTA PARA VARRER TECLADO

; *****

```

;*          TRATAMENTO DOS BOTÕES      *
;*****TRATAMENTO DOS BOTÕES*****
; NESTE TRECHO DO PROGRAMA ESTÃO TODOS OS TRATAMENTOS DOS BOTÕES

;*****TRATAMENTO DO BOTÃO 0 *****
; TRATA_BOTAO_0

    MOVF   FILTRO_BOTOES,F
    BTFSC  STATUS,Z           ; FILTRO JÁ IGUAL A ZERO ?
                                ; (FUNÇÃO JA FOI EXECUTADA?)
    GOTO   VARRE             ; SIM - VOLTA P/ VARREDURA DO TECLADO
                                ; NÃO
    DECFSZ FILTRO_BOTOES,F   ; FIM DO FILTRO ? (RUIDO?)
    GOTO   VARRE             ; NÃO - VOLTA P/ VARRE
                                ; SIM - BOTÃO PRESSIONADO

    MOVLW  B'00000001'
    XORWF  FLAG,F           ; INVERTE FLAG
                                ; ALTERA A MEMÓRIA UTILIZADA

    CALL   ATUALIZA_TELA_LINHA_2 ; CHAMA ROTINA P/ ATUALIZAR LCD

    GOTO   VARRE             ; VOLTA P/ VARREDURA DOS BOTÕES

;*****TRATAMENTO DO BOTÃO 1 *****
; TRATA_BOTAO_1

    DECFSZ FILTRO_BOTOES,F   ; FIM DO FILTRO ? (RUIDO?)
    GOTO   VARRE             ; NÃO - VOLTA P/ VARRE
                                ; SIM - BOTÃO PRESSIONADO

    DECFSZ TEMPO_TURBO,F     ; FIM DO TEMPO DE TURBO ?
    GOTO   VARRE             ; NÃO - VOLTA P/ VARRE
                                ; SIM

    MOVLW  TURBO_TECLA

```

```

MOVWF TEMPO_TURBO           ; RECARREGA TEMPORIZADOR DO TURBO
                                ; DAS TECLAS

BTFSC  TIPO_MEMORIA         ; ESTÁ UTILIZANDO MEMÓRIA DE DADOS ?

GOTO    INC_MEM_PROG         ; NÃO - ENTÃO PULA P/ INC_MEM_PROG
                                ; SIM

INC_MEM_DADOS

INCF    VALOR_DADOS,F       ; INCREMENTA VALOR_DADOS

CALL    ATUALIZA_TELA_LINHA_2 ; CHAMA ROTINA P/ ATUALIZAR LCD

GOTO    VARRE                ; VOLTA P/ VARREDURA DOS BOTÕES

INC_MEM_PROG

INCF    VALOR_PROG_LOW,F     ; INCREMENTA VALOR_PROG_LOW
BTFSC  STATUS,Z              ; HOUVE ESTOURO ?
INCF    VALOR_PROG_HIGH,F    ; SIM - INCREMENTA VALOR_PROG_HIGH
                                ; NÃO

MOVLW B'00111111'
ANDWF  VALOR_PROG_HIGH,F    ; LIMITA CONTADOR DA MEMÓRIA DE
                                ; PROGRAMA EM 14 BITS

CALL    ATUALIZA_TELA_LINHA_2 ; CHAMA ROTINA P/ ATUALIZAR LCD

GOTO    VARRE                ; VOLTA P/ VARREDURA DOS BOTÕES

;***** TRATAMENTO DO BOTÃO 2 *****
TRATA_BOTAO_2

DECFSZ FILTRO_BOTOES,F      ; FIM DO FILTRO ? (RUIDO?)

GOTO    VARRE                ; NÃO - VOLTA P/ VARRE
                                ; SIM - BOTÃO PRESSIONADO

DECFSZ TEMPO_TURBO,F        ; FIM DO TEMPO DE TURBO ?
Conectando o PIC 16F877A - Recursos Avançados

```

```

GOTO    VARRE           ; NÃO - VOLTA P/ VARRE
        ; SIM

MOVlw TURBO_TECLA
MOVwf TEMPO_TURBO      ; RECARREGA TEMPORIZADOR DO TURBO
                        ; DAS TECLAS

BTfsc TIPO_MEMORIA    ; ESTÁ UTILIZANDO MEMÓRIA DE DADOS ?
GOTO    DEC_MEM_PROG    ; NÃO - ENTÃO PULA P/ DEC_MEM_PROG
        ; SIM

DEC_MEM_DADOS
DECF    VALOR_DADOS,F   ; DECREMENTA VALOR_DADOS

CALL    ATUALIZA_TELA_LINHA_2 ; CHAMA ROTINA P/ ATUALIZAR LCD

GOTO    VARRE           ; VOLTA P/ VARREDURA DOS BOTÕES

DEC_MEM_PROG
MOVlw .1
SUBwf VALOR_PROG_LOW,F  ; DECREMENTA VALOR_PROG_LOW
BTfss STATUS,C          ; HOUVE ESTOURO ?
DECF    VALOR_PROG_HIGH,F ; SIM - DECREMENTA VALOR_PROG_HIGH
                        ; NÃO

MOVlw B'00111111'
ANDwf VALOR_PROG_HIGH,F ; LIMITA CONTADOR DA MEMÓRIA DE
                        ; PROGRAMA EM 14 BITS

CALL    ATUALIZA_TELA_LINHA_2 ; CHAMA ROTINA P/ ATUALIZAR LCD

GOTO    VARRE           ; VOLTA P/ VARREDURA DOS BOTÕES

; **** TRATAMENTO DO BOTÃO 3 ****
TRATA_BOTAO_3
MOVf    FILTRO_BOTOES,F
        Conectando o PIC 16F877A - Recursos Avançados

```

```

        BTFSC STATUS,Z           ; FILTRO JÁ IGUAL A ZERO ?
                                ; (FUNÇÃO JA FOI EXECUTADA?)

        GOTO    VARRE          ; SIM - VOLTA P/ VARREDURA DO TECLADO
                                ; NÃO

        DECFSZ FILTRO_BOTOES,F      ; FIM DO FILTRO ? (RUIDO?)

        GOTO    VARRE          ; NÃO - VOLTA P/ VARRE
                                ; SIM - BOTÃO PRESSIONADO

```

; *****TRECHO DO PROGRAMA PARA GRAVAR DADOS DA RAM NA MEMÓRIA *****

GRAVA_MEMORIA_PROGRAMA

```

        MOVLW END_MEM_PROG_H
        MOVWF ENDERECO_HIGH
        MOVLW END_MEM_PROG_L
        MOVWF ENDERECO_LOW       ; CARREGA ENDEREÇO ONDE O DADO SERÁ SALVO

        MOVF    VALOR_PROG_HIGH,W
        MOVWF DADO_HIGH_00
        MOVF    VALOR_PROG_LOW,W
        MOVWF DADO_LOW_00         ; CARREGA DADO A SER SALVO
                                ; EM DADO_HIGH_00 E DADO_LOW_00
        CALL    FLASH_PROGRAM_WRITE ; CHAMA ROTINA DE GRAVAÇÃO

```

GRAVA_MEMORIA_DADOS

```

        MOVLW END_MEM_DADO
        MOVWF ENDERECO_LOW       ; CARREGA ENDEREÇO ONDE O DADO SERÁ SALVO
        MOVF    VALOR_DADOS,W
        MOVWF DADO_LOW_00         ; CARREGA DADO A SER SALVO EM DADO_LOW_00

        CALL    DATA_EEPROM_WRITE ; CHAMA ROTINA DE GRAVAÇÃO

```

```

        GOTO    VARRE          ; VOLTA P/ VARREDURA DOS BOTÕES

```

;* FIM DO PROGRAMA *

```

END                  ; FIM DO PROGRAMA
Conectando o PIC 16F877A - Recursos Avançados

```

Dicas e comentários

Para a definição das posições de memória serem gravadas, tanto para dados quanto para programa, foram definidas três constantes no sistema com as seguintes inicializações:

- **END_MEM_DADOS** = 10h: Grava a informação na memória de dados na posição 10h;
- **END_MEM_PROG_H** = 08h e **END_MEM_PROG_L** = 00h: Grava a informação na memória de programa na posição 0800h.

Para o uso das memórias foram criadas quatro rotinas, já que elas trabalham com quatro variáveis, todas elas definidas no banco 0:

Rotina	Função	Variáveis (Bank0)
DATA EEPROM_READ	Lê um byte da memória de dados.	O endereço deve ser passado em ENDERECO_LOW. O dado lido será retornado em DADO_LOW.
DATA EEPROM_WRITE	Escreve um byte na memória de dados.	O endereço deve ser passado em ENDERECO_LOW e o dado a ser escrito em DADO_LOW.
FLASH_PROGRAM_READ	Lê uma word (14 bits) da memória de programa.	O endereço deve ser passado em ENDERECO_HIGH e ENDERECO_LOW. O dado lido será retornado em DADO_HIGH e DADO_LOW
FLASH_PROGRAM_WRITE	Escreve uma word (14 bits) na memória de programa.	O endereço deve ser passado em ENDERECO_HIGH e ENDERECO_LOW e o dado a ser escrito em DADO_HIGH e DADO_LOW

Exercícios propostos

Agora que o exemplo já foi estudado e esclarecido, aproveite para gerar novos problemas e soluções, seguindo os exercícios propostos:

1. Crie um sistema no qual do lado esquerdo você continua informando o dado para a EEPROM, no centro você escolhe a posição, de 0 a 255, e do lado direito é informada o tipo de operação: E para escrita e L para leitura. O botão S1 continua alterando entre os parâmetros a serem ajustados e os botões S2 e S3 alteram o parâmetro atual. O botão S4 efetua a operação: escreve na posição de memória especificada o valor ajustado ou leitura da posição especificada para indicação do valor encontrado.
2. Repita o exercício anterior para a memória FLASH.
3. Crie um programa que copie internamente todo o código para outra posição, como por exemplo na página 1. Depois, utilize o gravador para ler a memória de programa e verificar se a operação foi executada com sucesso.

Comunicação Serial 1 - SPI e I²C

Introdução

Neste capítulo começaremos o estudo de um assunto muito procurado pelos profissionais atuais: a comunicação serial. Dissemos que estaremos começando o estudo porque o PIC possui tantos recursos voltados a ele que utilizaremos dois capítulos para completá-lo. Nesta primeira parte, estudaremos dois tipos distintos de comunicação serial, denominados SPI e I²C. Esses protocolos poderão ser utilizados para a comunicação do PIC com outros microcontroladores, com memórias externas, drives de LCD, conversores, sensores e uma infinidade de outros periféricos.

Os dois protocolos mencionados (SPI e I²C)² fazem parte de um sistema comum do PIC denominado MSSP (Master Synchronous Serial Port). Como esse sistema é comum, ele compartilha os mesmos recursos (pinos, hardware internos e registradores) entre os dois casos. Por isso só podemos utilizar um sistema de cada vez. Eles serão estudados separadamente.

Comecemos, então, conhecendo as diferenças e semelhanças básicas entre eles:

SPI	PC
Foi criado pela Motorola e seu nome significa: Serial Peripheral Interface	Foi criado pela Philips e seu nome significa: Inter-Integrated Circuit
Opera em modo Master e Slave	Opera em modo Master e Slave
Comunicação síncrona (com Clock)	Comunicação síncrona (com Clock)
A comunicação é feita por três vias: <ul style="list-style-type: none"> • Clock (SCK) • Data In (SDI) • Data Out (SDO) 	A comunicação é feita por duas vias: <ul style="list-style-type: none"> • Clock (SCL) • Data In/Out(SDA)
Não endereçável 8 bits de dados	Endereçável (7 ou 10 bits) 8 bits de dados

Teoria e recursos do PIC para SPI

Antes de mais nada precisamos saber alguns conceitos básicos sobre a comunicação serial síncrona. Como o próprio nome diz, essa comunicação exige uma via de sincronismo entre os dois componentes envolvidos (receptor e transmissor) para que haja a marcação do momento exato da

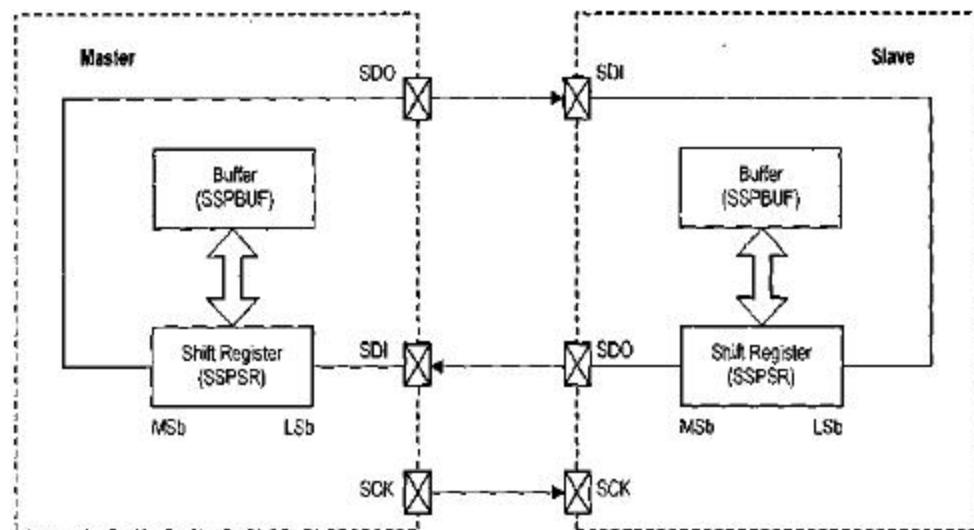
transmissão/recepção de cada bit. Isso é feito através do clock, que nada mais é que uma onda quadrada para sincronizar o sistema. Acontece que só pode existir um clock, e por isso não podemos deixar ambos os lados da comunicação responsáveis por ele. Para resolver esse problema foi criado o conceito de Master e Slave. O componente denominado Master (Mestre) será responsável pelo controle da comunicação, gerando o clock de sincronismo. Do outro lado, o componente será denominado Slave (Escravo) e ficará sempre aguardando o clock enviado pelo mestre.

Como a SPI não permite o endereçamento, a comunicação só pode ser feita entre dois pontos, sendo um deles o Master e o outro o Slave.

Como é observado na tabela comparativa apresentada anteriormente, neste modo de comunicação a ligação entre os componentes deve ser feita através de três vias:

- **Clock:** Trata-se da via de clock, que pode ser uma entrada (Slave) ou saída (Master). O SPI aceita os quatro tipos de clocks diferentes especificados para o padrão SPI. A descrição e a configuração desses tipos será vista posteriormente. No PIC 16F877A é chamado de **SCK** e está localizado no pino 18 (RC3);
- **Data In:** Trata-se da entrada de dados, ou seja, a via de recepção. No PIC 16F877A é chamada de **SDI** e está localizada no pino 23 (RC4);
- **Data Out:** Trata-se da saída de dados, ou seja, a via de transmissão. No PIC 16F877A é chamada de **SDO** e está localizada no pino 24 (RC5);

O próximo diagrama mostra a ligação entre dois componentes, que podem, por exemplo ser dois PICs:



O PIC do lado esquerdo é o Master e, por isso, é responsável pela geração do clock. Seu pino **SCK** deve ser configurado como saída. Já do lado direito o PIC é definido como Slave, ficando o pino de clock configurado como entrada.

Através do diagrama podemos também começar a entender melhor o funcionamento interno do SR. Esse protocolo de comunicação trabalha com um registrador interno (não acessível) denominado **SSPSR**. Esse registrador é do tipo rotativo (Shift Register). A cada ciclo do clock, um bit é lido da porta **SDI** é escrito nesse registrador, entrando pelo bit menos significativo e rotacionando todos os bits para a

esquerda. Ao mesmo tempo, o bit mais significativo é enviado para a porta SDO. Por isso, enquanto estamos recebendo um byte estamos também transmitindo outro. O registrador SSPBUF é o responsável pelo valor a ser transmitido e também pelo byte recebido.

A comunicação funciona então da seguinte maneira:

1. Quem manda na comunicação é o Master. Por isso, um valor qualquer é escrito no registrador **SSPBUF** deste PIC.
2. O valor então é movido para o registrador **SSPSR**.
3. Em seguida oito pulsos são gerados na saída de clock **SCK** (cada pulso transmite 1 bit).
4. Ao término dos oito pulsos o valor que estava no **SSPSR_{Master}** é trocado com o valor que estava **SSPSR_{slave}**.
5. Tanto de um lado quanto do outro o valor de **SSPSR** é então movido novamente para o registrador **SSPBUF**.
6. Também de ambos os lados flags são ativados para informar ao sistema o fim da comunicação. Os dados podem então ser acessados novamente em **SSPBUF**.
- 7.

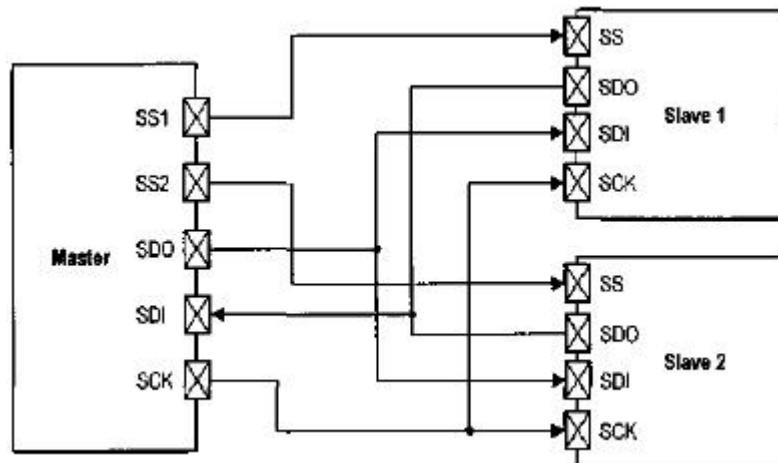
Bem simples não é mesmo? Desta forma, a comunicação é feita sempre em sentido duplo, ao mesmo tempo. Desta forma, três tipos diferentes de situações podem acontecer:

- **Master envia dado válido / Slave envia dado não-válido:** Neste caso o Master está enviando algum dado para o Slave; porém não interessa a informação que é enviada pelo Slave neste mesmo momento. É muito comum quando o Master está fazendo uma pergunta ao Slave, que terá de ser processada antes da resposta ser transmitida de volta. Neste caso, o dado recebido pelo Master ao final da transmissão deve ser descartado.
- **Master envia dado não válido / Slave envia dado válido:** Este caso é complementar ao anterior. Depois que o Master efetuou uma pergunta e o Slave a processou, uma resposta deverá ser retornada. Acontece que como é o Master quem manda no clock, ele precisará disparar uma nova transmissão para poder receber o dado do Slave.
- **Master envia dado válido / Slave envia dado válido:** Pode ser o caso mais raro, mas é usado para aumento da velocidade. Por exemplo: o Master faz uma pergunta e ao mesmo tempo recebe a resposta da pergunta anterior.

Obviamente, apesar do sistema de comunicação ser de transmissão e recepção ao mesmo tempo, nem sempre isso é necessário. Caso a comunicação seja em um só sentido, a via de transmissão do Slave (**SDO**) pode ser desativada configurando-se esse pino como entrada, a recepção do Master (**SDI**) sem uso.

Existe também um outro pino denominado **/SS (RA5)** que pode ser usado do lado Slave. Na maioria dos casos, este pino é opcional e serve como um sinal de Chip Select. Sua função é muito simples, quando este pino está em nível baixo (0), o sistema de comunicação funciona normalmente, com o pino **SDO** funcionando como saída de dados a cada pulso do clock. No entanto, se este pino for colocado em nível alto (1), o sistema de comunicação é desligado e o pino **SDO** fica flutuante. Com este recurso podemos montar uma rede de comunicação com um Master e vários Slaves, desde que o Master controle individualmente os pinos **/SS** de cada um dos Slaves. Este controle terá de ser implementado manualmente no software e deve garantir que somente um Slave está ativado de cada vez, evitando conflitos nas saídas **SDO**.

Quando configurado para Master, o pino /SS opera como um I/O normal (RA5).



Vejamos agora como configurar corretamente o sistema para efetuar uma transmissão.

A primeira coisa a ser feita diz respeito à configuração correta dos pinos, como entradas ou saídas:

Pino	Configuração
SDI	Este pino é configurado automaticamente pelo sistema como entrada .
SDO	Este pino deve ser configurado manualmente como saída através do TRISC .
SCK	Este pino deve ser configurado como saída no Master e entrada do Slave, através do TRISC .
/SS	Quando habilitado no Slave, deve ser configurado manualmente como entrada .

Depois devemos configurar a comunicação como sendo Slave ou Master. Existe mais de uma opção para cada tipo, devendo ser configuradas em **SSPCON<SSPM3:SSPM0>**:

SSPM3: SSPM0	Descrição
0000	SPI Master, com clock = Fosc / 4
0001	SPI Master, com clock = Fosc / 16
0010	SPI Master, com clock= Fosc / 64
0011	SPI Master, com clock = TMR2 / 2
0100	SPI Slave, com /SS habilitado
0101	SPI Slave, com /SS desabilitado.

Obs: As demais opções de configurações dizem respeitos aos outros modos (I C).

As quatro primeiras opções dizem respeito à escolha do modo Master, cada uma com uma freqüência diferente para o clock. Nas três primeiras, o clock é uma divisão (4,16 ou 64) da freqüência do oscilador do próprio PIC. Com isso, podemos deduzir que na freqüência máxima de 20 MHz nossa taxa de transmissão é de 5,0Mbps. Na outra opção, o clock é gerado com base na metade do **TMR2**. Assim, a base de tempo será **PR2 / 2**. O postscale não é usado para este caso.

As duas últimas opções devem ser escolhidas quando estamos configurando um PIC em modo Slave. Neste caso devemos escolher entre trabalhar ou não com o pino **/SS**. Mais para a frente será comentado um caso em que a ativação deste pino é obrigatória.

Outra configuração ajusta a condição de recebimento. Assim, o sistema pode efetuar a leitura da porta **SDI** em dois pontos distintos, conforme o valor imposto em **SSPSTAT<SMP>**:

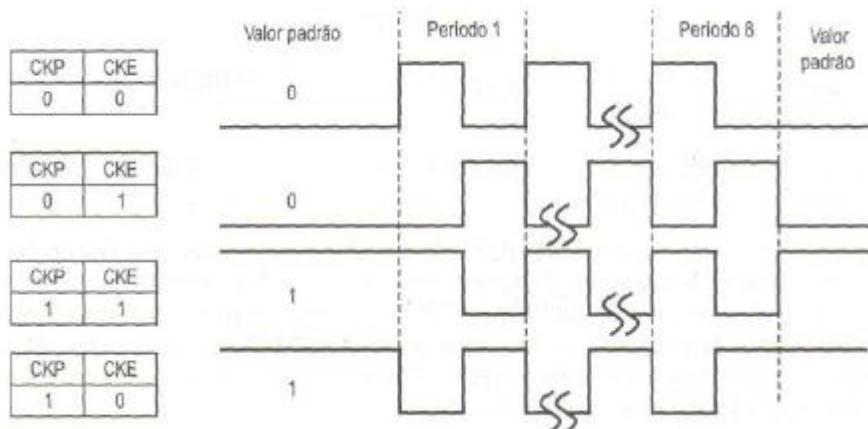
SMP	Descrição
0	A recepção é feita na borda do meio do período. Pode ser usada tanto no Master quanto no Slave.
1	A recepção é feita na borda do final do período. Só pode ser usada no Master.

O próximo passo diz respeito à escolha da forma de onda do clock. O padrão SPI disponibiliza quatro opções que podem ser ajustadas em dois bits distintos, **SSPCON<CKP>** e **SSPSTAT<CKE>**:

CKP	Descrição
0	Clock normalmente em nível baixo (0)
1	Clock normalmente em nível alto (1)

CKE	Descrição
0	Pulso do Clock no começo do período
1	Pulso do Clock no final do período

Graficamente fica muito mais fácil percebermos as diferenças entre os quatro modos resultantes da combinação desses 2 bits:



Para o Master não existe limitação quanto ao uso de qualquer um dos quatro tipos. Para o Slave, a questão é um pouco mais complicada.

Como os pinos de clock estão diretamente ligados, então o valor padrão da via (nível alto ou baixo) deve ser o mesmo. Desta forma o valor de **CKP** para o Slave deve ser obrigatoriamente o mesmo adotado para o Master.

O outro problema está em relação à transmissão/recepção do dado pelo Slave. Para o Master, o primeiro bit (o mais significativo) é colocado em **SDO** logo depois do começo do período, permanecendo lá até o final do mesmo. Desta forma, podemos considerar que o bit é transmitido no meio do período. Como ele sabe o momento exato de início do período, fica fácil também controlar a leitura da entrada **SDI** (no começo ou no fim). Para o Slave entretanto, um problema acontece quando **CKE=1**, pois não há como ele saber o início do período uma vez que não existe nenhuma borda neste ponto. Por esse motivo, para utilizarmos o Slave com **CKE=1**, obrigatoriamente teremos que habilitar o pino/**SS**. Quando o pino de **/SS** for colocado em nível baixo, o primeiro bit é colocado em **SDO** (até este momento ele estava flutuante) e o sistema fica aguardando o clock para dar continuidade na transmissão. Quanto à recepção, como o S/ave só pode operar com leitura no meio do período (**SMP=0**), a entrada **SDI** será lida logo na primeira borda do clock.

Para ligar a comunicação SPI, basta ativar o sistema SSP através do bit **SSPCON<SSPEN>**:

SSPEN	Descrição
0	Desabilita a comunicação.
1	Habilita a comunicação. Neste momento, os pinos SCK , SDO , SDI e em alguns casos o /SS , deixam de operar como I/Os convencionais.

Por último, basta escrever o dado que se deseja transmitir no registrador **SSPBUF**. Se isso for feito no Master, a transmissão será iniciada imediatamente após a escrita. Para o Slave, o sistema ficará aguardando o clock. No término dos oito pulsos do clock, o dado terá sido enviado do Master para o Slave e vice-versa, como já explicado. Os registradores **SSPBUF** serão, então, atualizados e o flag **SSPSTAT<BF>** indicará que o buffer está cheio:

BF	Descrição
0	Recebimento em operação. SSPBUF está vazio.
1	Recebimento terminado. SSPBUF carregado.

Esse bit é limpo automaticamente toda vez que o registrador **SSPBUF** é lido, indicando que o programa já processou a informação recebida.

Neste mesmo momento o bit da interrupção **PIR1<SSPIF>** é ativado. A interrupção só acontecerá no caso das suas chaves estarem ligadas.

Caso seja escrito outro valor em **SSPBUF** antes do término da transmissão/recepção, ele será ignorado, não afetando a operação. Entretanto, um bit será setado informando uma colisão de informações. Este bit é denominado **SSPCON<WCOL>** e deve ser limpo manualmente pelo programa. Por outro lado, caso o Slave receba um novo valor antes do **SSPBUF** ser lido (**BF=1**), este novo valor será perdido e o flag de overflow será ativado (**SSPCON<SSPOV>=1**). Este bit também deve ser limpo manualmente. Para o Master este flag não possui função.

Quanto ao modo SLEEP, o resultado é diferente quando aplicado ao Master ou ao Slave. No caso do Master, a comunicação será completamente interrompida, pois não é possível a geração do clock sem o oscilador estar funcionando. Por outro lado, para o Slave, a comunicação continuará funcionando e o PIC poderá acordar caso a interrupção de SSP esteja ligada.

Resumo dos registradores associados a SPI

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit1	Bit0
0Bh...	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
OCh	PIR1	PSPIIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
94h	SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
13h	SSPBUF	Buffer de transmissão/recepção							

- Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Teoria para I²C

O padrão de comunicação I²C possui características bem diferentes do SPI. Isso o torna muito mais poderoso, mas, em contrapartida, muito mais complexo também. Vejamos, então, suas principais características

- Este protocolo continua com o conceito de Master/Slave, entretanto ele permite o endereçamento de diversos pontos da rede por meio das próprias vias de comunicação. Com isso podemos ter um Master com diversos Slaves sem necessitarmos de pinos de controle adicionais. Na verdade, é possível também a estruturação de uma rede com diversos Mestres;
- A comunicação é feita somente em duas vias: Clock e Data. Como também se trata de uma comunicação síncrona, a via de clock continua sendo indispensável. Quanto aos dados, eles transitam em uma única via, tanto para a transmissão quanto para a recepção;
- Tanto o Master quanto o Slave podem transmitir e receber dados, mas o controle é sempre do Master;
- Para evitar conflitos na via de dados (**SDA**), os pinos são chaveados como entrada ou saída (impõe somente o nível baixo, forçando o GND), conforme a necessidade imposta pelo protocolo. Por esse fato, um resistor de pull-up é necessário, sendo recomendado valores de 10kΩ a 1kΩ. Os valores mais baixos (até 1kΩ) podem ser necessários para velocidades muito elevadas ou uma rede com muitos periféricos;
- O clock (**SCL**) continua sendo totalmente controlado pelo Mestre. Acontece que, nesse protocolo, pelo compartilhamento de uma única via de dados e pela possibilidade de mais de um Mestre, o clock necessário para a resposta do Escravo deve ser dado no momento correto, evitando conflitos no processamento da informação. Para que isso seja possível, a via de clock também é alterada como entrada e saída (impõe somente o nível baixo, forçando o GND) durante a transmissão, ficando em determinados momentos em um estado flutuante. Por isso, é necessário que esta linha possua um resistor de pull-up para evitar esse estado flutuante e não gerar problemas no protocolo. Este resistor deve ser de 10 kΩ a 1kΩ, adotando-se os valores mais baixos quando a velocidade de transferência é maior. O detalhamento deste funcionamento será visto mais à frente;
- Fica claro, então, que o valor padrão tanto para **SCL** quanto para **SDA** é o nível alto(1), com esses pinos em alta impedância (entradas). Aproveitamos para comentar também que os bits são lidos sempre da borda de descida do **SCL**;

- Além desse protocolo permitir a especificação do endereço, existem dois modos diferentes para que isso seja feito: 7 bits ou 10 bits. No primeiro caso é possível acessar até 128 endereços diferentes. Para o segundo caso, este número eleva-se para 1024 endereços.

Com essas explicações básicas, podemos concluir que na comunicação existem quatro situações distintas do protocolo que teremos de conhecer:

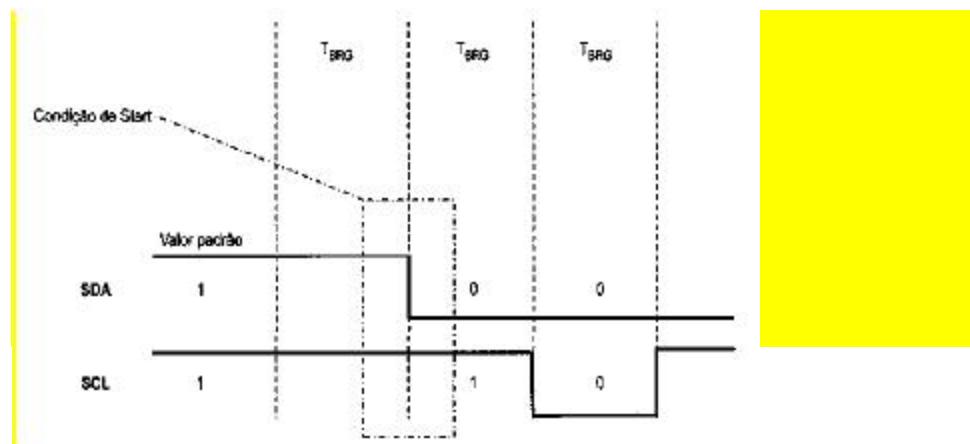
- Transmissão do Master/Recepção do Slave, com endereço de 7 bits;
- Recepção do Master/Transmissão do Slave, com endereço de 7 bits;
- Transmissão do Master/Recepção do Slave, com endereço de 10 bits;
- Recepção do Master/Transmissão do Slave, com endereço de 10 bits.

Nas próximas páginas serão mostradas quatro cartas de tempo completas, para cada uma das situações apresentadas. Antes, porém, vamos conhecer alguns outros pontos importantes deste protocolo que aparecerão nos diagramas mencionados.

Condição de Start

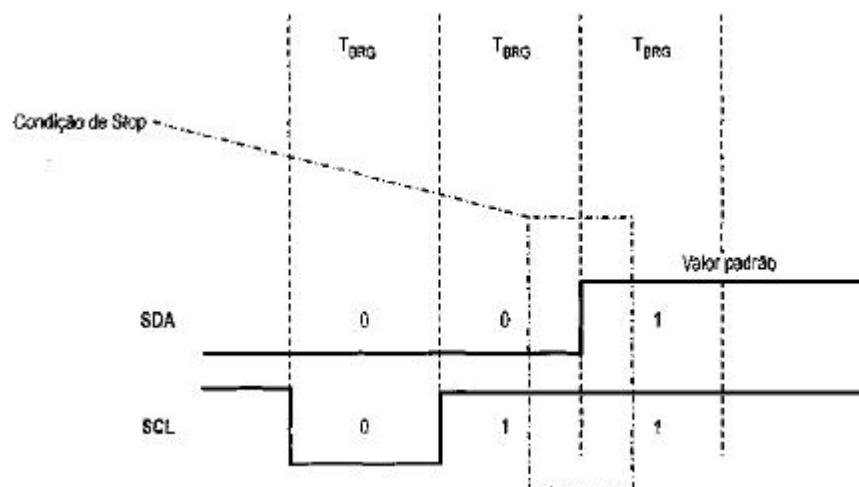
Trata-se de uma condição especial entre as vias **SDA** e **SCL** para informar à rede que uma transmissão irá começar. Essa condição é controlada pelo Mestre e todos os Escravos podem ficar prontos, esperando um dado ser transmitido. Em rede de Múltiplos Mestres, essa condição serve também para informar aos demais Mestres que a linha está ocupada.

Essa condição é atingida respeitando-se o seguinte diagrama: em que T_{BRG} é o tempo equivalente a um pulso de clock.



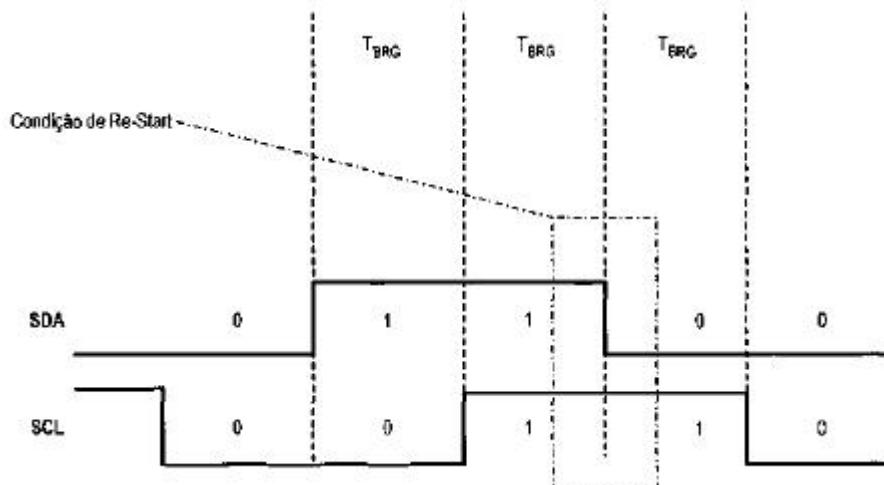
Condição de Stop

Outro estado especial dos pinos **SDA** e **SCL**, com efeito contrário à condição de Start, pois informa a toda a rede que a transmissão terminou, ficando a mesma disponível. Uma condição de Start obriga uma posterior condição de *Stop*, para que o sistema não fique travado.



Condição de Re-Start

É uma condição muito semelhante a de Start, mas que pode ser executada antes do Stop. Isso será necessário, por exemplo, para a implementação da comunicação com endereço de 10 bits ou nos casos em que precisamos enviar novamente o primeiro byte antes de finalizarmos a comunicação através de um Stop.



Condição de Acknowledge (ACK)

Na verdade, isso é uma resposta dada pelo Mestre ou pelo Escravo, no lugar do 9º bit, informando se o dado foi corretamente recebido ou não. Um acknowledge alto (1) representa um erro, enquanto um acknowledge baixo (0) representa OK. No final de uma transmissão de dados do Escravo para o Mestre, isto é, após o último byte, o Mestre deve sempre responder ACK=1.

Transmissão de endereço

Independentemente da informação a ser transmitida (endereço de 7 bits, endereço de 10 bits ou dados de 8 bits), a comunicação sempre é feita em pacotes de 8 bits e mais um retorno de acknowledge. Desta forma, para cada informação transmitida são necessários nove pulsos no clock. O último bit do primeiro byte não faz parte da informação, pois é utilizado como um flag de marcação para escrita (W-write) ou leitura (R-Read). Quando o Mestre deseja enviar dados ao escravo, esse bit recebe o valor 0 (zero). Quando ele deseja receber um valor do Escravo, ele começa perguntando com este bit em 1 (um). E por isso que o endereço só comporta 7 bits: são exatamente os sete que sobraram nesse mesmo byte.

Para o endereçamento de 10 bits, será necessária a transmissão inicial de 2 bytes só para o endereço. No primeiro byte, o bit menos significativo continua válido como R/W. Dos outros 7 bits, entretanto, somente dois serão considerados como a parte alta do endereço. Trata-se dos bits 2 e 1. Os bits de 7 a 3 devem ser escritos com o valor 11110. Após a transmissão do primeiro byte, SDA é transformado em entrada para checar o acknowledge ao ser dado o 9º pulso do clock. Se a resposta for afirmativa termos ACK=0.

Para a geração de um pulso no clock, o nível baixo é conseguido impondo-se o aterrramento do pino **SCL** (como se ele fosse uma saída). Em seguida, para o nível alto, desliga-se o aterrramento e o nível é conseguido pelo pull-up da linha. Durante todo o tempo do pulso, o pino **SCL** deve ser monitorado pelo Mestre e, caso ele desça, é porque algum Escravo está pedindo uma pausa no processo. O tempo do clock é, então, paralisado e o Mestre só volta a responder quando a linha de **SCL** estiver liberada.

Quando o clock estiver liberado, o Mestre irá iniciar a transmissão do segundo byte de endereço. Neste segundo byte estão os 8 bits menos significativos que faltam para completar o endereço de 10 bits. Ao término, um novo ACK será aguardado e conferido.

Transmissão de dados

Depois do término da transferência da parte relacionada ao endereço, um dos dois lados (Mestre ou Escravo) deverá transmitir um ou mais bytes de dados. Caso seja o Escravo que esteja transmitindo para o Mestre, para finalizar a comunicação o Mestre deve responder um ACK=1 ao término do último byte recebido, para só depois enviar uma condição de Stop. Isso se faz necessário para que o Escravo seja informado de que o Mestre não mais deseja receber nenhum dado, evitando conflitos quando a condição de Stop for gerada. Nesta condição o Escravo deve ter sua comunicação paralisada e resetada, ficando no aguardo de uma nova condição de Start. Com isso o protocolo fica mais robusto, evitando erros na contagem do número de bytes enviados.

Pausas

Sempre que algum módulo Slave (ou até outro Master) necessitar de uma pausa na comunicação, por exemplo, para processar a informação recebida, ela será conseguida mantendo-se a linha de clock em nível baixo, isto é, em zero.

Diagramas de tempo

Os diagramas apresentados nas páginas seguintes demonstram graficamente o protocolo I^C para os quatro casos possíveis já comentados:

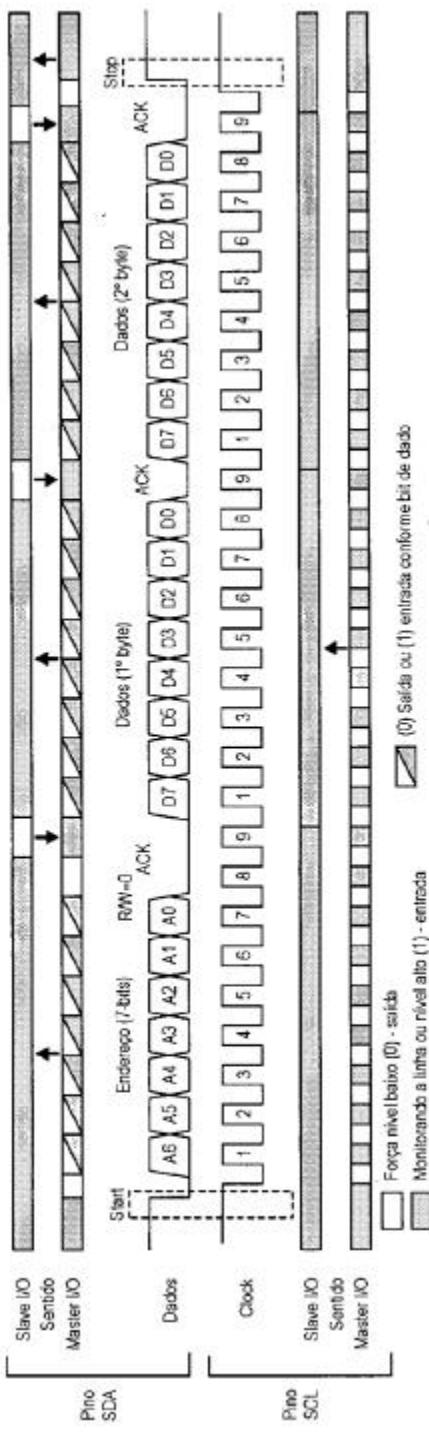
- Transmissão do Master/Recepção do Slave, com endereço de 7 bits;
- Recepção do Master/Transmissão do Slave, com endereço de 7 bits;
- Transmissão do Master/Recepção do Slave, com endereço de 10 bits;
- Recepção do Master/Transmissão do Slave, com endereço de 10 bits.

Em cada um deles são mostradas as informações que trafegam pelas vias de Dados e de Clock. Além disso, acima e abaixo dos desenhos das formas de onda existem também barras que representam qual dos dois lados está controlando a via. São duas barras para cada pino (**SDA** e **SCL**), uma representando o lado do Master e a outra o lado do Slave. Quando a barra está cinza significa que o pino está configurado como entrada e quando a barra está branca é porque o pino está impondo o nível 0 (baixo). Desta forma fica mais fácil visualizarmos o sentido de tráfego da informação:

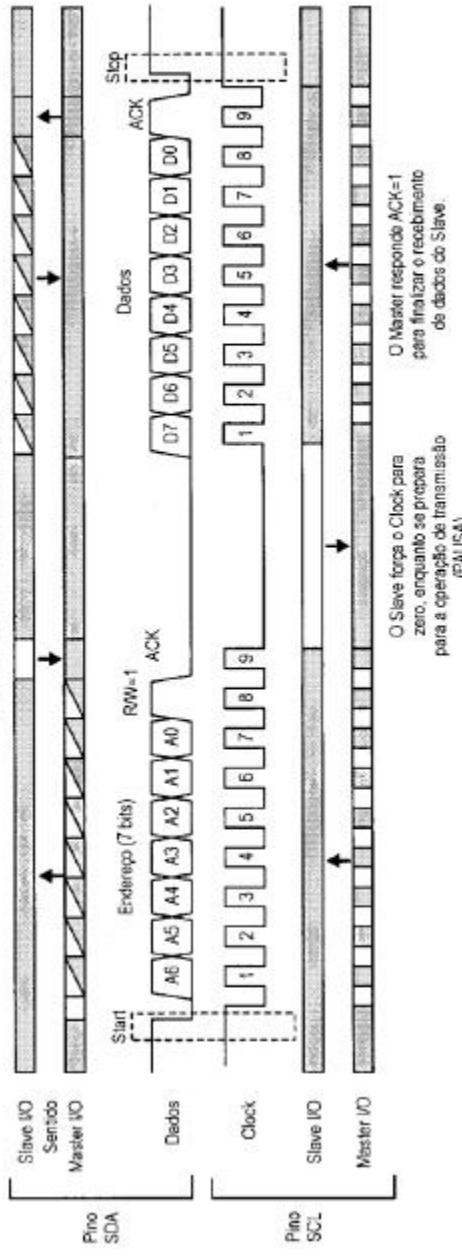
- Master para Slave;
- Slave para Master.

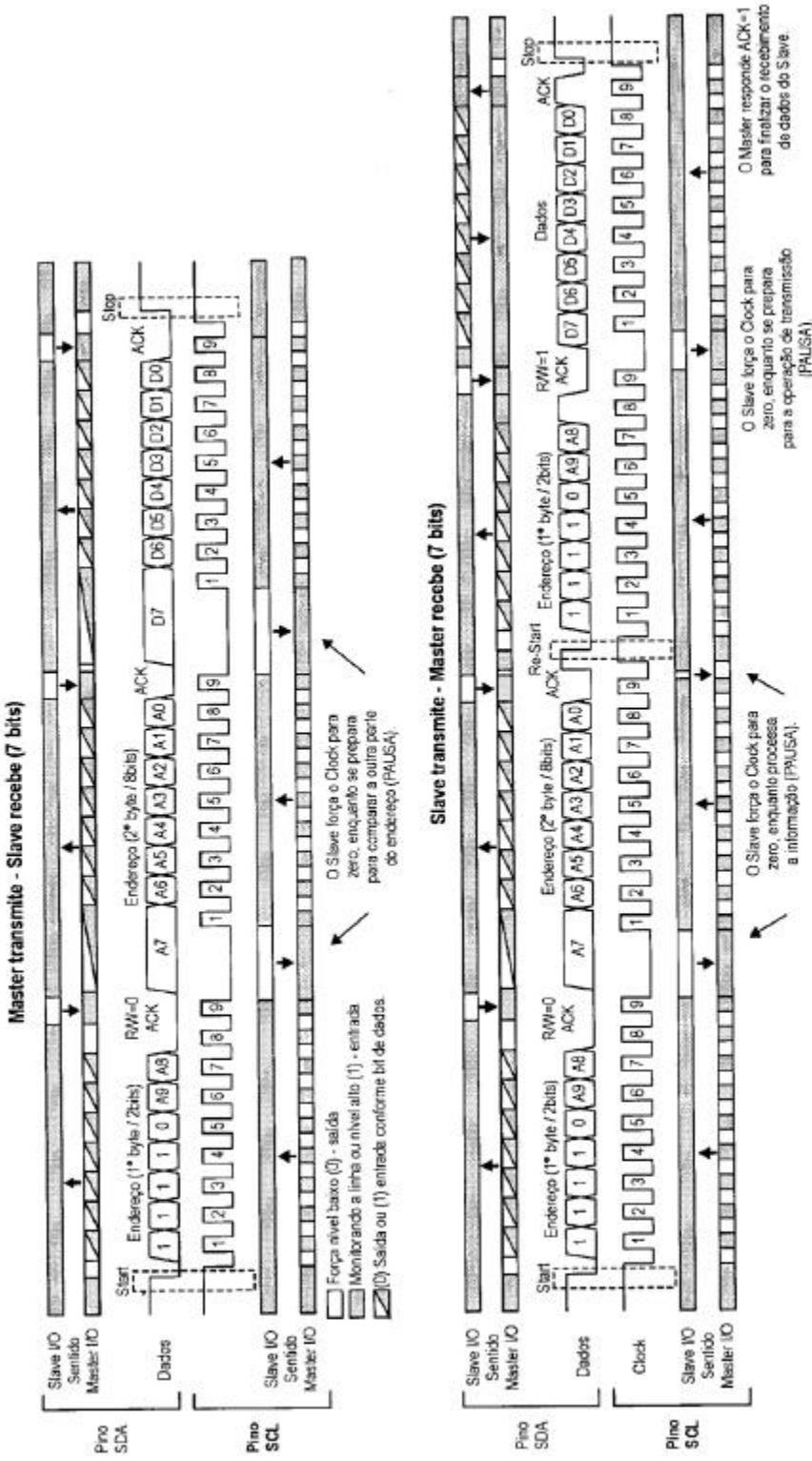
Observando essas barras para o pino de clock (**SCL**) podemos ver claramente quando o Slave força pausas na comunicação, impondo o nível baixo (zero) nesta via.

Master transmite - Slave recebe (7 bits)



Slave transmite - Master recebe (7 bits)





Comunicação Serial 1 - SPI e I²C

Recursos do PIC para I²C

Agora que as formas gerais do protocolo já estão explicadas, vamos ver como ativar e utilizar esse recurso no PIC 16F877A.

Antes de mais nada, vamos explicar melhor como o PIC controla os pinos **SCL** e **SDA**. Como já foi dito na teoria do I²C, para que o protocolo funcione corretamente, evitando conflitos na linha, cada um dos módulos ligados à rede deve somente impor nível baixo (0), sendo o nível alto imposto pelos resistores de pull-up. O hardware interno do PIC respeita essas condições. Para isso, entretanto, é necessário que ambos os pinos (**SCL** e **SDA**) sejam configurados inicialmente como entrada através de **TRISC**, para que os mesmos fiquem em alta impedância, possibilitando que o controle da I²C possa forçar o nível baixo quando necessário. A imposição do GND na linha é feita por intermédio de um transistor interno do tipo N. Este fato possibilita a checagem de colisão de dados na linha **SDA**, que será explicada posteriormente.

Caso um desses pinos, por exemplo o **SCL**, seja configurado como saída em nível alto, o sistema poderá continuar funcionando, pois quando a I²C não estiver impondo zero, a própria porta imporá 1. Isso não gerará problemas internos para o PIC, mas poderá gerar problemas de conflito na linha, caso seja utilizado mais de um Master ou caso algum Slave solicite uma pausa através da linha de clock.

Para facilitar o entendimento, dividiremos essas explicações em duas partes: uma dedicada ao Slave e outra ao Master.

Modo Slave

Inicie sempre configurando os pinos **SCL** e **SDA** como entrada por meio do **TRISC**.

Para configurarmos um PIC para trabalhar em modo Slave, devemos ajustar os bits 0= **SSPCON<SSPM3:SSPM0>**:

SSPM3: SSPM0	Descrição
0110	I ² C Slave, endereço de 7 bits.
0111	I ² C Slave, endereço de 10 bits.

Obs: As demais opções de configuração dizem respeitos aos outros modos (SPI e I²C Master)².

Depois é necessário também ajustar o bit **SSPSTAT<SMP>** que, neste caso, serve para habilitar um sistema interno de tratamento das bordas quando operando em freqüência de 400 kHz:

SMP	Descrição
0	Tratamento desabilitado.
1	Tratamento habilitado, quando usando 400kHz.

O bit **SSPSTAT<CKE>** serve para configurar o tipo de Smith Trigger utilizado nos pinos SCL e SDA.

CKE	Descrição
0	Entradas conforme especificações I ² C. ²
1	Entradas conforme especificações SMBUS.

Por último é necessário ainda habilitar o sistema de comunicação através do bit **SSPCON<SSPEN>**:

SSPEN	Descrição
0	Desabilita a comunicação.
1	Habilita a comunicação. Neste momento os pinos SCL e SDA deixam de operar como I/Os convencionais.

Feito isso, vejamos quais tarefas serão feitas automaticamente e quais deverão ser implementadas pelo programa. Trataremos separadamente os possíveis casos de transferência de informações.

Recepção para endereçamento de 7 bits

1. Tanto SCL quanto SDA estão em sua configuração padrão, isto é, como entradas. A linha receberá então uma condição de Start
2. A condição de Start é reconhecida e o bit **SSPSTAT<S>** é setado; porém nenhuma interrupção acontece.
3. O sistema aguardará, então, um byte com os 7-bits de endereço e mais o bit **R/W=0**, com oito pulsos de clock. Teremos **SSPSTAT<R/W>=0**. Como o byte recebido é equivalente a um endereço, então **SSPSTAT<D/A>=0**.
4. Neste momento, o endereço recebido é comparado automaticamente com o registrador **SSPADD**. Caso não seja o mesmo, a comunicação do lado Slave é resetada (ACK=1) e o mesmo ficará aguardando um novo Start. Caso o endereço seja idêntico, o bit **SSPSTAT<BF>** é setado, informando que um valor foi transferido para o buffer (**SSPBUF**), o pino **SDA** é transformado automaticamente em saída com nível baixo (ACK=0) e fica aguardando o 9º pulso de clock. Logo depois **SDA** é liberado, tornando-se novamente entrada.
Neste momento também o flag **PIR1<SSPIF>** é setado e a interrupção pode acontecer (se as chaves estiverem corretamente ligadas). É importante que a interrupção aconteça para que seja feita uma leitura do **SSPBUF**, forçando novamente **BF=0**.
5. O sistema aguardará então um byte de dado, com oito novos pulsos de clock. Como agora o byte recebido é equivalente a um dado, então **SSPSTAT<D/A>=1**. Neste momento, também teremos **BF=1** e o dado será colocado em **SSPBUF**.
6. Mais uma vez o pino **SDA** impõe um nível baixo para responder ACK=0 no 9º pulso do clock. **SDA** volta em seguida à condição de entrada.
7. Uma nova interrupção acontece (**SSPIF=1**) para que o dado recebido possa ser tratado.
8. Novos bytes de dados podem continuar sendo recebidos até que uma condição de Stop seja imposta na linha. Neste momento, a condição será reconhecida, setando o bit **SSPSTAT<P>** e finalizando a comunicação. A interrupção não é gerada.
9. Caso seja recebido uma condição de Re-Start (entre um Start e um Stop), o Slave ficará esperando um novo byte de endereço, com o **R/W**, e não mais um byte de dados.

Devido ao buffer duplo de recepção, isto é, o dado entra em **SSPSR** e depois é colocado em **SSPBUF**, a informação recebida pode então ser lida mesmo durante o recebimento do próximo byte. Depois que lemos **SSPBUF**, o flag **BF** é limpo automaticamente. Entretanto, caso o **SSPBUF** não seja lido antes da nova recepção terminar, o dado mais novo será perdido (**SSPBUF** não é atualizado) e o bit **SSPCON<SSPOV>** será setado, indicando um overflow. **SSPOV** deve ser limpo manualmente pelo

programa. Dependendo das condições de **BF** e **SSPOV**, a resposta automática do acknowledge pode ser negativa (ACK=1).

BF	SSPOV	Atualiza SSPBUF	ACK	Interrupção SSPIF=1
0	0	Sim	0(Ok)	Sim
1	0	Não	1 (Erro)	Sim
1	1	Não	1 (Erro)	Sim
0 (Nota)	1 (Nota)	Sim	1 (Erro)	Sim

Nota: Esta condição representa um erro de programação, pois o SSPBUF foi limpo e o bit de SSPOV não.

Transmissão para endereçamento de 7-bits

1. Tanto **SCL** quanto **SDA** estão em sua configuração padrão, isto é, como entradas. A linha receberá, então, uma condição de Start.
2. A condição de Start é reconhecida e o bit **SSPSTAT<S>** é setado; porém nenhuma interrupção acontece.
3. O sistema aguardará então um byte com os 7-bits de endereço e mais o bit **R/W=1**, com oito pulsos de clock. Teremos **SSPSTAT<R/W>=1**. Como o byte recebido é equivalente a um endereço, então **SSPSTAT<D/A>=0**.
4. O endereço recebido é comparado automaticamente com o registrador **SSPADD**. Caso não seja o mesmo, a comunicação do lado Slave é resetada (ACK=1) e o mesmo ficará aguardando um novo Start. Caso o endereço seja idêntico, o pino **SDA** é transformado automaticamente em saída com nível baixo (ACK=0) e o sistema aguarda o 9º pulso de clock. Logo depois **SDA** é transformado novamente em entrada.
5. O sistema automaticamente altera **SSPCON<CKP>=0**, desabilitando o clock (pausa). Isso é feito impondo-se nível baixo (0) em **SCL** (saída).
6. O flag **PIR1<SSPIF>** é, então, setado e a interrupção pode acontecer (se as chaves estiverem corretamente ligadas). Importante que a interrupção aconteça para que seja escrito o valor a ser transmitido em **SSPBUF**, o que irá fazer com que **BF=1**.
7. Depois do buffer atualizado, para que a comunicação proceda, basta impor **CKP=1**. Neste momento, **SCL** será novamente liberado (entrada).
8. O clock será liberado e o byte existente em **SSPBUF** será enviado nos próximos oito pulsos. Ao término teremos **BF=0**.
9. Mais uma vez o pino **SDA** fica aguardando um ACK no 9º pulso do clock. Caso seja recebido um ACK=1 é porque a comunicação será encerrada, sendo aguardada, então, uma condição de Stop. Neste momento a condição será reconhecida, setando-se o bit **SSPSTAT<P>** e finalizando-se a comunicação. A interrupção não é gerada. Caso ACK=0, o sistema volta ao item 5 para que um novo byte de dado seja transmitido.
10. Durante a transferência de dados não pode acontecer uma situação de Re-Start. Para reiniciar completamente o sistema é necessário terminar a transferência com um ACK=1 por parte do Master, seguido de um Stop. Só então uma nova condição de Start poderá ser gerada.

Caso tente-se escrever um valor em **SSPBUF** enquanto uma transmissão está em progresso (**BF=1**) o valor de **SSPBUF** não será atualizado e o bit **SSPCON<WCOL>** será selado, indicando uma colisão na escrita. Este bit deve ser limpo manualmente pelo programador.

Recepção para endereçamento de 10 bits

1. Tanto **SCL** quanto **SDA** estão em sua configuração padrão, isto é, como entradas. A linha receberá, então, uma condição de Start.
2. A condição de Start é reconhecida e o bit **SSPSTAT<S>** é selado; porém nenhuma interrupção acontece.
3. O sistema aguardará então um byte com os 2-bits de endereço (A9:A8) e mais o bit **R/W=0**, com oito pulsos de clock. Teremos, então, **SSPSTAT<R/W>=0**. Como o byte recebido é equivalente a um endereço, teremos também **SSPSTAT<D/A>=0**.
4. Neste momento o endereço recebido é comparado automaticamente com o registrador **SSPADD**. Caso não seja o mesmo, a comunicação do lado Slave é resetada (**ACK=1**) e o mesmo ficará aguardando um novo Start. Caso esta parte do endereço esteja correta, o bit **SSPSTAT<BF>** é selado, informando que um valor foi transferido para o buffer (**SSPBUF**) e teremos **SSPSTAT<UA>=1**, indicando que o endereço deve ser atualizado. O pino **SDA** é transformado automaticamente em saída com nível baixo (**ACK=0**) e fica aguardando o 9º pulso de clock. Logo depois, **SDA** é transformado novamente em entrada. Neste momento, o flag **PIR1<SSPIF>** é selado e a interrupção pode acontecer (se as chaves estiverem corretamente ligadas). É importante que a interrupção aconteça para que seja feita uma leitura do **SSPBUF**, forçando novamente **BF=0**. O registrador **SSPADD** deve, então, ser atualizado com a parte baixa do endereço (A7:A0), o que forçará automaticamente **UA=0**. Depois do ACK e até o momento em que o **SSPADD** é atualizado, o Slave trava a comunicação (pausa) impondo um nível baixo (0) em **SCL**. Este controle é totalmente automático.
5. Será aguardado, então, um byte com a segunda parte do endereço (A7:A0), para os próximos oito pulsos do clock. Como o byte recebido ainda é equivalente a um endereço, continuamos com **SSPSTAT<D/A>=0**.
6. Todo o processo descrito para o item 4 será então repetido. No final, o sistema entra em nova pausa até que **SSPADD** seja novamente atualizado com a parte alta do endereço (A9:A8).
7. O sistema aguardará então um byte de dado, com mais oito pulsos de clock. Como o byte recebido é equivalente a um dado, então **SSPSTAT<D/A>=1**. Neste momento também teremos **BF=1** e o dado será colocado em **SSPBUF**.
8. O pino **SDA** é transformado automaticamente em saída com nível baixo (**ACK=0**) e o sistema aguarda o 9º pulso do clock. Logo depois **SDA** é transformado novamente em entrada.
9. Uma nova interrupção acontece (**SSPIF=1**) para que o dado recebido possa ser tratado.
10. Novos bytes de dados podem continuar sendo recebidos até que uma condição de Stop seja imposta na linha. Neste momento, a condição será reconhecida, setando-se o bit **SSPSTAT<P>** e finalizando-se a comunicação. A interrupção não é gerada.
11. Caso seja recebido uma condição de Re-Start (entre um Start e um Stop), o Slave ficará esperando um novo byte de endereço, com o bit de **R/W** e não mais um byte de dados.

Os comentários feitos a respeito do buffer e dos flags **BF** e **SSPOV** para a comunicação de 7-bits também são válidos aqui.

Transmissão para endereçamento de 10 bits

1. Tanto **SCL** quanto **SDA** estão em sua configuração padrão, isto é, como entradas. A linha receberá, então, uma condição de Start.
2. A condição de Start é reconhecida e o bit **SSPSTAT<S>** é selado; porém nenhuma interrupção acontece.
3. O sistema aguardará então um byte com os 2 bits de endereço (A9:A8) e mais o bit **R/W=0**, com oito pulsos de clock. Teremos, então, **SSPSTAT<R/W>=0**. Como o byte recebido é equivalente a um endereço, teremos também **SSPSTAT<D/A>=0**.
4. Neste momento, o endereço recebido é comparado automaticamente com o registrador **SSPADD**. Caso não seja o mesmo, a comunicação do lado Slave é resetada (ACK=1) e ficará aguardando um novo Start. Caso esta parte do endereço esteja correta, o bit **SSPSTAT<BF>** é selado, informando que um valor foi transferido para o buffer (**SSPBUF**) e teremos **SSPSTAT<UA>=1**, indicando que o endereço deve ser atualizado. O pino **SDA** é transformado automaticamente em saída com nível baixo (ACK=0) enquanto aguarda o 9º pulso do clock. Logo depois **SDA** é transformado novamente em entrada. Neste momento, o flag **PIR1<SSPIF>** é selado e a interrupção pode acontecer (se as chaves estiverem corretamente ligadas). É importante que a interrupção aconteça para que seja feita uma leitura do **SSPBUF**, forçando novamente **BF=0**. O registrador **SSPADD** deve, então, ser atualizado com a parte baixa de endereço (A7:A0), o que forçará automaticamente **UA=0**. Depois do ACK e até o momento em que o **SSPADD** é atualizado, o Slave trava a comunicação (pausa) impondo um nível baixo (0) em **SCL**. Esse controle é totalmente automático.
5. Será aguardado então um byte com a segunda parte do endereço (A7:A0) nos próximos oito pulsos do clock. Como o byte recebido ainda é equivalente a um endereço, continuamos com **SSPSTAT<D/A>=0**.
6. Todo o processo descrito para o item 4 será repetido. No final o sistema entra em pausa até que **SSPADD** seja novamente atualizado com a parte alta do endereço (A9:A8).
7. A linha fornecerá, então, uma condição de Re-Start, informando que o próximo byte é novamente uma informação de controle.
8. Será aguardado, então, outro byte com a parte alta do endereço e com **R/W=1**, para os próximos oito pulsos do clock. Como o byte recebido ainda é equivalente a um endereço, continuamos com **SSPSTAT<D/A>=0**. O valor recebido será colocado em **SSPBUF** e **BF=1**.
9. O pino **SDA** é transformado automaticamente em saída com nível baixo (ACK=0), enquanto o sistema aguarda o 9º pulso do clock. Logo depois, **SDA** é transformado novamente em entrada.
10. O sistema automaticamente altera **SSPCON<CKP>=0**, desabilitando o clock (pausa). Isso é feito transformando **SCL** em saída com nível baixo (0).
11. O flag **PIR1<SSPIF>** é, então, selado e a interrupção pode acontecer (se as chaves estiverem corretamente ligadas). É importante que a interrupção aconteça para que **SSPBUF** seja lido, forçando **BF=0** novamente. Depois devemos escrever o valor a ser transmitido em **SSPBUF**, o que irá fazer com que **BF=1**.

12. Depois do buffer atualizado, para que a comunicação proceda, basta impor CKP=1. Neste momento, **SCL** será novamente liberado.
13. O clock será liberado e o byte existente em **SSPBUF** será enviado nos próximos oito pulsos. Ao término teremos **BF=0**.
14. Mais uma vez o pino SDA fica aguardando um ACK no 9º pulso do clock. Caso seja recebido um ACK=1 é porque a comunicação será encerrada, sendo aguardada então uma condição de Stop. Neste momento, a condição será reconhecida, setando-se o bit **SSPSTAT<P>** e finalizando-se a comunicação. A interrupção não é gerada. Caso ACK=0, o sistema volta ao item 5 para que um novo byte de dado seja transmitido.
15. Durante a transferência de dados não pode acontecer uma nova situação de Re-Start, Para reiniciar completamente o sistema é necessário terminar a transferência com um ACK=1 por parte do Master, seguido de um Stop. Só então uma nova condição de Start poderá ser gerada.

A condição de colisão (**WCOL**) descrita na comunicação de 7-bits também é válida para a de 10-bits.

Tratamento da interrupção SSP

Como só existe uma interrupção de porta serial (SSP) e diversas condições, os bits adicionais desta porta devem ser checados para sabermos exatamente que atitude deve ser tomada. Veja algumas condições:

Situação				
BF	UA	D/A	SSPOV	
1	X	X	X	Sempre que a interrupção acontecer BF=1 , é porque alguma informação foi colocada em SSPBUF . Para o caso da transmissão, este flag pode ser checado para sabermos se o processo já terminou, evitando uma colisão (WCOL).
1	X	0	X	Foi recebido um endereço. Como o endereço é checado automaticamente, a interrupção deve somente ler SSPBUF para limpar BF .
1	1	0	X	Somente quando configurada comunicação em 10 bits. A interrupção deverá atualizar o endereço, primeiro com o valor baixo e da segunda vez com o valor alto.
1	0	0	X	Esta situação só acontece para 7 bits ou quando recebido o segundo byte de controle para a transmissão quando operando em 10 bits, onde também teremos R/W=1.
1	X	X	1	Significa que a última informação recebida foi perdida, pois o SSPBUF não havia sido limpo.

Endereçamento global

O Slave possui ainda um último recurso, que podemos chamar de endereço global. Isso significa que o endereço 0 (zero) poderá ser aceito automaticamente por todos os PICs. Desta forma, cada unidade da rede possui um endereço específico (**SSPADD**) e um endereço global (Zero).

Para habilitar o uso de endereço global em um módulo Slave, basta ativar o bit **SSPCON2<GCEN>**:

GCEN	Descrição
0	Endereço global (Zero)
1	Endereço global (Zero) habilitado.

Quando o endereço global é inserido na rede, todos os módulos S/ave irão receber a informação e todos aqueles com **GCEN** deverão responder com um ACK. Isso nos lembra um comentário muito importante. Quando ACK=0, o Slave coloca **SDA** como saída em nível baixo. Porém, quando ACK=1, **SDA** continua como entrada e o nível alto é imposto pelo resistor de pull-up. Isso possibilita que diversos módulos respondam ACKs diferentes, tendo sempre a preferência o erro, ou seja, ACK=0.

Com a habilitação do endereço global a interrupção será gerada e o tratamento ou não das informações será uma escolha do programador.

Resumo dos registradores associados a I²C Slave²

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
OCh	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
94h	SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSMPM3	SSPM2	SSPM1	SSPM0
91h	SPCON2	GCEN	ACKST AT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEM
13h	SSPBUF	Buffer de transmissão/recepção							
93H	SSPADD	Endereço (Parte alta ou parte baixa)							

Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Modo Master

O Master não diferencia automaticamente o tipo de comunicação em 7 ou 10-bits, ele somente transmite e recebe informações de 8-bits. Por isso, toda a lógica da comunicação deve ser feita por intermédio do programa. Internamente, o PIC possui controle de hardware para os seguintes eventos:

- *Gerar/Monitorar condição de Start; *Gerar/Monitorar Acknowledge;
- *Gerar/Monitorar condição de Stop; *Gerar/Monitorar o clock,
- *Gerar/Monitorar condição de Re-Start, *Transmitir informação de 8-bits;
- *Gerar/Monitorar condição de Pausa; *Receber informação de 8-bits.

Inicie sempre configurando os pinos **SCL** e **SDA** como entrada através do **TRISC**.

Para configurarmos um PIC para trabalhar em modo Master, devemos ajustar os bits * **SSPCON<SSPM3:SSPM0>**:

SSPM3 SSPM0	Descrição
1000	I ² C Master, controle por hardware com Clock=Fosc / (4x (SSPADD+1))
1001	Reservado
1010	Reservado
1011	Reservado
1100	Reservado
1101	Reservado

SSPM3: SSPM0	Descrição
1110	Reservado
1111	Reservado
Obs: As demais opções de configuração dizem respeitos aos outros modos (SPI e I ² C Slave).	

Depois, é necessário também ajustar o bit **SSPSTAT<SMP>** que, neste caso, serve para habilitar um sistema interno de tratamento das bordas, quando operando em freqüência de 400 kHz:

SMP	Descrição
0	Tratamento desabilitado.
1	Tratamento habilitado, quando usando 400 kHz.

O bit **SSPSTAT<CKE>** serve para configurar o tipo de Smith Trigger utilizado nos pinos **SCL** e **SDA**:

CKE	Descrição
0	Entradas conforme especificações I ² C.
1	Entradas conforme especificações SMBUS.

A freqüência do clock deve ser configurada através do registrador SSPADD, respeitando-se a seguinte fórmula:

$$\text{SSPADD} = \left\lceil \frac{\frac{F_{\text{osc}}}{\text{Freq.}}}{4} \right\rceil - 1$$

As freqüências padronizadas para o protocolo I²C são as de 100 kHz e 400 kHz. Com o PIC, entretanto, é possível configurar outras freqüências. O fato é que todos os periféricos da rede devem estar aptos a responder na freqüência escolhida. Por isso, o recomendável é verificar a freqüência mais alta suportada pelos periféricos e escolher o valor mais próximo possível dos padronizados.

Por último, é necessário ainda habilitar o sistema de comunicação através do bit **SSPCON<SSPEN>**:

SSPSI	Descrição
0	Desabilita a comunicação.
1	Habilita a comunicação. Neste momento os pinos SCL e SDA deixam de operar como I/Os convencionais

Feito isso, vejamos que tarefas serão feitas automaticamente e quais deverão ser implementadas pelo programa. Trataremos separadamente os possíveis casos de transferência de informações.

Transmissão para endereçamento de 7 bits

1. Tanto SCL quanto SDA estão em sua configuração padrão, isto é, como entradas.

2. Deve ser imposta uma condição de Start na linha. Para isso, basta ativar o bit **SSPCON2<SEN>**. A condição de Sstart será, então, gerada automaticamente.
3. Ao término do Start, o flag da interrupção será ativado (**SSPIF=1**) e **SEN=0**. Deve-se, então, escrever em **SSPBUF** o endereço do periférico com o qual desejamos nos comunicar. O endereço deve ser escrito nos bits de 7 a 1. O bit 0 deve ser considerado como o valor **R/W=0**.
4. Automaticamente, logo após a escrita no registrador SSPBUF, teremos **SSPSTAT<BF>=1**, **SSPSTAT<R/W>=1** e o byte será transmitido através de oito pulsos gerados em **SCL**. No término da transmissão teremos novamente **BF=0**.
5. O pino **SDA** aguarda um ACK (deve ser 0) no Q-pulso do clock. O valor recebido é armazenado no bit **SSPCON2<ACKSTAT>**. O valor de ACK não é checado automaticamente.
6. Neste momento, **R/W=0** e **SSPIF=1**, podendo gerar uma nova interrupção.
7. O dado a ser transmitido é, então, colocado em **SSPBUF**. Teremos **BF=1** e **R/W=1**.
8. Mais uma vez teremos a transmissão do byte através de oito pulsos em **SCL**, a partir de momento em que nenhum periférico esteja solicitando uma pausa. No término **BF=0**.
9. Novamente o pino **SDA** aguarda um ACK (deve ser 0) no 9s pulso do clock. O valor recebido é armazenado no bit **SSPCON2<ACKSTAT>**. O valor de ACK não é checado automaticamente.
10. Neste momento **R/W=0** e **SSPIF=1**, podendo gerar outra interrupção.
11. Se um novo byte deve ser transmitido, o processo é reiniciado no item 7. Para interromper a transmissão, uma condição de Stop deve ser gerada. Isso é conseguido por intermédio do bit **SSPCON2<PEN>=1**.
12. No final da condição de Stop teremos automaticamente **PEN=0** e **SSPIF=1**.

Recepção para endereçamento de 7 bits

1. Tanto **SCL** quanto **SDA** estão em sua configuração padrão, isto é, como entradas.
2. Deve ser imposta uma condição de Start na unha. Para isso, basta ativar o bit **SSPCON2<SEN>**. A condição de Start será, então, gerada automaticamente.
3. Ao término do Start, o flag da interrupção será ativado (**SSPIF=1**) e **SEN=0**. Deve-se, então, escrever em **SSPBUF** o endereço do periférico com o qual desejamos nos comunicar. O endereço deve ser escrito nos bits de 7 a 1. O bit 0 deve ser considerado como o valor **R/W=1**.
4. Automaticamente, logo após a escrita no registrador **SSPBUF**, teremos **SSPSTAT<BF>=1**, **SSPSTAT<R/W>=1** e o byte será transmitido através de oito pulsos gerados em **SCL**. No término da transmissão teremos novamente **BF=0**.
5. O pino **SDA** aguarda um ACK (deve ser 0) no 99 pulso do clock. O valor recebido é armazenado no bit **SSPCON2<ACKSTAT>**. O valor de ACK não é checado automaticamente.
6. Neste momento, **R/W=0** e **SSPIF=1**, podendo gerar uma nova interrupção.

7. O Master deve, então, ser colocado em modo de recepção, ajustando-se **SSPCON2<RCEN>=1**.
8. Assim que a linha de clock não estiver em pausa, será aguardado um byte de dados, para os próximos oito pulsos do clock. Como o byte recebido é equivalente a um dado, teremos **SSPSTAT<D/A>=1**. O valor recebido será colocado em **SSPBUF** e **BF=1**.
9. Neste momento, **RCEN=0** (automaticamente) desligando-se o modo de recepção. Também teremos **SSPIF=1**, gerando uma nova interrupção.
10. Se um novo byte deve ser recebido, então o Master deverá responder ACK=0. Caso seja o último byte desejado, então ACK=1. Para responder o ACK, o valor desejado deve ser colocado no bit **SSPCON2<ACKDT>**. Liga-se a geração do sinal de ACK através de **SSPCON2<ACKEN>=1**.
11. Ao término do ACK teremos **ACKEN=0** e **SSPIF=1**, gerando-se outra interrupção. Caso tenha sido respondido ACK=0, então o processo deve retornar ao item 5 para que um novo byte seja recebido. Se ACK=1, o processo deve ser finalizado através de uma condição de Stop que será gerada com **SSPCON2<PEN>**.
12. No final da condição de Stop teremos automaticamente **PEN=0** e **SSPIF=1**.

Existem duas situações que podem gerar uma transmissão:

- **O Master efetua uma pergunta e deseja uma resposta:** Neste caso, primeiramente será feita uma recepção pelo Slave (pergunta) e depois uma transmissão (resposta). Entre a pergunta e a resposta pode ser gerada uma condição de Re-Start, não sendo necessário liberar a linha (Stop). Um exemplo disso são as memórias E PROM externas. Primeiro devemos informar qual endereço desejamos acessar para que depois ela responda o valor contido no endereço em questão.
- **O Master só deseja uma resposta:** Em casos de Slaves mais "burros" é provável que somente com seu endereço já seja possível responder corretamente. Por exemplo, para um módulo de teclado, basta que ele transmita o número da tecla pressionada, não precisando processar perguntas diferentes.

Transmissão para endereçamento de 10 bits

1. Tanto **SCL** quanto **SDA** estão em sua configuração padrão, isto é, como entradas.
2. Deve ser imposta uma condição de Start na linha. Para isso, basta ativar o bit **SSPCON2<SEN>**. A condição de Start será, então, gerada automaticamente.
3. Ao término do Start, o flag da interrupção será ativado (**SSPIF=1**) e **SEN=0**. Deve-se, então, escrever em **SSPBUF** a parte alta do endereço do periférico com o qual desejamos nos comunicar. Os 2 bits do endereço (A9:A8) devem ser escritos nos bits de 2 e 1. O bit 0 deve ser considerado como o valor **R/W=0**. Os demais bits (7:3) devem conter os valores 11110.
4. Automaticamente, logo após a escrita no registrador SSPBUF, teremos **SSPSTAT<BF>=1**, **SSPSTAT<R/W>=1** e o byte será transmitido através de oito pulsos gerados em SCL. No término da transmissão teremos novamente **BF=0**.

5. O pino **SDA** aguarda um ACK (deve ser 0) no 9º pulso do clock. O valor recebido é armazenado no bit **SSPCON2<ACKSTAT>**. O valor de ACK não é checado automaticamente.
6. Neste momento, **R/W=0** e **SSPIF=1**, podendo gerar uma nova interrupção.
7. A segunda parte do endereço (A7:A0) é, então, escrita em **SSPBUF**.
8. Automaticamente, logo após a escrita no registrador SSPBUF, teremos **SSPSTAT<BF>=1**. **SSPSTAT<R/W>=1** e o byte será transmitido através de oito pulsos gerados em **SCL**. No término da transmissão teremos novamente **BF=0**.
9. O pino **SDA** aguarda um ACK (deve ser 0) no 9º pulso do clock. O valor recebido é armazenado no bit **SSPCON2<ACKSTAT>**. O valor de ACK não é checado automaticamente.
10. Neste momento, **R/W=0** e **SSPIF=1**, podendo gerar uma nova interrupção.
11. O dado a ser transmitido é, então, colocado em **SSPBUF**. Teremos **BF=1** e **R/W=1**.
12. Mais uma vez teremos a transmissão do byte através de oito pulsos em **SCL**, a partir de momento em que nenhum periférico esteja solicitando uma pausa. No término **BF=0**;
13. Novamente o pino **SDA** aguarda um ACK (deve ser 0) no 9º pulso do clock. O valor recebido é armazenado no bit **SSPCON2<ACKSTAT>**. O valor de ACK não é checado automaticamente.
14. Neste momento, **R/W=0** e **SSPIF=1**, podendo gerar outra interrupção.
15. Se um novo byte deve ser transmitido, o processo é reiniciado no item 11. Para interromper a transmissão, uma condição de Stop deve ser gerada. Isso é conseguido por intermédio do br. **SSPCON2<PEN>=1**.
16. No final da condição de Stop teremos automaticamente **PEN=0** e **SSPIF=1**.

Recepção para endereçamento de 10 bits

1. Tanto **SCL** quanto **SDA** estão em sua configuração padrão, isto é, como entradas.
2. Deve ser imposta uma condição de Start na linha. Para isso, basta ativar o bit **SSPCON2<SEN>**. A condição de Start será, então, gerada automaticamente.
3. Ao término do Start, o flag da interrupção será ativado (**SSPIF=1**) e **SEN=0**. Deve-se, então escrever em **SSPBUF** a parte alta do endereço do periférico com o qual desejamos nos comunicar. Os 2 bits do endereço (A9:A8) devem ser escritos nos bits de 2 e 1. O bit 0 deve ser considerado como o valor **R/W=0**. Os demais bits (7:3) devem conter os valores 11110.
4. Automaticamente, logo após a escrita no registrador SSPBUF, teremos **SSPSTAT<BF>=1**, **SSPSTAT<R/W>=1** e o byte será transmitido através de oito pulsos gerados em **SCL**. No término da transmissão teremos novamente **BF=0**.
5. O pino **SDA** aguarda um ACK (deve ser 0) no 9º pulso do clock. O valor recebido é armazenado no bit **SSPCON2<ACKSTAT>**. O valor de ACK não é checado automaticamente.
6. Neste momento, **R/W=0** e **SSPIF=1**, podendo gerar uma nova interrupção.
7. A segunda parte do endereço (A7:A0) é, então, escrita em **SSPBUF**.

8. Automaticamente, logo após a escrita no registrador SSPBUF, teremos **SSPSTAT<BF>=1**, **SSPSTAT<R/W>=1** e o byte será transmitido através de oito pulsos gerados em SCL. No término da transmissão teremos novamente **BF=0**.
9. O pino **SDA** aguarda um ACK (deve ser 0) no 9º pulso do clock. O valor recebido é armazenado no bit **SSPCON2<ACKSTAT>**. O valor de ACK não é checado automaticamente.
10. Neste momento, **R/W=0** e **SSPIF=1**, podendo gerar uma nova interrupção.
11. Deve ser gerada então uma condição de fle-Start. Para isso, basta ativar o bit **SSPCON2<RSEN>=1**.
12. Ao término do Re-Start, o flag da interrupção será ativado (**SSPIF=1**) e **RSEN=0**.
13. Mais uma vez devemos colocar em **SSPBUF** o valor 11110, seguido dos bits A9:A8 e o último bit como **R/W=1**. Teremos **BF=1** e **R/W=1**.
14. Teremos novamente a transmissão do byte através de oito pulsos em SCL, a partir do momento em que nenhum periférico esteja solicitando uma pausa. No término **BF=0**.
15. Novamente o pino **SDA** aguarda um ACK (deve ser 0) no 9º pulso do clock. O valor recebido é armazenado no bit **SSPCON2<ACKSTAT>**. O valor de ACK não é checado automaticamente.
16. Neste momento, **R/W=0** e **SSPIF=1**, podendo gerar outra interrupção.
17. O Master deve, então, ser colocado em modo de recepção, ajustando-se **SSPCON2<RCEN>=1**.
18. Assim que a linha de clock não estiver em pausa, será aguardado um byte de dados para os próximos oito pulsos do clock. Como o byte recebido é equivalente a um dado, teremos **SSPSTAT<D/A>=1**. O valor recebido será colocado em **SSPBUF** e **BF=1**.
19. Neste momento, **RCEN=0** (automaticamente) desligando-se o modo de recepção. Também teremos **SSPIF=1**, gerando uma nova interrupção.
20. Se um novo byte deve ser recebido, então o Master deverá responder ACK=0. Caso seja o último byte desejado, então ACK=1. Para responder o ACK, o valor desejado deve ser colocado no bit **SSPCON2<ACKDT>**. Liga-se a geração do sinal de ACK através de **SSPCON2<ACKEN>=1**.
21. Ao término do ACK teremos **ACKEN=0** e **SSPIF=1**, gerando-se outra interrupção. Caso tenha sido respondido ACK=0, então o processo deve retornar ao item 13 para que um novo byte seja recebido. Se ACK=1, o processo deve ser finalizado por intermédio de uma condição de Stop que será gerada com **SSPCON2<PEN>**.
22. No final da condição de Stop teremos automaticamente **PEN=0** e **SSPIF=1**.

Outras considerações

Existem outros bits e flags envolvidos com o Mas/erque não foram comentados até o momento.

Como um sistema pode possuir mais de um Master, as condições de Start e Conectando o PIC 16F877A - Recursos Avançados

Stop geradas na linha acarretarão na ativação de **SSPIF**, podendo gerar uma interrupção em qualquer Master. Os flags **SSPSTAT<S>** e **SSPSTAT<P>** informam, respectivamente, as condições de Start e Stop. Desta forma é possível que outros Masters percebam quando a linha está ou não ocupada.

Aqui valem também os comentários para os flags **SSPCON<WCOL>** e **SSPCON<SSPOV>** que indicam, respectivamente, uma escrita inválida (na hora errada) em **SSPBUF** e o recebimento de um novo byte sem a leitura do byte anterior.

As condições de Start, Stop, Re-Start e ACK não podem ser geradas fora dos momentos corretos, quando o modo de comunicação não está em stand-by. A situação de stand-by acontece sempre que o hardware da I2C acaba o controle de um evento qualquer.

Detecção de colisão na linha (Bus collision)

Outro recurso que o Master possui está relacionado à colisão de informações na linha de dados (**SDA**). Vejamos como isso funciona.

Para que o PIC possa impor o nível baixo da linha (normalmente nós consideramos uma condição de saída) existe um transistor interno conectando esse pino ao GND. Quando essa conexão está feita, a linha realmente está em nível baixo e não há o que monitorar. No outro caso, entretanto, quando esse transistor está em aberto, o pino é considerado uma entrada e o nível alto é imposto pelo resistor de pull-up (externo). Nesta situação, o PIC monitora a linha. Caso seja necessário transmitir um bit em ' (nível alto / transistor aberto) e a linha esteja em zero, é porque algum outro ponto da rede está em conflito de comunicação.

Quando isso acontecer, o bit **PIR2<BCLIF>** será selado, e se **PIE2<BCLIE>=1** a interrupção será gerada. Teremos ainda o reset do sistema de comunicação, voltando ao estado de stand-by.

As colisões podem acontecer durante a transmissão de dados e as condições de Start, Stop, Re-Start e ACK.

Resumo dos registradores associados a I2C Master

Endereço	Nome	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0Bh...	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
OCh	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
94h	SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
13h	SSPBUF	Buffer de transmissão/recepção							
93h	SSPADD	Ajuste da frequência							

—
Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Lógica do exemplo

Nosso exemplo para este capítulo usará somente uma parte da teoria apresentada. Faremos uso x protocolo I²C, operando como Master, para comunicação com uma memória E PROM externa. A memória em questão é a 24C04, a mesma indicada no esquema do apêndice F.

Implementamos, então, um sistema muito simples que mostra um valor no LCD, podendo o mesmo ser alterado através de dois botões (S1 e S2). Esse valor é apresentado em hexadecimal e pode varar de 00 a FF. Através do botão S3 salvamos o valor atual em uma posição fixa da memória. O valor pode, então, ser novamente alterado através dos botões.

Para recuperar o valor salvo anteriormente, basta pressionar o botão S4.

Essa memória trabalha com o protocolo I²C de 7 bits. No entanto, ela distorce um pouco e interpretação desse protocolo. Para que ela funcione, devemos respeitar o seguinte:

O endereço (posição dentro da memória) onde queremos escrever foi dividido em 2 bytes ENDERECO_HIGH (P10:P8) e ENDERECO_LOW (P7:P0), pois essa memória possui 0,5 Kbyte

disponível e, portanto, mais de 8 bits são necessários para endereçar toda a memória. Assim sendo, para escrever um byte nessa memória devemos primeiramente enviar a posição (endereço interno) e depois o dado.

Esse tipo de memória não aceita endereçamento na rede, isto é, aquele endereçamento suportado pelo protocolo I²C. Por isso, ela substitui o byte de endereçamento por outro chamado de controle. O byte de controle é composto por:

[1] [0] [1] [0] [P10] [P9] [P8] [R/W]

Onde P10:P8 é a parte alta da posição onde desejamos escrever dentro da memória e R/W é o flag que indica se estaremos efetuando uma operação de escrita (R/W=0) ou leitura (R/W=1). No caso dessa memória, só é usado o bit P8 (512 bytes - > 9 bits).

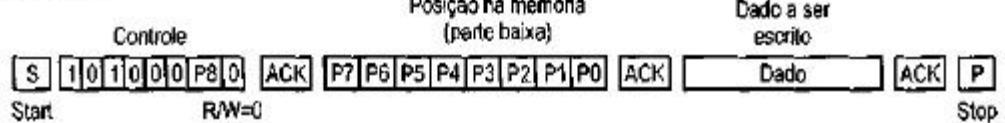
O segundo byte enviado (byte de dado no protocolo I²C) é equivalente ao complemento do endereço, isto é, sua parte baixa:

[P7] [P6] [P5] [P4] [P3] [P2] [P1] [P0]

Desta forma, teremos operações diferenciadas para a escrita e leitura da E2PROM externa.

Escrta

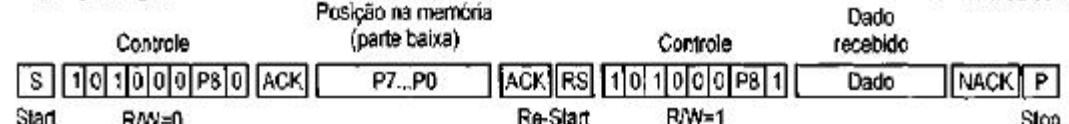
ACK = Acknowledge
recebido da E²PROM.
0 = Ok 1 = Erro



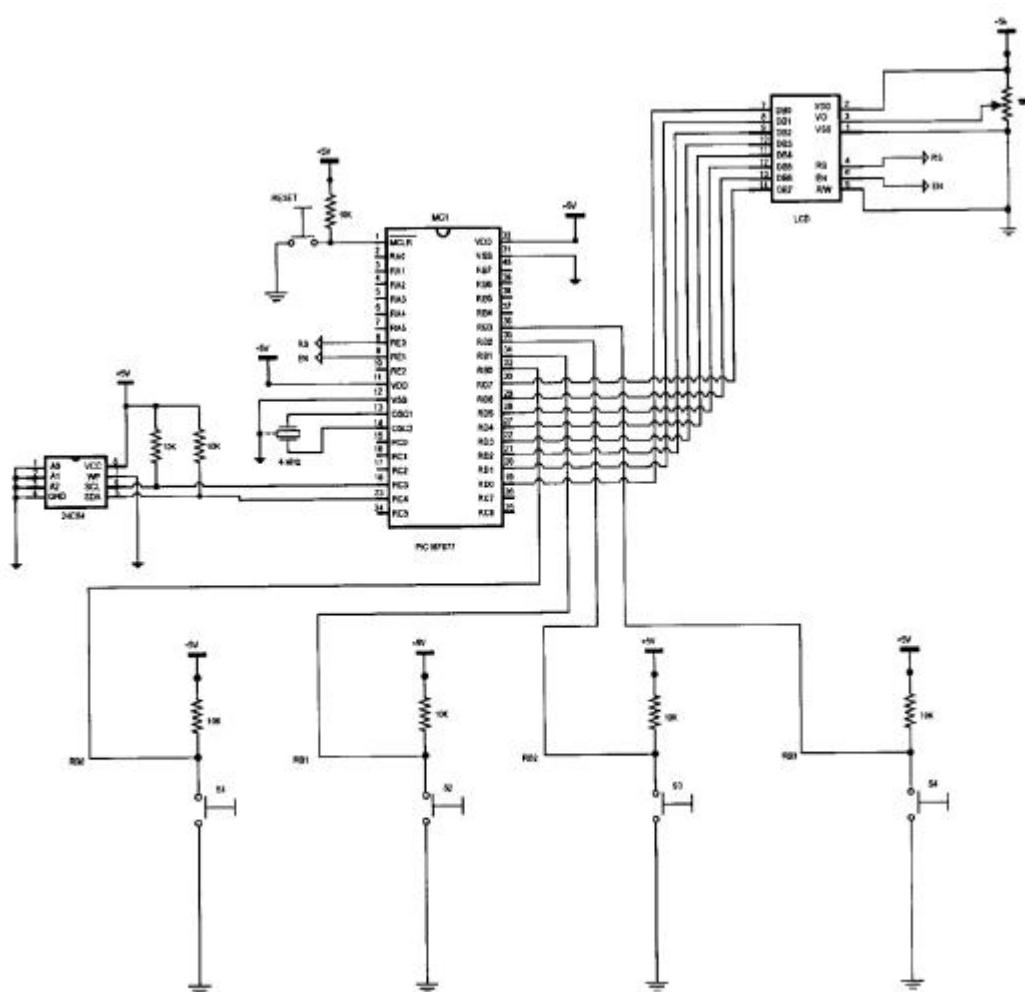
Leritura

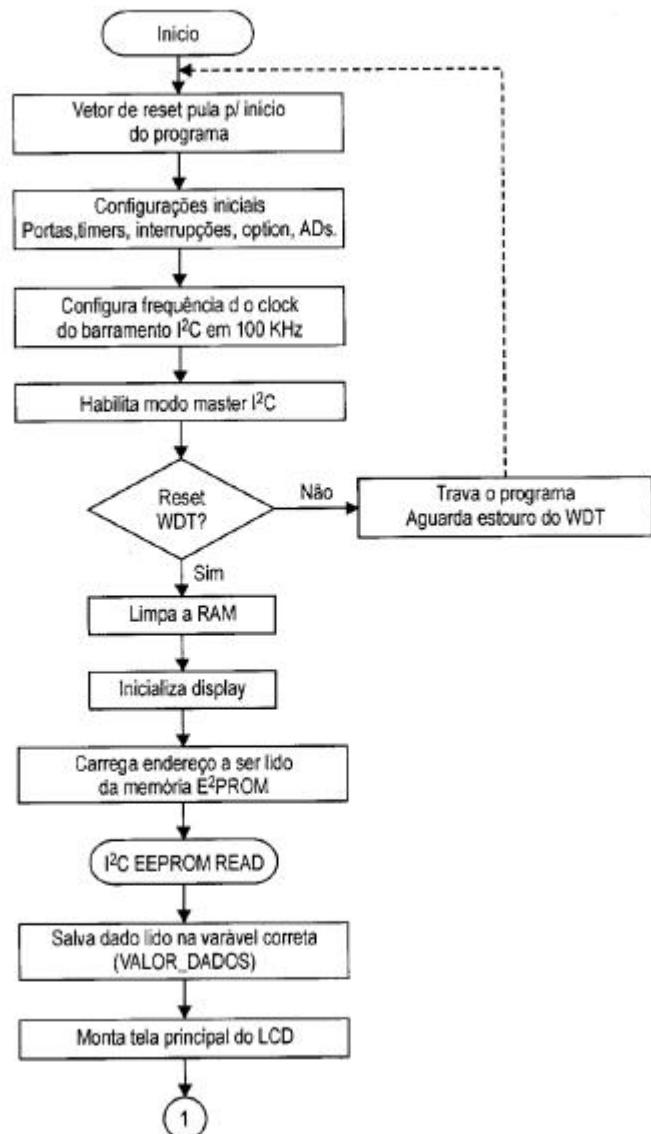
ACK = Acknowledge
recebido da E²PROM.
0 = Ok 1 = Erro

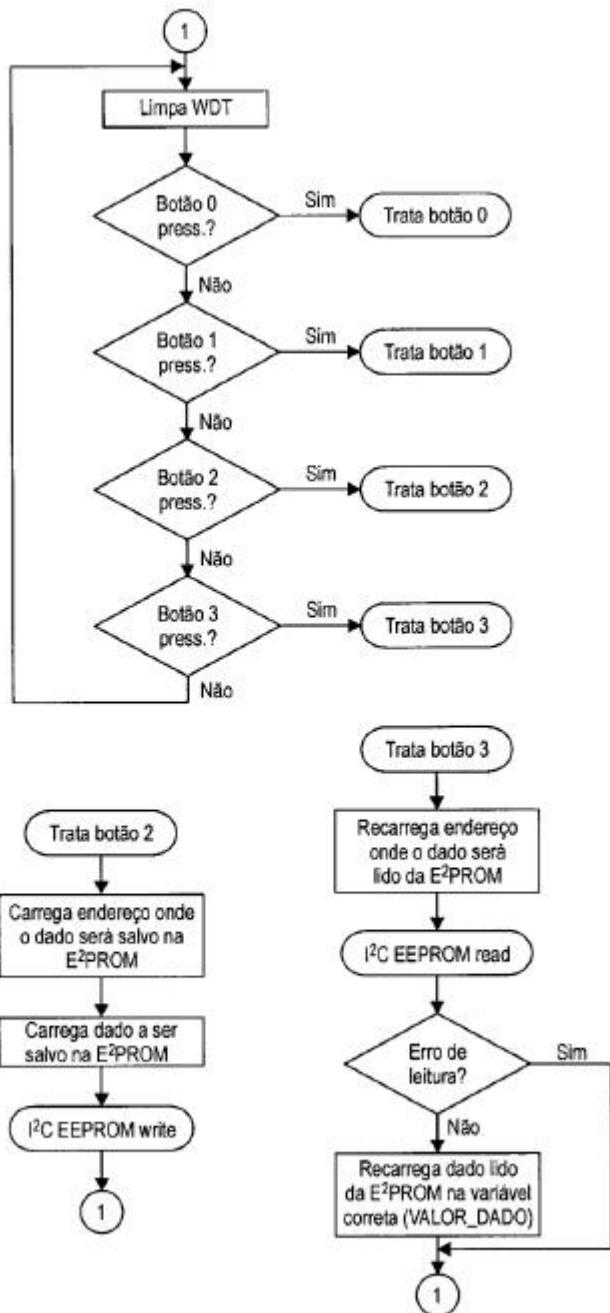
NACK = Acknowledge
respondido pelo PIC.
1 = Fim da com.

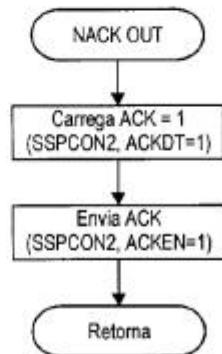
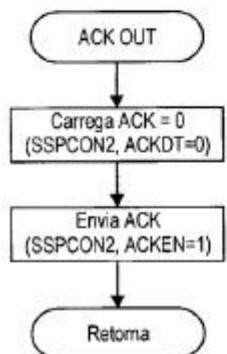
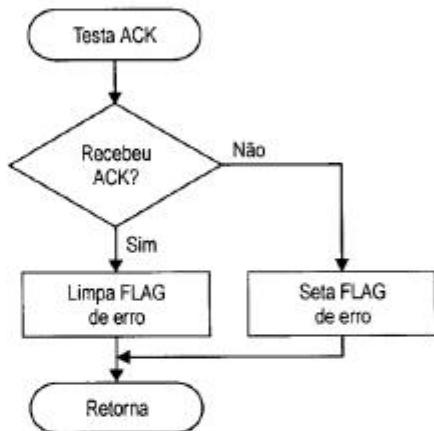


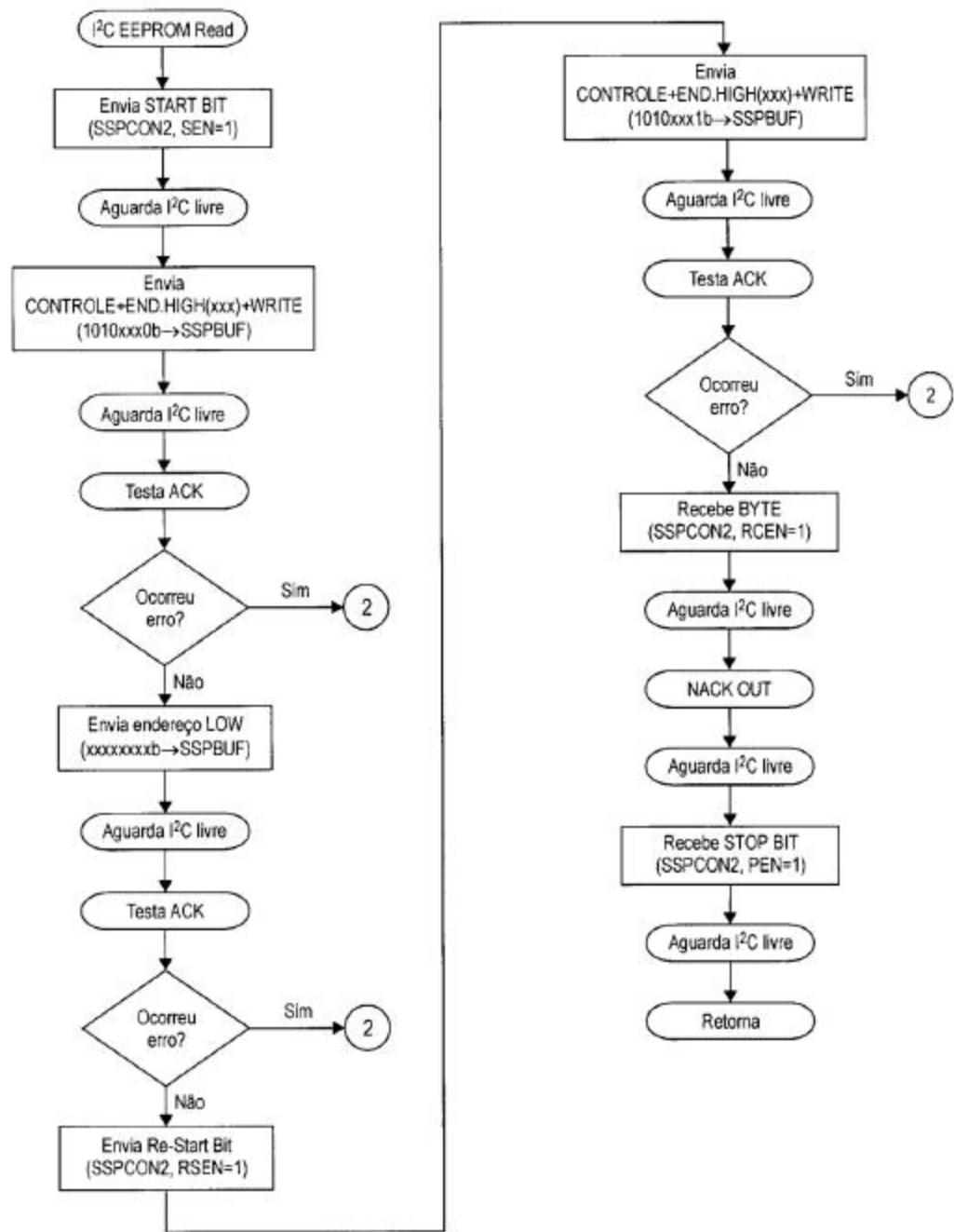
Esquema elétrico

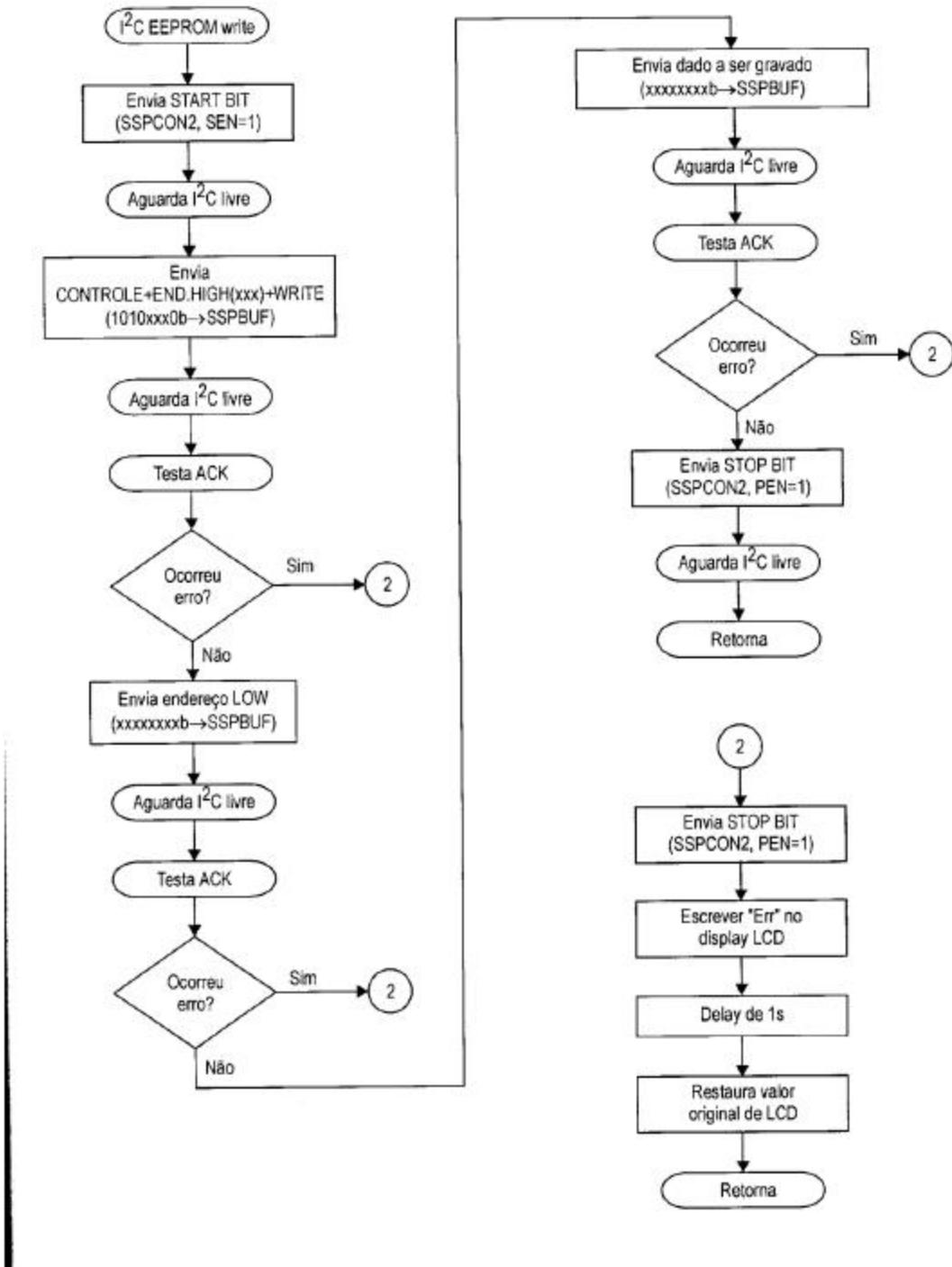












```

;*****
;*      CONECTANDO O PIC - RECURSOS AVANÇADOS      *
;*      EXEMPLO 8          *
;*      *
;*      NICOLÁS CÉSAR LAVINIA e DAVID JOSÉ DE SOUZA      *
;*      *
;*      *
;*      *****

;* VERSÃO : 2.0          *
;* DATA : 24/02/2003      *
;*****


;*****
;*      DESCRIÇÃO GERAL      *
;*****


; ESTE EXEMPLO FOI ELABORADO PARA EXPLICAR O FUNCIONAMENTO DA LEITURA/ESCRITA
; NA MEMÓRIA E2PROM SERIAL EXTERNA, UTILIZANDO O MASTER I2C.

;

;*****
;*      CONFIGURAÇÕES PARA GRAVAÇÃO      *
;*****


__CONFIG _CP_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &
_PWRTE_ON & _WDT_ON & _XT_OSC


;*****
;*      DEFINIÇÃO DAS VARIÁVEIS      *
;*****


; ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO LOGO NO INÍCIO DO BANCO 0


CBLOCK 0X20           ; POSIÇÃO INICIAL DA RAM

FILTRO_BOTOES        ; FILTRO PARA RUIDOS
TEMPO_TURBO          ; TEMPORIZADOR P/ TURBO DAS TECLAS
TEMPO1

```

TEMPO0 ; CONTADORES P/ DELAY

FLAG ; FLAG DE USO GERAL

AUX ; REGISTRADOR AUXILIAR DE USO GERAL

ENDERECO_HIGH ; REGISTRADORES DE ENDEREÇO PARA

ENDERECO_LOW ; ACESSO À MEMÓRIA EEPROM SERIAL EXTERNA

; MAPEADOS NO BANCO 0 DA RAM

BUFFER ; REGISTRADOR PARA LEITURA/GRAVAÇÃO NA
EEPROM SERIAL

; EXTERNA

VALOR_DADOS ; REGISTRADOR DE DADO PARA EEPROM SERIAL
EXTERNA

; MAPEADO NO BANCO 0 DA RAM

ENDC

;* DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC *

; O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE
; OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE
; DE REDIGITAÇÃO.

#INCLUDE <P16F877A.INC> ; MICROCONTROLADOR UTILIZADO

;* DEFINIÇÃO DOS BANCOS DE RAM *

; OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR
; ENTRE OS BANCOS DE MEMÓRIA.

#DEFINE BANK1 BSF STATUS,RP0 ; SELECCIONA BANK1 DA MEMORIA RAM

#DEFINE BANK0 BCF STATUS,RP0 ; SELECCIONA BANK0 DA MEMORIA RAM
Conectando o PIC 16F877A - Recursos Avançados

```

;*****
;*          CONSTANTES INTERNAS          *
;*****


; A DEFINIÇÃO DE CONSTANTES FACILITA A PROGRAMAÇÃO E A MANUTENÇÃO.

FILTRO_TECLA    EQU    .200           ; FILTRO P/ EVITAR RUIDOS DOS BOTÕES

TURBO_TECLA     EQU    .60            ; TEMPORIZADOR P/ TURBO DAS TECLAS

END_EEPROM_H    EQU    0X00          ; ENDEREÇO P/ LEITURA E GRAVAÇÃO
END_EEPROM_L    EQU    0X00          ; NA MEMÓRIA EEPROM SERIAL

;*****
;*          DECLARAÇÃO DOS FLAGs DE SOFTWARE          *
;*****


; A DEFINIÇÃO DE FLAGs AJUDA NA PROGRAMAÇÃO E ECONOMIZA MEMÓRIA RAM.

#define F_ERRO      FLAG,0          ; 1 --> ERRO NA LEITURA DA EEPROM SERIAL

;*****
;*          ENTRADAS          *
;*****


; AS ENTRADAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#define BOTAO_0      PORTB,0         ; ESTADO DO BOTÃO 0
; 1 -> LIBERADO
; 0 -> PRESSIONADO

#define BOTAO_1      PORTB,1         ; ESTADO DO BOTÃO 1
; 1 -> LIBERADO
; 0 -> PRESSIONADO

#define BOTAO_2      PORTB,2         ; ESTADO DO BOTÃO 2

```

Conectando o PIC 16F877A - Recursos Avançados

; 1 -> LIBERADO
; 0 -> PRESSIONADO

#DEFINE BOTAO_3 PORTB,3 ; ESTADO DO BOTÃO 3

; 1 -> LIBERADO
; 0 -> PRESSIONADO

;

/* SAÍDAS */

; AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#DEFINE DISPLAY PORTD ; BARRAMENTO DE DADOS DO DISPLAY

#DEFINE RS PORTE,0 ; INDICA P/ O DISPLAY UM DADO OU COMANDO
; 1 -> DADO
; 0 -> COMANDO

#DEFINE ENABLE PORTE,1 ; SINAL DE ENABLE P/ DISPLAY
; ATIVO NA BORDA DE DESCIDA

#DEFINE SCL PORTC,3 ; VIA DE CLOCK DA EEPROM
; I/O DEVE ESTAR COMO SAÍDA

/* ENTRADAS/SAÍDAS */

#DEFINE SDA PORTC,4 ; VIA DE DADOS BIDIRECIONAL DA EEPROM
; I/O DEVE INICIAR COMO ENTRADA

;

/* VETOR DE RESET DO MICROCONTROLADOR */

; POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA

```
ORG    0X0000          ; ENDEREÇO DO VETOR DE RESET
GOTO   CONFIG           ; PULA PARA CONFIG DEVIDO A REGIÃO
                           ; DESTINADA AS ROTINAS SEGUINTE
```

;*****

;^{*} ROTINA DE DELAY (DE 1MS ATÉ 256MS) *

;*****

; ESTA É UMA ROTINA DE DELAY VARIÁVEL, COM DURAÇÃO DE 1MS X O VALOR PASSADO
; EM WORK (W).

DELAY_MS

```
MOVWF TEMPO1           ; CARREGA TEMPO1 (UNIDADES DE MS)
MOVLW .250
MOVWF TEMPO0           ; CARREGA TEMPO0 (P/ CONTAR 1MS)
```

CLRWDT ; LIMPA WDT (PERDE TEMPO)

DECFSZ TEMPO0,F ; FIM DE TEMPO0 ?

GOTO \$-2 ; NÃO - VOLTA 2 INSTRUÇÕES

; SIM - PASSOU-SE 1MS

DECFSZ TEMPO1,F ; FIM DE TEMPO1 ?

GOTO \$-6 ; NÃO - VOLTA 6 INSTRUÇÕES

; SIM

RETURN ; RETORNA

;*****

;^{*} ROTINA DE ESCRITA DE UM CARACTER NO DISPLAY *

;*****

; ESTA ROTINA ENVIA UM CARACTER PARA O MÓDULO DE LCD. O CARACTER A SER

; ESCRITO DEVE SER COLOCADO EM WORK (W) ANTES DE CHAMAR A ROTINA.

ESCREVE

MOVWF DISPLAY ; ATUALIZA DISPLAY (PORTD)

NOP ; PERDE 1US PARA ESTABILIZAÇÃO

Conectando o PIC 16F877A - Recursos Avançados

```

        BSF      ENABLE           ; ENVIA UM PULSO DE ENABLE AO DISPLAY
        GOTO    $+1
        BCF      ENABLE           ;;

MOVlw .1
CALL    DELAY_MS          ; DELAY DE 1MS
RETURN             ; RETORNA

;*****
;*          ROTINA DE ESCRITA LINHA 1 DO LCD
;*****          *
; ESTA ROTINA ESCREVE A LINHA 1 DA TELA PRINCIPAL DO LCD, COM A FRASE:
; LINHA 1 - " MASTER I2C "
ATUALIZA_TELA_LINHA_1
BCF      RS               ; SELECCIONA O DISPLAY P/ COMANDOS
MOVLW 0X83                ; COMANDO PARA POSICIONAR O CURSOR
CALL    ESCREVE           ; LINHA 0 / COLUNA 3
BSF      RS               ; SELECCIONA O DISPLAY P/ DADOS
; COMANDOS PARA ESCREVER AS
; LETRAS DE "MASTER I2C"
MOVLW 'M'
CALL    ESCREVE
MOVLW 'A'
CALL    ESCREVE
MOVLW 'S'
CALL    ESCREVE
MOVLW 'T'
CALL    ESCREVE
MOVLW 'E'
CALL    ESCREVE
MOVLW 'R'
CALL    ESCREVE
MOVLW ''

```

```

CALL    ESCREVE
MOVLW '1'
CALL    ESCREVE
MOVLW '2'
CALL    ESCREVE
MOVLW 'C'
CALL    ESCREVE

RETURN           ; RETORNA

;*****
;*          ROTINA DE ESCRITA LINHA 2 DO LCD
;*****          *
; ESTA ROTINA ESCREVE A LINHA 2 DA TELA PRINCIPAL DO LCD.
; A ROTINA LEVA EM CONTA A VARIÁVEL VALOR_DADOS PARA FORMAR A LINHA 2.

ATUALIZA_TELA_LINHA_2
BCF    RS           ; SELECCIONA O DISPLAY P/ COMANDO
MOVLW 0XC6           ; COMANDO PARA POSICIONAR O CURSOR
CALL    ESCREVE      ; LINHA 1 / COLUNA 6

BSF    RS           ; SELECCIONA O DISPLAY P/ DADOS

SWAPF  VALOR_DADOS,W ; INVERTE NIBLE DO VALOR_DADOS
ANDLW  B'00001111'   ; MASCARA BITS MAIS SIGNIFICATIVOS
MOVWF AUX            ; SALVA EM AUXILIAR

MOVLW 0X0A
SUBWF AUX,W          ; AUX - 10d (ATUALIZA FLAG DE CARRY)
MOVLW 0X30            ; CARREGA WORK COM 30h
BTFSR  STATUS,C       ; RESULTADO É POSITIVO? (É UMA LETRA?)
MOVLW 0X37            ; SIM - CARREGA WORK COM 37h
                      ; NÃO - WORK FICA COM 30h (NÚMERO)
ADDWF AUX,W           ; SOMA O WORK AO AUXILIAR
                      ; (CONVERSÃO ASCII)

Conectando o PIC 16F877A - Recursos Avançados

```

```

CALL    ESCREVE          ; ENVIA CARACTER AO DISPLAY LCD

MOVF   VALOR_DADOS,W      ; CARREGA WORK COM VALOR_DADOS
ANDLW  B'00001111'        ; MASCARA BITS MAIS SIGNIFICATIVOS
MOVWF  AUX                ; SALVA EM AUXILIAR

MOVLW  0X0A
SUBWF  AUX,W              ; AUX - 10d (ATUALIZA FLAG DE CARRY)
MOVLW  0X30                ; CARREGA WORK COM 30h
BTFSC  STATUS,C            ; RESULTADO É POSITIVO? (É UMA LETRA?)
MOVLW  0X37                ; SIM - CARREGA WORK COM 37h
                                ; NÃO - WORK FICA COM 30h (NÚMERO)
ADDWF  AUX,W              ; SOMA O WORK AO AUXILIAR
                                ; (CONVERSÃO ASCII)
CALL    ESCREVE          ; ENVIA CARACTER AO DISPLAY LCD

```

```

MOVLW 'h'
CALL    ESCREVE          ; ESCREVE "h" NO DISPLAY
RETURN                         ; RETORNA

```

,*****

;* ROTINA DE CHECAGEM DE EVENTOS I2C LIBERADOS *

,*****

; ESTA ROTINA AGUARDA ATÉ QUE TODOS OS EVENTOS DA I2C ESTEJAM LIBERADOS.

AGUARDA_I2C_LIVRE

```

BANK1                      ; ALTERA P/ BANK1
BTFSC  SSPSTAT,R_W          ; ESTÁ OCORRENDO ALGUM EVENTO I2C?
GOTO   $-1                  ; SIM, ESPERA TERMINAR
MOVF   SSPCON2,W             ; MASCARA SSPCON2 (ATUALIZA FLAG ZERO)
ANDLW  B'00011111'          ; BITS DE EVENTOS LIBERADOS?
BTFSS  STATUS,Z              ; NÃO - AGUARDA
GOTO   $-3                  ; SIM - VOLTA P/ BANK0
BANK0
RETURN                     ; RETORNA

```

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

```
.*          ACK OUT          *
```

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

```
; ESTA ROTINA ENVIA UM ACK OUT PARA O BARRAMENTO I2C.
```

ACK_OUT

```
BANK1          ; ALTERA P/ BANK1  
BCF    SSPCON2,ACKDT   ; CARREGA ACK  
BSF    SSPCON2,ACKEN   ; TRANSMITE  
BANK0          ; VOLTA P/ BANK0  
RETURN         ; RETORNA
```

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

```
.*          NACK OUT          *
```

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

```
; ESTA ROTINA ENVIA UM NACK OUT PARA O BARRAMENTO I2C.
```

NACK_OUT

```
BANK1          ; ALTERA P/ BANK1  
BSF    SSPCON2,ACKDT   ; CARREGA NACK  
BSF    SSPCON2,ACKEN   ; TRANSMITE  
BANK0          ; VOLTA P/ BANK0  
RETURN         ; RETORNA
```

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

```
.*          ROTINA PARA TESTAR SE O ACK FOI RECEBIDO          *
```

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

```
; ESTA ROTINA TESTA O BIT DE ACK RECEBIDO NO REGISTRADOR SSPCON2. PARA
```

```
; FACILITAR O RESTANTE DO SOFTWARE, A ROTINA COPIA ESTE FLAG NO FLAG F_ERRO
```

```
; PRESENTE NO BANCO 0 DA RAM, POIS O REGISTRADOR SSPCON2 ENCONTRA-SE NO BANK1.
```

TESTA_ACK

```
BANK1          ; ALTERA P/ BANK1  
BTFSC  SSPCON2,ACKSTAT   ; RECEBEU ACK ?  
                                Conectando o PIC 16F877A - Recursos Avançados
```

```
GOTO RECEBEU_NACK      ; NÃO - SINALIZA ERRO  
                      ; SIM  
BANK0                ; VOLTA P/ BANK0  
BCF     F_ERRO        ; LIMPA FLAG DE ERRO  
RETURN              ; RETORNA
```

```
RECEBEU_NACK  
BANK0                ; VOLTA P/ BANK0  
BSF     F_ERRO        ; SETA FLAG P/ INDICAR ERRO  
RETURN              ; RETORNA
```

```
;*****  
;*          LEITURA DA EEPROM SERIAL EXTERNA           *  
;*****  
; ESTA ROTINA LÊ A MEMÓRIA SERIAL EXTERNA. O ENDEREÇO DEVE SER PASSADO PELAS  
; VARIÁVEIS ENDERECO_HIGH E ENDERECO_LOW. O VALOR LIDO É RETORNADO EM BUFFER.  
; CASO ALGUM ERRO DE LEITURA OCORRA, A ROTINA DESVIA P/ I2C_ERRO.
```

```
I2C EEPROM READ  
BANK1                ; ALTERA P/ BANK1  
BSF     SSPCON2,SEN    ; INICIA START BIT  
BANK0                ; VOLTA P/ BANK0  
CALL    AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO  
  
BCF     STATUS,C       ; ZERA O CARRY  
RLF     ENDERECO_HIGH,W ; ROTACIONA ENDERECO_HIGH  
IORLW   B'10100000'      ; JUNTA AO BYTE DE CONTROLE  
MOVWF   SSPBUF         ; TRANSMITE CONTROLE + END_HIGH  
CALL    AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO  
CALL    TESTA_ACK       ; CHAMA ROTINA P/ TESTAR ACK  
BTFSC   F_ERRO         ; OCORREU ERRO DE ACK ?  
GOTO    I2C_ERRO        ; SIM - PULA P/ I2C_ERRO  
                      ; NÃO  
MOVF    ENDERECO_LOW,W  
MOVWF   SSPBUF         ; TRANSMITE ENDEREÇO BAIXO
```

```

CALL AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO
CALL TESTA_ACK ; CHAMA ROTINA P/ TESTAR ACK
BTFSC F_ERRO ; OCORREU ERRO DE ACK ?
GOTO I2C_ERRO ; SIM - PULA P/ I2C_ERRO
; NÃO

BANK1 ; ALTERA P/ BANK1
BSF SSPCON2,RSEN ; REINICIA START BIT
BANK0 ; VOLTA P/ BANK0
CALL AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO

RLF ENDERECO_HIGH,W ; ROTACIONA ENDERECO_HIGH
IORLW B'10100001' ; JUNTA AO BYTE DE CONTROLE
MOVWF SSPBUF ; TRANSMITE CONTROLE + END_HIGH
CALL AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO
CALL TESTA_ACK ; CHAMA ROTINA P/ TESTAR ACK
BTFSC F_ERRO ; OCORREU ERRO DE ACK ?
GOTO I2C_ERRO ; SIM - PULA P/ I2C_ERRO
; NÃO

BANK1 ; ALTERA P/ BANK1
BSF SSPCON2,RCEN ; INICIA LEITURA DO BYTE
BANK0 ; VOLTA P/ BANK0
CALL AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO
MOVF SSPBUF,W
MOVWF BUFFER ; SALVA DADO EM BUFFER
CALL NACK_OUT ; ENVIA NACK --> FIM
CALL AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO

BANK1 ; ALTERA P/ BANK1
BSF SSPCON2,PEN ; INICIA STOP BIT
BANK0 ; VOLTA P/ BANK0
CALL AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO

RETURN ; RETORNA

```

Conectando o PIC 16F877A - Recursos Avançados

```

;*****
;*          ESCRITA NA EEPROM SERIAL EXTERNA          *
;*****  

;  

; ESTA ROTINA GRAVA UM DADO NA MEMÓRIA SERIAL EXTERNA. O ENDEREÇO DEVE SER  

; PASSADO PELAS VARIÁVEIS ENDERECO_HIGH E ENDERECO_LOW. O VALOR A SER GRAVADO  

; DEVE SER PASSADO EM BUFFER.  

;  

; CASO ALGUM ERRO DE GRAVAÇÃO OCORRA, A ROTINA DESVIA P/ I2C_ERRO.

```

I2C_EEPROM_WRITE

```

        BANK1           ; ALTERA P/ BANK1
        BSF    SSPCON2,SEN      ; INICIA START BIT
        BANK0           ; VOLTA P/ BANK0
        CALL   AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO

        BCF    STATUS,C       ; ZERA O CARRY
        RLF    ENDERECO_HIGH,W  ; ROTACIONA ENDERECO_HIGH
        IORLW  B'10100000'      ; JUNTA AO BYTE DE CONTROLE
        MOVWF SSPBUF          ; TRANSMITE CONTROLE + END_HIGH
        CALL   AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO
        CALL   TESTA_ACK        ; CHAMA ROTINA P/ TESTAR ACK
        BTFSC F_ERRO          ; OCORREU ERRO DE ACK ?
        GOTO   I2C_ERRO        ; SIM - PULA P/ I2C_ERRO
                                ; NÃO
        MOVF   ENDERECO_LOW,W
        MOVWF SSPBUF          ; TRANSMITE ENDEREÇO BAIXO
        CALL   AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO
        CALL   TESTA_ACK        ; CHAMA ROTINA P/ TESTAR ACK
        BTFSC F_ERRO          ; OCORREU ERRO DE ACK ?
        GOTO   I2C_ERRO        ; SIM - PULA P/ I2C_ERRO
                                ; NÃO
        MOVF   BUFFER,W
        MOVWF SSPBUF          ; GRAVA DADO
        CALL   AGUARDA_I2C_LIVRE ; AGUARDA FIM DO EVENTO
        CALL   TESTA_ACK        ; CHAMA ROTINA P/ TESTAR ACK

```

Conectando o PIC 16F877A - Recursos Avançados

```

BTFSC F_ERRO           ; OCORREU ERRO DE ACK ?

GOTO   I2C_ERRO        ; SIM - PULA P/ I2C_ERRO

                                ; NÃO

BANK1                          ; ALTERA P/ BANK1

BSF    SSPCON2,PEN          ; INICIA STOP BIT

BANK0                          ; VOLTA P/ BANK0

CALL   AGUARDA_I2C_LIVRE    ; AGUARDA FIM DO EVENTO

RETURN                         ; RETORNA

```

;*****

;* ROTINA P/ SINALIZAR ERRO NA I2C *

;*****

; ESTA ROTINA SOMENTE É EXECUTA CASO ALGUM ERRO DE LEITURA/GRAVAÇÃO OCORRA
; COM A MEMÓRIA SERIAL.
; A ROTINA ENVIA UM STOP BIT PARA FINALIZAR A COMUNICAÇÃO COM A MEMÓRIA
; SERIAL, ENVIA UMA MENSAGEM DE ERRO AO DISPLAY E APÓS 1s RETORNA À TELA
; PRINCIPAL.

I2C_ERRO

```

BANK1                          ; ALTERA P/ BANK1

BSF    SSPCON2,PEN          ; INICIA STOP BIT

BANK0                          ; VOLTA P/ BANK0

BCF    RS                  ; SELECCIONA O DISPLAY P/ COMANDO

MOVLW 0XC6                    ; COMANDO PARA POSICIONAR O CURSOR

CALL   ESCREVE               ; LINHA 1 / COLUNA 6

BSF    RS                  ; SELECCIONA O DISPLAY P/ DADOS

MOVLW 'E'

CALL   ESCREVE

MOVLW 'r'

CALL   ESCREVE

MOVLW 'r'

CALL   ESCREVE               ; ESCREVE "Err" NO LCD

```

```
MOVlw .250
CALL    DELAY_MS
MOVlw .250
CALL    DELAY_MS
MOVlw .250
CALL    DELAY_MS
MOVlw .250
CALL    DELAY_MS ; DELAY DE 1seg.
```

```
CALL    ATUALIZA_TELA_LINHA_2 ; ATUALIZA TELA PRINCIPAL
RETURN ; RETORNA
```

```
;*****
;*      CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE *
;*****
; NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS
; CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA AS
; VARIÁVEIS DE RAM E AGUARDA O ESTOURO DO WDT.
```

CONFIG

```
CLRF    PORTA          ; GARANTE TODAS AS SAÍDAS EM ZERO
CLRF    PORTB
CLRF    PORTC
CLRF    PORTD
CLRF    PORTE
```

```
BANK1           ; SELECCIONA BANCO 1 DA RAM
```

```
MOVlw B'11111111'
MOVWF TRISA      ; CONFIGURA I/O DO PORTA
```

```
MOVlw B'11111111'
MOVWF TRISB      ; CONFIGURA I/O DO PORTB
```

```
MOVlw B'11110101'
Conectando o PIC 16F877A - Recursos Avançados
```

MOVWF TRISC ; CONFIGURA I/O DO PORTC

MOVLW B'00000000'

MOVWF TRISD ; CONFIGURA I/O DO PORTD

MOVLW B'00000100'

MOVWF TRISE ; CONFIGURA I/O DO PORTE

MOVLW B'11011111'

MOVWF OPTION_REG ; CONFIGURA OPTIONS
; PULL-UPs DESABILITADOS
; INTER. NA BORDA DE SUBIDA DO RB0
; TIMER0 INCREM. PELO CICLO DE MÁQUINA
; WDT - 1:128
; TIMER - 1:1

MOVLW B'00000000'

MOVWF INTCON ; CONFIGURA INTERRUPÇÕES
; DESABILITADA TODAS AS INTERRUPÇÕES

MOVLW B'00000111'

MOVWF ADCON1 ; CONFIGURA CONVERSOR A/D
; CONFIGURA PORTA E PORTE COMO I/O DIGITAL

MOVLW B'00001001'

MOVWF SSPADD ; VELOCIDADE: 100KHz @ 4MHz

MOVLW B'10000000'

MOVWF SSPSTAT ; DESABILITA SLEW-RATE CONTROL (100
KHz)

BANK0 ; SELECCIONA BANCO 0 DA RAM

MOVLW B'00101000'

MOVWF SSPCON ; HABILITA I2C - MASTER MODE

; CONFIGURA PINOS COMO DA I2C

BSF SCL ; INICIALIZA SCL EM 1

; AS INSTRUÇÕES A SEGUIR FAZEM COM QUE O PROGRAMA TRAVE QUANDO HOUVER UM
; RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT. DESTA FORMA,
; SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA UM ESTOURO DE WDT
; E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO START-UP DO PIC.

BTFSC STATUS,NOT_TO ; RESET POR ESTOURO DE WATCHDOG TIMER ?
GOTO \$; NÃO - AGUARDA ESTOURO DO WDT
; SIM

:* INICIALIZAÇÃO DA RAM *

; ESTA ROTINA IRÁ LIMPAR TODA A RAM DO BANCO 0, INDO DE 0X20 A 0X7F

MOVlw 0X20

MOVWF FSR ; APONTA O ENDEREÇAMENTO INDIRETO PARA
; A PRIMEIRA POSIÇÃO DA RAM

LIMPA_RAM

CLRF INDF ; LIMPA A POSIÇÃO
INCF FSR,F ; INCREMENTA O PONTEIRO P/ A PRÓX. POS.
MOVF FSR,W
XORLW 0X80 ; COMPARA O PONTEIRO COM A ÚLT. POS. +1
BTFSS STATUS,Z ; JÁ LIMPOU TODAS AS POSIÇÕES?
GOTO LIMPA_RAM ; NÃO - LIMPA A PRÓXIMA POSIÇÃO
; SIM

:* CONFIGURAÇÕES INICIAIS DO DISPLAY *

; ESTA ROTINA INICIALIZA O DISPLAY P/ COMUNICAÇÃO DE 8 VIAS, DISPLAY PARA 2

; LINHAS, CURSOR APAGADO E DESLOCAMENTO DO CURSOR À DIREITA.

Conectando o PIC 16F877A - Recursos Avançados

INICIALIZACAO_DISPLAY

```
BCF      RS           ; SELECCIONA O DISPLAY P/ COMANDOS
MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
CALL    ESCREVE       ; INICIALIZAÇÃO

MOVLW .3
CALL    DELAY_MS      ; DELAY DE 3MS (EXIGIDO PELO DISPLAY)

MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
CALL    ESCREVE       ; INICIALIZAÇÃO

MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
CALL    ESCREVE       ; INICIALIZAÇÃO

MOVLW B'00111000'    ; ESCREVE COMANDO PARA
CALL    ESCREVE       ; INTERFACE DE 8 VIAS DE DADOS

MOVLW B'00000001'    ; ESCREVE COMANDO PARA
CALL    ESCREVE       ; LIMPAR TODO O DISPLAY

MOVLW .1
CALL    DELAY_MS      ; DELAY DE 1MS

MOVLW B'00001100'    ; ESCREVE COMANDO PARA
CALL    ESCREVE       ; LIGAR O DISPLAY SEM CURSOR

MOVLW B'00000110'    ; ESCREVE COMANDO PARA INCREM.
CALL    ESCREVE       ; AUTOMÁTICO À DIREITA

BSF      RS           ; SELECCIONA O DISPLAY P/ DADOS
```

;*****

/* INICIALIZAÇÃO DA RAM */
;

; Conectando o PIC 16F877A - Recursos Avançados

; ESTE TRECHO DO PROGRAMA LÊ OS DADOS DA MEMÓRIAS E2PROM EXTERNA E
; ATUALIZA A RAM.

LE_MEMORY_EEPROM

```
    MOVLW END EEPROM_H
    MOVWF ENDERECO_HIGH
    MOVLW END EEPROM_L
    MOVWF ENDERECO_LOW      ; CARREGA ENDEREÇO P/ LEITURA
```

```
    CALL I2C EEPROM READ      ; CHAMA ROTINA P/ LER DADO
```

```
    MOVF BUFFER,W
```

```
    MOVWF VALOR_DADOS      ; SALVA DADO LIDO EM VALOR_DADOS
```

* ROTINA DE ESCRITA DA TELA PRINCIPAL *

; ESTA ROTINA ESCREVE A TELA PRINCIPAL DO PROGRAMA, COM AS FRASES:

; LINHA 1 - " MASTER I2C "

; LINHA 2 - " xxh "

```
    CALL ATUALIZA_TELA_LINHA_1 ; ATUALIZA TELA LINHA 1 DO LCD
```

```
    CALL ATUALIZA_TELA_LINHA_2 ; ATUALIZA TELA LINHA 2 DO LCD
```

* VARREDURA DOS BOTÕES *

; ESTA ROTINA VERIFICA SE ALGUM BOTÃO ESTÁ PRESSIONADO E CASO AFIRMATIVO

; DESVIA PARA O TRATAMENTO DO MESMO.

VARRE

```
    CLRWDAT      ; LIMPA WATCHDOG TIMER
```

***** VERIFICA ALGUM BOTÃO PRESSIONADO *****

Conectando o PIC 16F877A - Recursos Avançados

VARRE_BOTOES

BTFS S BOTAO_0 ; O BOTÃO 0 ESTA PRESSIONADO ?

GOTO TRATA_BOTAO_0 ; SIM - PULA P/ TRATA_BOTAO_0

; NÃO

BTFS S BOTAO_1 ; O BOTÃO 1 ESTA PRESSIONADO ?

GOTO TRATA_BOTAO_1 ; SIM - PULA P/ TRATA_BOTAO_1

; NÃO

BTFS S BOTAO_2 ; O BOTÃO 2 ESTA PRESSIONADO ?

GOTO TRATA_BOTAO_2 ; SIM - PULA P/ TRATA_BOTAO_2

; NÃO

BTFSS BOTA0_3 ; O BOTÃO 3 ESTA PRESSIONADO ?

GOTO TRATA_BOTAO_3 ; SIM - PULA P/ TRATA_BOTAO_3

; NÃO

***** FILTRO P/ EVITAR RUIDOS *****

MOVLW FILTRO_TECLA ; CARREGA O VALOR DE FILTRO_TECLA

MOVWF FILTRO_BOTOES ; SALVA EM FILTRO_BOTOES

; RECARREGA FILTRO P/ EVITAR RUIDOS

; NOS BOTÕES

MOVLW .1

MOVWF TEMPO_TURBO ; CARREGA TEMPO DO TURBO DAS TECLAS

; COM 1 - IGNORA O TURBO A PRIMEIRA

; VEZ QUE A TECLA E PRESSIONADA

GOTO VARRE ; VOLTA PARA VARRER TECLADO

,

TRATAMENTO DOS BOTÕES

*

; NESTE TRECHO DO PROGRAMA ESTÃO TODOS OS TRATAMENTOS DOS BOTÕES
Conectando o PIC 16F877A - Recursos Avançados

; ***** TRATAMENTO DO BOTÃO 0 *****

TRATA_BOTAO_0

```
DECFSZ FILTRO_BOTOES,F      ; FIM DO FILTRO ? (RUIDO?)  
GOTO    VARRE           ; NÃO - VOLTA P/ VARRE  
                      ; SIM - BOTÃO PRESSIONADO  
  
DECFSZ TEMPO_TURBO,F      ; FIM DO TEMPO DE TURBO ?  
GOTO    VARRE           ; NÃO - VOLTA P/ VARRE  
                      ; SIM  
MOVlw  TURBO_TECLA  
MOVWF TEMPO_TURBO        ; RECARREGA TEMPORIZADOR DO TURBO  
                      ; DAS TECLAS  
  
INCF    VALOR_DADOS,F    ; INCREMENTA VALOR_DADOS
```

```
CALL    ATUALIZA_TELA_LINHA_2 ; CHAMA ROTINA P/ ATUALIZAR LCD
```

```
GOTO    VARRE           ; VOLTA P/ VARREDURA DOS BOTÕES
```

; ***** TRATAMENTO DO BOTÃO 1 *****

TRATA_BOTAO_1

```
DECFSZ FILTRO_BOTOES,F      ; FIM DO FILTRO ? (RUIDO?)  
GOTO    VARRE           ; NÃO - VOLTA P/ VARRE  
                      ; SIM - BOTÃO PRESSIONADO  
  
DECFSZ TEMPO_TURBO,F      ; FIM DO TEMPO DE TURBO ?  
GOTO    VARRE           ; NÃO - VOLTA P/ VARRE  
                      ; SIM  
MOVlw  TURBO_TECLA  
MOVWF TEMPO_TURBO        ; RECARREGA TEMPORIZADOR DO TURBO  
                      ; DAS TECLAS
```

```
DECF    VALOR_DADOS,F           ; DECREMENTA VALOR_DADOS
```

```
CALL    ATUALIZA_TELA_LINHA_2 ; CHAMA ROTINA P/ ATUALIZAR LCD
```

```
GOTO    VARRE                 ; VOLTA P/ VARREDURA DOS BOTÕES
```

```
; ***** TRATAMENTO DO BOTÃO 2 *****
```

```
TRATA_BOTAO_2
```

```
MOVF    FILTRO_BOTOES,F
```

```
BTFSZ   STATUS,Z              ; FILTRO JÁ IGUAL A ZERO ?
```

```
; (FUNÇÃO JA FOI EXECUTADA?)
```

```
GOTO    VARRE                 ; SIM - VOLTA P/ VARREDURA DO TECLADO
```

```
; NÃO
```

```
DECFSZ FILTRO_BOTOES,F       ; FIM DO FILTRO ? (RUIDO?)
```

```
GOTO    VARRE                 ; NÃO - VOLTA P/ VARRE
```

```
; SIM - BOTÃO PRESSIONADO
```

```
; *****TRECHO DO PROGRAMA PARA GRAVAR DADO DA RAM NA MEMÓRIA *****
```

```
GRAVA_MEMORY_EEPROM
```

```
MOVLW  END_EEPROM_H
```

```
MOVWF  ENDERECHO_HIGH
```

```
MOVLW  END_EEPROM_L
```

```
MOVWF  ENDERECHO_LOW         ; CARREGA ENDEREÇO ONDE O DADO SERÁ SALVO
```

```
; END. -> 0x0000
```

```
; PRIMEIRA POSIÇÃO DA EEPROM
```

```
MOVF    VALOR_DADOS,W
```

```
MOVWF  BUFFER                ; CARREGA DADO A SER SALVO EM BUFFER
```

```
CALL    I2C_EEPROM_WRITE      ; CHAMA ROTINA DE GRAVAÇÃO
```

```
MOVLW  .10
```

```
CALL    DELAY_MS               ; GARANTE TEMPO DE ESCRITA (10ms)  
Conectando o PIC 16F877A - Recursos Avançados
```

```
GOTO    VARRE           ; VOLTA P/ VARREDURA DOS BOTÕES
```

```
; ***** TRATAMENTO DO BOTÃO 3 *****
```

```
TRATA_BOTAO_3
```

```
MOVF    FILTRO_BOTOES,F
BTFSC   STATUS,Z      ; FILTRO JÁ IGUAL A ZERO ?
; (FUNÇÃO JA FOI EXECUTADA?)
GOTO    VARRE          ; SIM - VOLTA P/ VARREDURA DO TECLADO
; NÃO
DECFSZ FILTRO_BOTOES,F      ; FIM DO FILTRO ? (RUIDO?)
GOTO    VARRE          ; NÃO - VOLTA P/ VARRE
; SIM - BOTÃO PRESSIONADO
```

```
; ***** TRECHO DO PROGRAMA PARA LER DADO DA MEMÓRIA E ATUALIZAR RAM *****
```

```
LER_MEMORY_EEPROM
```

```
MOVLW  END_EEPROM_H
MOVWF ENDERECO_HIGH
MOVLW  END_EEPROM_L
MOVWF ENDERECO_LOW       ; CARREGA ENDEREÇO DE LEITURA
; END. -> 0x0000
; PRIMEIRA POSIÇÃO DA EEPROM
```

```
CALL    I2C_EEPROM_READ      ; CHAMA ROTINA DE LEITURA
```

```
BTFSC  F_ERRO
```

```
GOTO    $+3
```

```
MOVF    BUFFER,W
```

```
MOVWF  VALOR_DADOS        ; ATUALIZA RAM COM O VALOR LIDO
```

```
CALL    ATUALIZA_TELA_LINHA_2 ; CHAMA ROTINA P/ ATUALIZAR LCD
```

```
GOTO    VARRE           ; VOLTA P/ VARREDURA DOS BOTÕES
Conectando o PIC 16F877A - Recursos Avançados
```

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

Dicas e comentários

As constantes END_EEPROM_H e END_EEPROM_L representam a posição a ser gravada/lida da memória externa.

Este programa não utiliza as interrupções e possui uma rotina (AGUARDAJ2CJJVRE) para saber se o sistema está liberado para a próxima ação. Ele também testa o ACK e gera uma mensagem de erro, caso alguma coisa saia fora do padrão.

Exercícios propostos

Agora que o exemplo já foi estudado e esclarecido, aproveite para gerar novos problemas e soluções, seguindo os exercícios propostos:

1. Faça três modificações no primeiro exercício proposto no capítulo 10.
 - Utilize a memória externa;
 - Limite os dados mostrados no display entre 0x41 e 0x5A;
 - Mostre os dados em ASCII, ou seja, entre A (0x41) e Z (0x5A).
2. Utilizando o exercício anterior, grave na memória uma mensagem de até 16 caracteres. Depois, crie um programa que, ao ser inicializado, leia os 16 caracteres da memória e mostre a mensagem lida no LCD.

Anotações

Comunicação Serial 2 - USART

Introdução

Esta é a segunda parte sobre comunicação serial e veremos agora outro módulo interno do PIC destinado a esse recurso. Trata-se da USART. A grande vantagem da separação dos recursos de comunicação em dois módulos é que eles são independentes, podendo ser utilizados simultaneamente.

O nome USART significa Universal Synchronous Asynchronous Receiver Transmitter. Com um nome desse, o negócio parece um tanto complicado, não é mesmo? Mas não é complicado não. Acontece que esse é um protocolo universal e possui dois modos distintos de trabalho: o sincronizado e o não-sincronizado. Mas vamos logo ao que interessa e tratemos de conhecer melhor este recurso.

Teoria

Como estávamos dizendo, a USART, que também é conhecida como SCI (Serial Communications Interface), possui dois modos de funcionamento, vejamos as características de cada um deles:

Modo assíncrono

A comunicação é feita somente com duas vias; entretanto, como este modo não é sincronizado, essas duas vias são utilizadas para dados. Uma delas para transmissão (TX) e a outra para recepção (RX). Isso possibilita que as informações sejam enviadas e recebidas ao mesmo tempo, cada qual na sua via. Este recurso é conhecido como Full Duplex. Esse modo é o utilizado, por exemplo, na porta serial dos computadores, para implementar o padrão RS-232, mas pode ser utilizado para acesso a outros sistemas também.

Mas como é possível os dados serem transmitidos entre dois pontos se não há sincronismo entre eles? Quando estudamos a comunicação SSP, vimos que uma via era perdida exatamente para essa função. Era definida como clock do sistema e servia para informar os dois lados (Master e Slave) do momento correto de transmissão de cada bit. Como aqui não há essa via, a sincronização deve ser feita pela própria via de dados. Isso será conseguido através do Baud Rate ou velocidade de transmissão. Vejamos como funciona.

Comecemos definindo exatamente o que é o Baud Rate. Para que o sistema funcione, veremos que o tamanho dos dados (intervalo de cada bit) deve ser completamente padronizado, e ambos os lados devem estar ajustados para o mesmo valor. Como essa comunicação trabalha sempre com base nos

bits, essa velocidade é normalmente indicada em bits por segundo, ou bps. Com ela somos capazes de calcular o tempo de duração de cada bit.

$$T_{\text{BIT}} = 1/\text{BaudRate}$$

Com isso, existe somente um sincronismo de tempo feito para a transmissão/recepção de cada byte. Esse sincronismo é conseguido através do Start bit.

Devemos entender também que ambas as vias devem ser tratadas igualmente, pois o TX de um lado deve estar conectado ao RX do outro, e vice-versa. Em ambos os lados, TX é sempre saída e RX é sempre entrada. Desta forma, quando falarmos de transmissão ou recepção, serve para qualquer uma das vias.

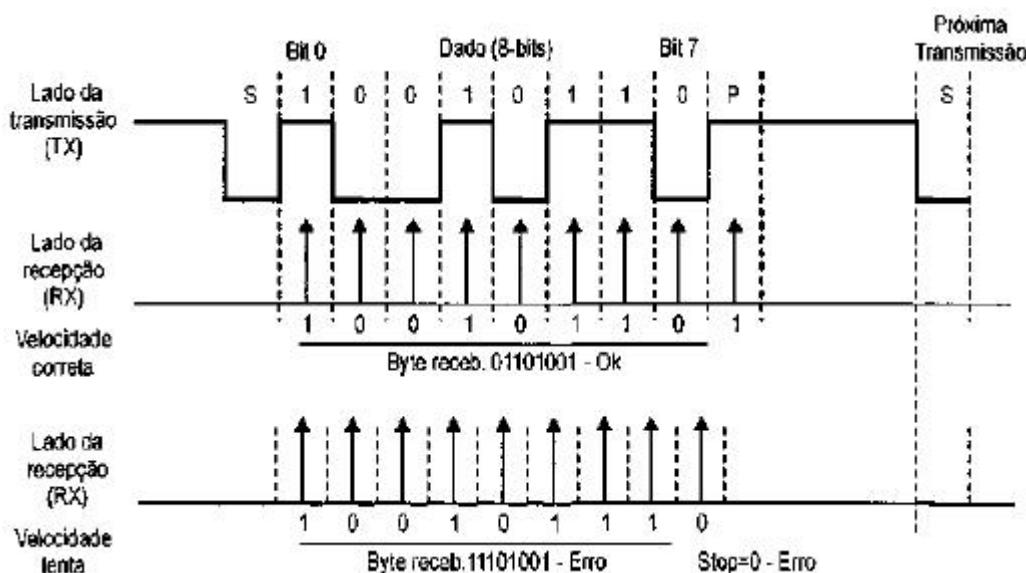
As vias possuem seu estado padrão como sendo o nível alto. Temos, então, uma situação de stand-by. Quando um lado inicia uma transmissão, ele força seu TX para nível baixo, mantendo-o assim pelo tempo T_{BIT} . Essa borda de descida é reconhecida pelo outro lado (em RX) e é suficiente para iniciar o processo de sincronização para recebimento desse byte. Este pulso em nível baixo é chamado de Start Bit.

Depois disso, os dois lados já sabem o que fazer. TX enviará então os 8 bits de dados, todos eles com o mesmo tamanho do Start Bit. Como RX soube exatamente o momento de início do Start Bit, ele deixa passar o tempo e depois coleta os 8 bits, pegando o dado mais ou menos no meio do tempo do bit ($T_{\text{BIT}}/2$).

Por último, para garantir o processo, TX envia um Stop Bit, que nada mais é que outro bit com valor fixo em 1, garantindo assim que a linha voltará ao seu estado padrão e o sistema voltará ao stand-by ficando apto ao próximo dado. O lado RX deve considerar a leitura do Stop Bit para garantir que nenhum erro grosseiro aconteça com a recepção. Caso o Stop Bit seja 0 (zero), pode ter acontecido um erro de temporização, e no seu lugar foi lido o bit 8 ou, então, o próximo Start.

Repare também que o erro desse processo é acumulativo. Por exemplo, caso o lado TX esteja cor sua velocidade no limite superior do erro, e o RX com a velocidade menor do que devia, cada bit será lido mais perto do começo do pulso. Como existem 10 bits ao total (Start + Dado + Stop), é possível que no final, aconteça a leitura errada de um bit.

Vejamos o processo graficamente:



O primeiro caso de recepção está com a velocidade bem próxima à velocidade do transmissor, e por isso não houve erro na recepção. Já no outro caso a velocidade está mais lenta, e houve erro no último bit e no Stop Bit. Por isso, é importante o acerto e a precisão da velocidade em ambos os lados. Quanto maior o Baud Rate, mais crítica é a situação.

Observe também que a ordem de transmissão dos bits é a inversa das comunicações SSP. Aqui, o bit menos significativo é enviado primeiro.

Esse padrão aceita também a comunicação com 9 bits, sendo que o bit adicional poder ser utilizado para dado, paridade ou endereçamento. A paridade nada mais é que uma confirmação matemática dos 8 bits de dados. Somando-se a quantidade de bits em 1 (incluindo dados e paridade), o resultado correto deve ser um número par (quando utilizando paridade PAR) ou ímpar (quando utilizando paridade IMPAR). Quanto ao endereçamento, respeita-se o seguinte critério: 0 para dado e 1 para endereço.

O importante é que ambos os lados (TX e RX) estejam configurados para operar com a mesma quantidade de bits. A configuração mais comum é a de 8 bits de dado (sem paridade) com 1 bit de Stop e é normalmente chamada de padrão 8N1.

No caso de comunicações padronizadas, o Baud Rate (BR) também obedece a valores pré--ajustados, tais como 300, 1.200, 2.400, 9.600, 19.200bps e muitos outros.

Modo síncrono

Este modo pode ser considerado como uma certa mistura entre os padrões SPI e I²C.

Assim como no modo assíncrono, aqui também trabalhamos com somente duas vias, só que neste caso uma é destinada ao clock (CK) e a outra aos dados (DT). Desta forma, os dados devem trafegar em uma única via, impossibilitando a transmissão e recepção simultâneas. É o mesmo conceito utilizado no padrão I2C. Essa comunicação é chamada de Half Duplex e pode ser utilizada para a troca de dados com outros microcontroladores ou diversos periféricos existentes no mercado, tais como A/Ds, D/As, memórias, etc.

Quanto à forma em que a informação trafega na linha, é bem mais simples que o padrão I2C, não possuindo o sistema de endereçamento e parecendo-se mais com o formato SPI. Para cada pulso (borda de descida) é transmitido um bit.

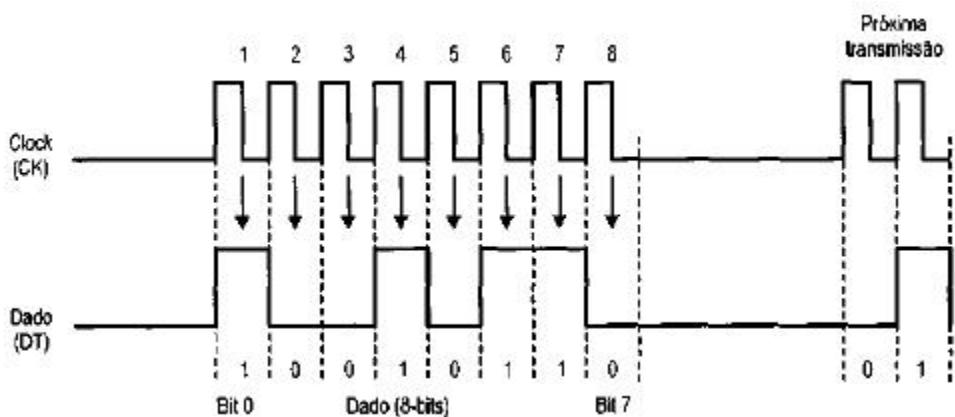
Este modo também opera com Mestre e Escravo, sendo o clock sempre gerenciado pelo Mestre. Para o Mestre, a via CK é sempre uma saída e para o Escravo ela é sempre uma entrada. Quanto à via de dados, ela muda constantemente de sentido, hora para a transmissão, hora para a recepção. Assim sendo, para qualquer uma das pontas, a via DT é saída para a transmissão e entrada para recepção.

O tempo de duração de um bit também define o Baud Rate, do mesmo modo descrito na comunicação assíncrona.

Neste caso, o nível baixo (0) é o padrão para a via CK. Para a via DT, não existe um padrão obrigatório, pois ela não opera sem CK. Porém, recomendamos mantê-la também em nível baixo quando não está sendo usada.

Observe que, neste padrão, também o bit menos significativo (bit 0) é enviado primeiro.

Para o modo síncrono também é aceita a comunicação com 9 bits e, neste caso, o Mestre sempre gerará pacotes de nove pulsos de clock. O importante é que ambos os lados da comunicação (Mestre e Escravo) estejam configurados para operar com 9 bits.



Recursos do PIC

Vamos estudar agora como operar com a USART do PIC 16F877A. Para continuarmos com a mesma linha de raciocínio, dividiremos mais uma vez parte do assunto entre os dois modos de operação, mas antes podemos explicar a maioria dos conceitos que são compartilhados por ambos.

Os pinos da comunicação (**RC6/TX/CK**) e (**RC7/RX/DT**) são os mesmos nos dois modos e são controlados diretamente pelo sistema da USART. Somente para evitarmos possíveis conflitos, caso a USART seja desabilitada, é recomendável que ambos sejam ajustados como entrada através do TRISC.

A definição entre os modos de operação é feita através de **TXSTA<SYNC>**:

SYNC	Descrição
0	Modo Assíncrono.
1	Modo Síncrono.

O ajuste do Baud Rate (BR) é feito por meio de um registrador denominado **SPBRG** e do bit **TXSTA<BRGH>**. A combinação desses parâmetros e do modo de operação definem o cálculo do BR:

BRGH	Descrição
0	Ajuste para baixa velocidade.
1	Ajuste para alta velocidade.

SYNC	BRGH=0	BRG=1
	BR = $F_{osc}/(64 \times (\text{SPBRG}=1))$	BR = $F_{osc}/(16 \times (\text{SPBRG}+1))$
1	BR = $F_{osc}/(4 \times (\text{SPBRG}=1))$	Não válido

Primeiramente observe que o ajuste de **BRGH** não tem efeito quando estamos trabalhando no modo síncrono. Neste caso, o tempo de cada bit (TBIT) será múltiplo do tempo de ciclo de máquina (TCY). Para o menor ajuste possível (**SPBRG=0**), teremos $T_{BIT}=TCY$. Nesta situação, a parte alta do clock é gerada nos subtempos Q4 e Q1, ficando Q2 e Q3 para a parte baixa.

Por exemplo, para uma Fosc de 4 MHz, teremos:

- $BR_{MAX} = 1.000.000 \text{bps}$
- $BR_{MIN} = 3.906 \text{bps}$

Para o modo síncrono, **BRGH** surte efeito alterando a fórmula para o cálculo de BR. Observe que, quando **BRGH=0**, tbit será múltiplos de $64T_{CY}$. Isso diminui a velocidade máxima e aumenta o erro para nos aproximarmos das velocidades padronizadas. Para o segundo caso (**BRGH=1**), teremos um ajuste mais preciso, pois tbit será múltiplos de $16T_{CY}$. Agora teremos um erro menor e um aumento da velocidade máxima. Por outro lado, perdemos na velocidade mínima.

Peguemos o mesmo exemplo dado para o caso do modo síncrono (Fosc = 4 MHz), primeiro para **BRGH=0**:

- $BR_{MAX} = 62.500 \text{bps}$
- $BR_{MIN} = 244 \text{bps}$

E agora para **BRGH=1**:

- $BR_{MAX} = 250.000 \text{bps}$
- $BR_{MIN} = 976 \text{bps}$

Vamos, agora, calcular o valor de **SPBRG** para uma velocidade de 9.600bps, com cristal de 4 MHz e **BRGH=0**:

$$BR = Fosc/(64x(SPBRG+1))$$

$$SPBRG = (Fosc/(64 \times BR)) - 1$$

$$SPBRG = (4.000.000 / (64 \times 9.600)) - 1$$

$$SPBRG = 5,510 \rightarrow 5$$

Como houve um arredondamento, existirá um erro que deve ser calculado:

$$BR = 4.000.000 / (64 \times (5+1))$$

$$BR = 10.416 \text{bps}$$

$$Erro = (10.416 - 9.600) / 9.600$$

$$Erro = 8,5\% \text{ (muito elevado)}$$

Vejamos o mesmo caso para **BRGH=1**:

$$BR = Fosc/(16x(SPBRG+1))$$

$$SPBRG = (Fosc/(16 \times BR)) - 1$$

$$SPBRG = (4.000.000 / (16 \times 9600)) - 1$$

$$SPBRG = 25,042 \rightarrow 25$$

$$BR = 4.000.000 / (16 \times (25+1))$$

$$BR = 9.615 \text{ bps}$$

$$\text{Erro} = (9.615 - 9.600) / 9.600$$

$$\text{Erro} = \mathbf{0,16\%} \text{ (muito bom!)}$$

Devido à grande diferença de erro, devemos adotar o segundo cálculo. Existem casos, principalmente quando BR é elevado, que para melhorarmos a situação será necessário alterarmos o valor do oscilador (Fosc), tentando chegar em números mais precisos. Não esqueça de que existe ainda a tolerância de funcionamento do sistema de oscilação. O uso de cristais pode também ser obrigatório para as velocidades mais altas.

Para ativar o sistema da USART, configurando os pinos corretamente, deve-se ajustar o bit **RCSTA<SPEN>**:

SPEN	Descrição
0	USART desabilitada. Pinos como I/Os convencionais.
1	USART habilitada. Pinos controlados automaticamente.

A partir deste ponto o usuário deve escolher entre as operações de transmissão e/ou recepção, mas, como elas são bem diferentes para os modos assíncronos e síncronos, serão explicadas na divisão dos tópicos. Por enquanto, vejamos somente quais são e os bits que as controlam:

Operação	Bit	Observações
Transmissão Recepção unitária	TXSTA<TXEN> RCSTA<SREN>	Ativa o sistema de transmissão de bytes. Ativa o sistema de recepção de somente 1 byte. Somente para modo síncrono.
Recepção contínua	RCSTA<CREN>	Ativa o sistema de recepção contínua.

O que devemos explicar, que ainda é comum aos modos de operação, diz respeito ao dado que será transmitido e/ou recebido.

Primeiramente devemos optar pela comunicação com 8 ou 9 bits. Isso é feito separadamente para a transmissão e para a recepção, através dos bits **TXSTA<TX9>** e **RCSTA<RX9>**:

TX9	Descrição
0	Transmissão feita em 8 bits.
1	Transmissão feita em 9 bits.

RX9	Descrição
0	Recepção feita em 8 bits.
1	Recepção feita em 9 bits.

Este 9º bit deve ser escrito em **TXSTA<TX9D>** para a transmissão e será recebido em **RCSTA<RX9D>** do lado da recepção. O PIC não possui sistema automático para implementação desse bit como sendo a paridade. Se o uso da paridade for necessária, essa implementação terá de ser feita através do software. Existe um auxílio para uso desse bit como endereçamento, mas isso será visto posteriormente.

Quanto ao dado propriamente dito, para o caso da transmissão, ele deve ser escrito no registrador **TXREG**. Caso o sistema de transmissão esteja ligado, a simples escrita nesse registrador irá iniciá-la. Este dado será, então, transferido a um registrador interno denominado **TSR** (sem acesso pelo programa) para que seja enviado pela porta serial. Neste momento, **TXREG** fica vazio e liberado para uma nova escrita. Desta forma, o nosso buffer de saída é duplo, podendo haver um byte em **TXREG** e outro em **TSR**. O segundo byte só será passado de **TXREG** para **TSR** quando o último bit (7 ou 8) do byte anterior for transmitido, deixando **TSR** vazio.

Toda vez que o valor for passado de **TXREG** para **TSR**, o bit **PIR1<TXIF>** será setado, podendo gerar a interrupção. O interessante é que esse bit não precisa ser limpo manualmente como os demais flags de interrupção. Ele será limpo automaticamente sempre que **TXREG** for escrito. Isso serve para sabermos quando **TXREG** está liberado para uma nova escrita. O problema é que, se não desejamos transmitir nada, não escreveremos em **TXREG** e não limparemos **TXIF**. Isso irá travar o programa entrando sem parar na interrupção. A solução, para este caso, é desligar a interrupção ou mesmo a transmissão.

O bit **TXIF** não pode ser utilizado para sabermos que uma transmissão já foi completada, pois, depois de **TXREG** ser colocado em **TSR**, o dado ainda precisará ser enviado pela porta serial. Para saber quando realmente a transmissão foi finalizada, devemos saber quando **TSR** está vazio. O bit **TXSTA<TRMT>** possui esta função:

TRMF	Descrição
1	Transmissão finalizada. TSR liberado.
0	Transmissão em curso. TSR ocupado.

Para a recepção, o dado recebido é primeiramente armazenado em um registrador interno não--acessível (RSR). Quando esse registrador está completo, a informação é então passada para o registrador RCREG. O mais interessante é que este registrador possui dois níveis de pilha, podendo ser escrito e lido duas vezes. Com isso nosso buffer de entrada é triplo, podendo haver um dado sendo recebido em RSR e mais dois já recebidos em RCREG. O registrador RCREG trabalha com o sistema FIFO (*First In/First Out*), isto é, o primeiro a ser recebido será o primeiro a ser lido.

Cada vez que o valor de **RSR** é transferido para **RCREG** o bit **PIR1<RCIF>** é setado, podendo gerar a interrupção. Como no caso da transmissão, esse bit é limpo automaticamente pelo hardware sempre que **RCREG** estiver vazio. Isso significa que, se foram recebidos 2 bytes antes de ser efetuada uma conferência, **RCREG** terá de ser lido duas vezes (pilha) para que o bit **RCIF** seja limpo. Caso o terceiro byte seja completado antes da leitura dos duas já existentes em **RCREG**, um erro de overflow irá acontecer e o bit **RCSTA<OERR>** será setado. O 3º byte que se encontra em **RSR** será perdido. Este bit deve ser limpo manualmente. Isso também pode ser conseguido desativando-se o modo de recepção.

Para trabalhar com o 9º bit, é necessário que ele seja escrito (**TX9D**) antes do dado em **TXREG**, possibilitando a atualização correta do **TSR**.

O sistema de recepção checa também o Stop Bit, mas não toma nenhuma atitude automática em caso de erro. A situação do Stop Bit é armazenada em **RCSTA<FERR>**:

FERR	Descrição
0	0 Stop Bit foi recebido corretamente (Stop Bit = 1).
1	0 Stop Bit foi recebido errado (Stop Bit = 0).

Para que esse bit possa ser checado pelo programa, assim como o 99 (quando usado), uma leitura deve ser feita antes em **RCREG** para que os mesmos sejam atualizados.

Por último devemos comentar sobre o sistema de endereçamento. Este sistema só funciona quando ajustada a comunicação para 9 bits que é controlado através do bit **RCSTA<ADDEN>**:

ADDEN	Descrição
0	Sistema de endereçamento desativado.
1	Sistema de endereçamento ativado.

Quando esse sistema está ativo, o dado recebido só é transferido de **RSR** para **RCREG** quando o {f for 1, podendo então gerar a interrupção. Caso seja recebido uma informação com o 9- em O, ela será reconhecida como dado e será descartada. Para que o processo funcione corretamente é necessário, então, começarmos com o endereçamento ativo para recebermos um endereço que será tratado e comparado como o endereço da própria unidade. O endereço será o byte recebido, possibilitando 256 valores diferentes. Caso o endereço recebido corresponda à unidade em questão, o sistema de endereçamento deve então ser desativado para que o próximo valor possa ser recebido como um dado válido.

Vejamos agora as particularidades e observações do roteiro de trabalho de cada modo.

Modo assíncrono

Para o modo assíncrono, quando habilitamos a USART através do bit **RCSTA<SPEN>**, o pino **TX** é transformado em saída com nível alto (1) e o pino **RX** é transformado em entrada.

Para ativar o sistema de recepção, basta tornar **RCSTA<CREN>=1**. Para esse modo não é possível escolher o sistema de recepção unitária e, por isso, **RCSTA<SREN>** não possui função.

A partir deste momento cada byte recebido será colocado em **RCREG** e a interrupção será ativada através de **RCIF**. A lógica do sistema será, então, tratar das informações recebidas. É recomendável também a implementação da checagem do Stop Bit (**FERR**) e do estouro de recepção (**OERR**).

Para ativar o sistema de transmissão, basta tornar **TXSTA<TXEN>=1**. O importante é lembrar de que os sistemas de transmissão e recepção são totalmente independentes (só compartilham o mesmo Baud Rate) e, por isso, cada um deles pode ser ativado e desativado, conforme a necessidade.

Depois da ativação, basta escrever um valor em **TXREG** e o mesmo será enviado automaticamente pela porta serial, com a geração do Sfart B/f e do Stop B/f. O importante é a checagem do bit **TXIF** para que o sistema não sobre escreva **TXREG** antes de **TSR** ser atualizado.

Quando um dos sistemas é desativado (Transmissão ou Recepção), o pino relacionado a ele é mantido como entrada.

Este modo não pode operar em SLEEP.

Resumo dos registradores associados à USART Assíncrona

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0Bh...	INTCON	GIE	PEIE	TOIE	INTE	RBIE	T0IF	INTF	RBIF
OCh	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCPIF	TMR2IF	TMR1IF
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
98h	TXSTA	CSRC	TX9	TXEN	SYNC	-	BRGH	TMRT	TX9D
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
19h	TXREG	Buffer de transmissão							
1Ah	RCREG	Buffer de recepção							
99h	SPBRG	Acerto do Baud Rate							

Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Modo síncrono

Para este modo existe mais um bit de configuração relacionado ao sistema Master/Slave. Trata-se do **TXSTA<CSRC>**:

CSRC	Descrição
0	Slave (CK como entrada).
1	Master (CK como saída).

Com essa seleção, o sistema configura corretamente o estado do pino de clock (CK). O estado do pino de dados (**DT**) será variável, conforme o sistema se encontre em recepção ou transmissão. Devemos, então, escolher qual operação desejamos efetuar:

Operação	Bit	Observações
Transmissão	TXSTA<TXEN>	O pino DT será colocado como saída e o sistema está pronto para transmitir o dado escrito em TXREG. A escolha de um modo de recepção tem prioridade sobre este modo.
Recepção unitária	RCSTA<SREN>	pino DT é colocado como entrada e o sistema de recepção de somente 1 byte é ativado.
Recepção contínua	RCSTA<CREN>	pino DT é colocado como entrada e o sistema de recepção contínua é ativado.

Transmissão

No caso do Master, o sistema funciona de forma muito parecida com o descrito anteriormente para o modo assíncrono, operando da mesma forma com **TXREG**, **TXIF** e **TRMT**. A única diferença é que, quando escrevemos algum dado em TXREG, o mesmo será transmitido por **DT**, com pulsos sendo gerados em CK e sem a presença de Start ou Stop bit. CK pode gerar oito ou nove pulsos, dependendo do estado de **TXSTA<TX9>**.

Ao término da transmissão, o sistema ficará em stand-by, aguardando um novo dado em **TXREG**.

Para o Slave, a diferença é que, depois de escrevermos em **TXREG**, o sistema ficará aguardando os clocks enviados pelo Master.

Recepção

A recepção para o Master e para o Slave também é idêntica, exceto pela geração do clock. Quando ativada uma operação de recepção, a operação de transmissão é desligada imediatamente e o pino **DT** torna-se entrada.

A recepção contínua manterá a unidade recebendo dados até que o bit **RCSTA<CREN>** seja limpo manualmente. Já a recepção unitária (**RCSTA<SREN>=1**) receberá somente um byte e depois desliga--se (**RCSTA<SREN>=0**). Se por acaso forem ligados as duas maneiras de recepção, o modo contínuo terá prioridade. No modo contínuo, o clock não é desativado, gerando pulsos constantes. Isso torna a operação nesse sistema mais difícil.

Ao receber um dado, o mesmo será enviado a **RCREG** e o flag **PIR1<RCIF>** será ativado, podendo gerar a interrupção.

Operação em SLEEP

Somente as unidades S/aves do modo síncrono podem operar em SLEEP.

Quando um dado é recebido durante o SLEEP, ao final da recepção a informação é transportada de **RSR** para **RCREG**, ativando o flag de interrupção **PIR1<RCXIF>**. [Deixando essa interrupção habilitada, o PIC será acordado e o dado recebido poderá ser tratado.]

Uma transmissão também é possível durante o SLEEP. O primeiro dado escrito em **TXREG** será imediatamente escrito em **TSR**. O segundo byte (se houver) será mantido em **TXREG**. O sistema é, então, colocado em SLEEP. Quando chegar o primeiro pacote de pulsos, o dado de **TSR** será transmitido. Ao final, o segundo dado será colocado em **TSR** e o flag **TXIF** será setado. Se a interrupção estiver habilitada, o PIC acordará.

Resumo dos registradores associados à USART Síncrona

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0Bh...	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
0Ch	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
8Ch	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
98h	TXSTA	CSRC	TX9	TXEN	SYNC	-	BRGH	TMRT	TX9D
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
19h	TXREG	Buffer de transmissão							
1AH	RCREG	Buffer de recepção							
99h	SPBRG	Acerto do baud Rate							

Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Lógica do exemplo

Neste exemplo implementaremos uma comunicação assíncrona Full Duplex, isto é, ativaremos tanto a transmissão quanto a recepção.

Para tornar nosso sistema versátil e simples, montaremos um programa capaz de operar somente com o hardware proposto, ou interligando esta ao PC por meio do conector DB9 (RS-232).

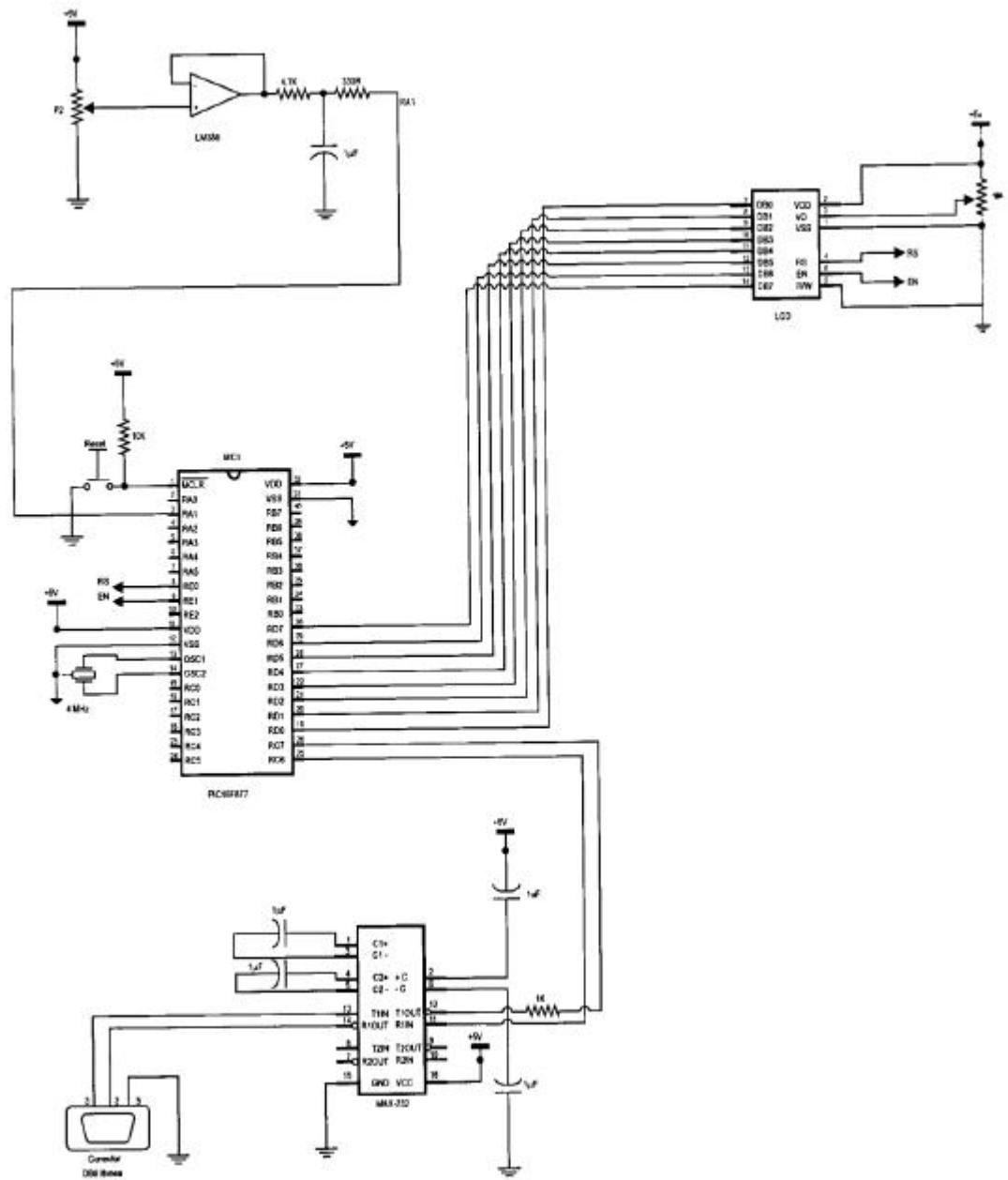
A transmissão será feita lendo-se a tensão do potenciômetro P2 através do A/D, limitando os valores entre 0 e 255 (8-bits) e enviando esse resultado para a porta serial e para o LCD. Desta forma, será possível visualizarmos o dado transmitido. Para facilitar ainda mais o usuário, mostraremos o valor em decimal (d) e em hexadecimal (h). A transmissão será realizada no padrão 8N1 com uma velocidade de 9.600bps.

Quanto à recepção, o valor obtido pela porta serial será diretamente impresso no display de LCD, através do código ASCII.

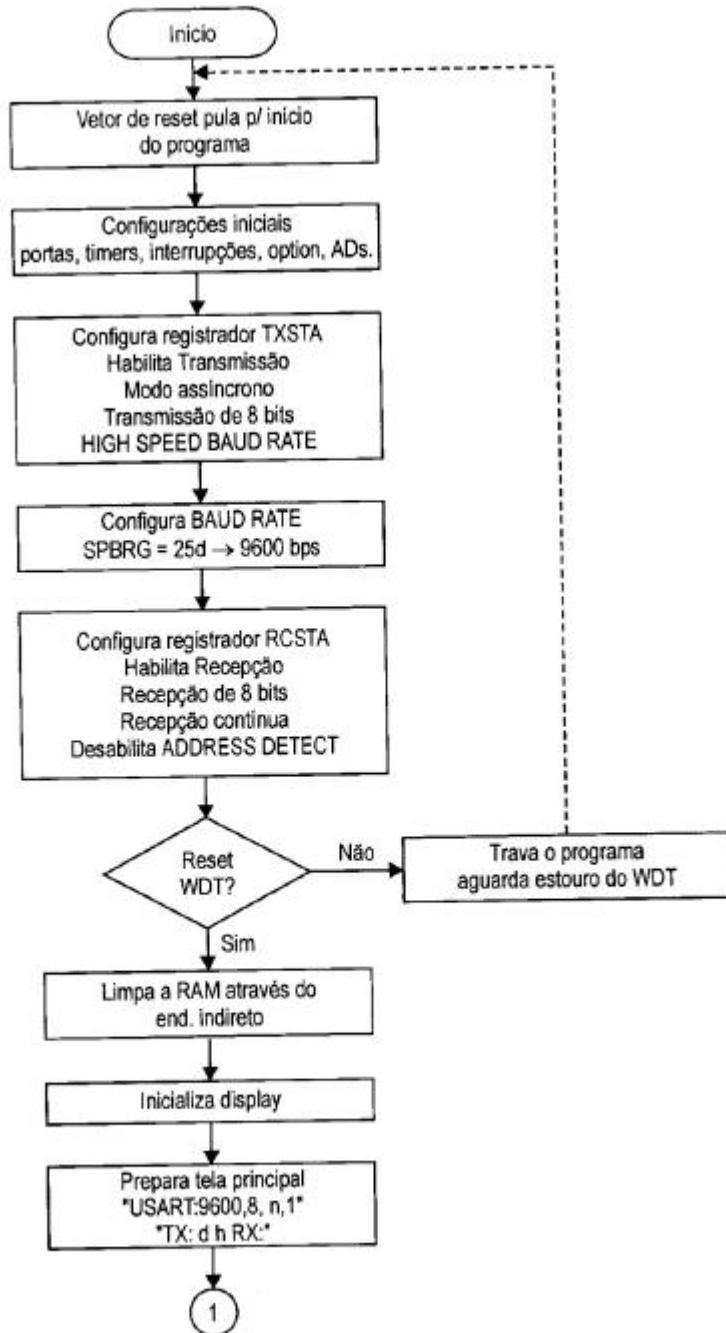
Para que o sistema funcione sem o PC, basta interligar os pinos 2 e 3 do conector DB9. Isso fará com que tudo que seja transmitido por TX seja imediatamente recebido em RX. Tanto a transmissão quanto a recepção são contínuas.

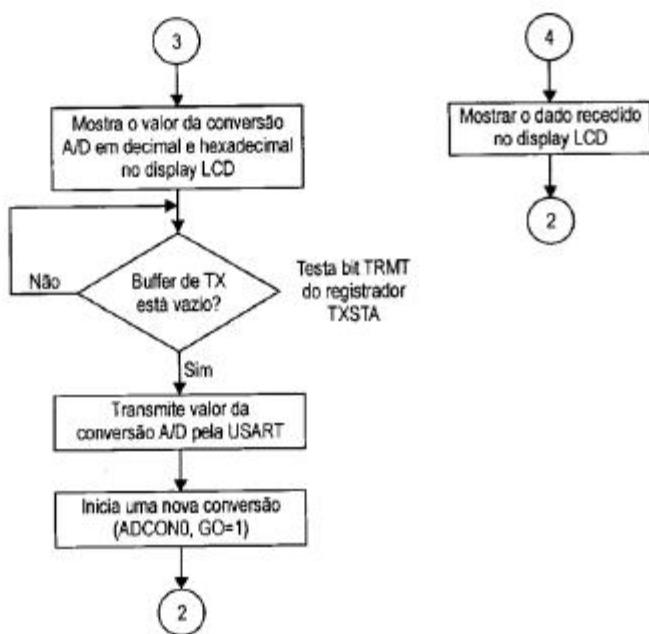
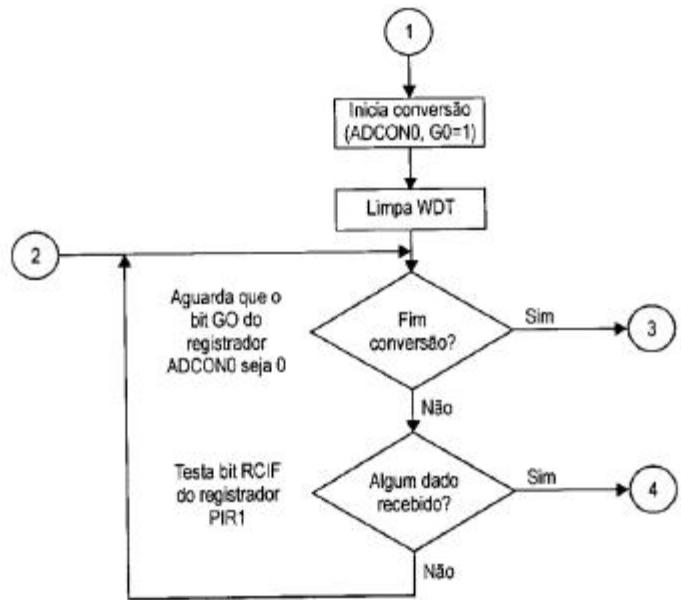
Como a comunicação está regulada para 9.600bps e a placa opera em 4 MHz, o exemplo de cálculo para já apresentado pode ser considerado.

Esquema Elétrico



Fluxograma





```

;*****CONNECTANDO O PIC - RECURSOS AVANÇADOS*****
;*          EXEMPLO 9
;*
;*          NICOLÁS CÉSAR LAVINIA e DAVID JOSÉ DE SOUZA
;*
;*****VERSAO : 2.0
;*          DATA : 24/02/2003
;*****DESCRICAO GERAL
;*
;*          ESTE EXEMPLO FOI ELABORADO PARA EXPLICAR O FUNCIONAMENTO DA USART DO PIC.
;*          O SOFTWARE CONVERTE O CANAL 1 DO CONVERSOR A/D (POTENCIÔMETRO P2) E MOSTRA
;*          NO DISPLAY O VALOR CONVERTIDO EM DECIMAL E HAXADECIMAL.
;*          ALÉM DE MOSTRAR O VALOR NO DISPLAY, O SOFTWARE TRANSMITE PELA USART O VALOR
;*          DA CONVERSÃO. OS VALORES RECEBIDOS PELA USART TAMBÉM SÃO MOSTRADOS NO LCD
;*          COMO CARACTERES ASCII.
;*
;*****CONFIGURAÇÕES PARA GRAVAÇÃO
;*
;_CONFIG_CPD_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &
;_PWRTE_ON & _WDT_ON & _XT_OSC
;*****DEFINIÇÃO DAS VARIÁVEIS
;*
;*          ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO LOGO NO INÍCIO DO BANCO 0
;*
;          CBLOCK 0X20
;*
;          POSIÇÃO INICIAL DA RAM
;*
;          TEMPO0
;          TEMPO1
;*
;          TMEPORIZADORES P/ ROTINA DE DELAY
;*
;          AUX
;          ; REGISTRADOR
;*
;          AUXILIAR DE USO GERAL
;*
;          UNIDADE
;*
;          ARMAZENA VALOR DA UNIDADE
;          DEZENA
;          ; ARMAZENA
;*
;          VALOR DA DEZENA
;          CENTENA
;*
;          ARMAZENA VALOR DA CENTENA
;          ENDCA
;*
;*****DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC
;*
;*          O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE
;*          OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE
;*          DE REDIGITAÇÃO.
;*
;#INCLUDE
;<P16F877A.INC>
;MICROCONTROLADOR UTILIZADO
;*
;*****DEFINIÇÃO DOS BANCOS DE RAM
;*
;*          OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR
;*          ENTRE OS BANCOS DE MEMÓRIA.
;*
#define BANK1      BSF

```

BANK1 DA MEMORIA RAM	STATUS,RP0 ; SELECIONA
#DEFINE	BANK0 BCF STATUS,RP0 ; SELECIONA
BANK0 DA MEMORIA RAM	
;***** ; * CONSTANTES INTERNAS * ;***** ; A DEFINIÇÃO DE CONSTANTES FACILITA A PROGRAMAÇÃO E A MANUTENÇÃO. ; ESTE PROGRAMA NÃO UTILIZA NENHUMA CONSTANTE.	
;***** ; * DECLARAÇÃO DOS FLAGs DE SOFTWARE * ;***** ; A DEFINIÇÃO DE FLAGS AJUDA NA PROGRAMAÇÃO E ECONOMIZA MEMÓRIA RAM. ; ESTE PROGRAMA NÃO UTILIZA NENHUM FLAG DE USUÁRIO	
;***** ; * ENTRADAS * ;***** ; AS ENTRADAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E ; FUTURAS ALTERAÇÕES DO HARDWARE. ; ESTE PROGRAMA UTILIZA UMA ENTRADA P/ O CONVERSOR A/D. ; ESTA ENTRADA NÃO PRECISA SER DECLARADA, POIS O SOFTWARE NUNCA FAZ ; REFERÊNCIA A ELA DE FORMA DIRETA, POIS O CANAL A/D A SER CONVERTIDO É ; SELECIONADO NO REGISTRADOS ADCON0 DE FORMA BINÁRIA E NÃO ATRAVÉS DE ; DEFINES. PORÉM PARA FACILITAR O ENTENDIMENTO DO HARDWARE VAMOS DECLARAR ; ESTA ENTRADA NORMALMENTE.	
#DEFINE	CAD_P2 PORTA,1 ; ENTRADA A/D
P/ O POTENCIÔMETRO P2	
; ALÉM DA ENTRADA DO CONVERSOR A/D, TEMOS A ENTRADA DA USART (RECEPÇÃO). ; NOVAMENTE ESTA ENTRADA NÃO NECESSITA SER DECLARADA, PORÉM, PARA ; FACILITAR O ENTENDIMENTO DO HARDWARE VAMOS DECLARAR ESTA ENTRADA ; NORMALMENTE.	
#DEFINE	RXUSART PORTC,7 ; ENTRADA DE
RX DA USART	
;***** ; * SAÍDAS * ;***** ; AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E ; FUTURAS ALTERAÇÕES DO HARDWARE.	
#DEFINE	DISPLAY PORTD ; BARRAMENTO
DE DADOS DO DISPLAY	
#DEFINE	RS PORTE,0 ; INDICA P/ O
DISPLAY UM DADO OU COMANDO	
; 1 -> DADO ; 0 -> COMANDO	
#DEFINE	ENABLE PORTE,1 ; SINAL DE
ENABLE P/ DISPLAY	
BORDA DE DESCIDA	; ATIVO NA

```

; TEMOS TAMBÉM A SAÍDA DE TX DA USART.
; NOVAMENTE ESTA SAÍDA NÃO NECESSITA SER DECLARADA, PORÉM, PARA FACILITAR O
; ENTENDIMENTO DO HARDWARE VAMOS DECLARAR ESTA SAÍDA NORMALMENTE.

#DEFINE TXUSART
                  PORTC,6
                  ; SAÍDA DE TX

DA USART

;***** VETOR DE RESET DO MICROCONTROLADOR *****
; POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA

ORG      0X0000
;

ENDEREÇO DO VETOR DE RESET
GOTO    CONFIG
; PULA PARA

CONFIG DEVIDO A REGIÃO
; DESTINADA AS

ROTINAS SEGUINTESS

;***** ROTINA DE DELAY (DE 1MS ATÉ 256MS) *****
; ESTA É UMA ROTINA DE DELAY VARIÁVEL, COM DURAÇÃO DE 1MS X O VALOR PASSADO
; EM WORK (W).

DELAY_MS

TEMPO1 (UNIDADES DE MS)
MOVWF  TEMPO1
; CARREGA

TEMPO0 (P/ CONTAR 1MS)
MOVLW .250
MOVWF  TEMPO0
; CARREGA

WDT (PERDE TEMPO)
CLRWDT
; LIMPA

TEMPO0 ?
DECFSZ TEMPO0,F
; FIM DE

GOTO    $-2
; NÃO -

VOLTA 2 INSTRUÇÕES
; SIM - PASSOU-

SE 1MS
DECFSZ TEMPO1,F
; FIM DE

TEMPO1 ?
GOTO    $-6
; NÃO -

VOLTA 6 INSTRUÇÕES
; SIM
RETURN
; 

RETORNA

;***** ROTINA DE ESCRITA DE UM CARACTER NO DISPLAY *****
; ESTA ROTINA ENVIAM CARACTER PARA O MÓDULO DE LCD. O CARACTER A SER
; ESCRITO DEVE SER COLOCADO EM WORK (W) ANTES DE CHAMAR A ROTINA.

ESCREVE
MOVWF

```

```

DISPLAY          ; ATUALIZA
; PORTD
DISPLAY (PORTD)           NOP ;
; PERDE 1US PARA ESTABILIZAÇÃO
; PULSO DE ENABLE AO DISPLAY
BSF              ; ENVIA UM
ENABLE
GOTO    $+1      ;,
BCF              ;,
ENABLE
;.
MOVlw   .1
CALL
DELAY_MS        ; DELAY DE 1MS
RETURN
; RETORNA
;*****
;*          AJUSTE DECIMAL          *
;*      W [HEX] = CENTENA [DEC] : DEZENA [DEC] ; UNIDADE [DEC]      *
;*****          ; ESTA ROTINA RECEBE UM ARGUMENTO PASSADO PELO WORK E RETORNA NAS VARIÁVEIS
; CENTENA, DEZENA E UNIDADE O NÚMERO BCD CORRESPONDENTE AO PARÂMETRO PASSADO.
AJUSTE_DECIMAL
MOVWF  AUX
; SALVA VALOR A CONVERTER EM AUX
CLRF
UNIDADE
CLRF
DEZENA
CLRF
CENTENA
; RESETA
REGISTRADORES
MOVF   AUX,F
BTFS C STATUS,Z
; VALORA
CONVERTER = 0 ?
RETURN
; SIM -
; RESETA
RETORNA
; NÃO
INCF
UNIDADE,F
; INCREMENTA
UNIDADE
MOVF
UNIDADE,W
XORLW 0X0A
BTFS S STATUS,Z
; UNIDADE = 10d ?
GOTO    $+3      ; NÃO
; SIM
CLRF
UNIDADE
; RESETA
UNIDADE
INCF
DEZENA,F
; INCREMENTA

```

DEZENA

```
MOVF  
DEZENA,W  
XORLW 0X0A  
BTFS  
STATUS,Z  
; DEZENA = 10d ?  
GOTO $+3  
; NÃO
```

```
; SIM  
CLRF  
DEZENA  
; RESETA
```

DEZENA

```
INCF  
CENTENA,F  
; INCREMENTA
```

CENTENA

```
DECFSZ AUX,F  
; FIM  
GOTO $-.14  
; NÃO -
```

VOLTA P/ CONTINUAR CONVERSÃO

```
RETURN  
; SIM
```

```
*****  
; *      CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE      *  
; *****  
; NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS  
; CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA A  
; MÁQUINA E AGUARDA O ESTOURO DO WDT.
```

CONFIG

```
CLRF    PORTA
```

GARANTE TODAS AS SAÍDAS EM ZERO

```
CLRF    PORTB  
CLRF    PORTC  
CLRF    PORTD  
CLRF    PORTE
```

BANK1

SELECCIONA BANCO 1 DA RAM

```
MOVLW  
B'11111111'  
MOVWF TRISA  
;
```

CONFIGURA I/O DO PORTA

```
MOVLW  
B'11111111'  
MOVWF TRISB  
;
```

CONFIGURA I/O DO PORTB

```
MOVLW  
B'10111111'  
MOVWF TRISC  
;
```

CONFIGURA I/O DO PORTC

```
MOVLW  
B'00000000'  
MOVWF TRISD  
;
```

CONFIGURA I/O DO PORTD

```
MOVLW  
B'00000100'  
MOVWF TRISE
```

```

; ; CONFIGURA I/O DO PORTE

        MOVLW
        B'11011011'
        MOVWF
        OPTION_REG
        ; CONFIGURA

OPTIONS

        ; PULL-UPs

DESABILITADOS

        ; INTER. NA

BORDA DE SUBIDA DO RB0

        ; TIMER0

INCREM. PELO CICLO DE MÁQUINA

        ; WDT - 1:8
        ; TIMER - 1:1

        MOVLW
        B'00000000'
        MOVWF
        INTCON
        ; CONFIGURA

INTERRUPÇÕES

        ; DESABILITA

TODAS AS INTERRUPÇÕES

        MOVLW
        B'00000100'
        MOVWF
        ADCON1
        ; CONFIGURA

CONVERSOR A/D

        ; RA0, RA1 E RA3

COMO ANALÓGICO

        ; RA2, RA4 E RA5

COMO I/O DIGITAL

        ; PORTE COMO

I/O DIGITAL

        ; JUSTIFICADO À

ESQUERDA

        ; 8 BITS EM

ADRESH E 2 BITS EM ADRESL

        ; Vref+ = VDD

(+5V)

        ; Vref- = GND (0V)

        MOVLW
        B'00100100'
        MOVWF TXSTA
        ;

CONFIGURA USART

        ; HABILITA TX

ASSINCRONO

        ; MODO

DE 8 BITS

        ; TRANSMISSÃO

        ; HIGH SPEED

```

```

BAUD RATE           MOVLW .25
                   MOVWF SPBRG
                   ;
ACERTA BAUD RATE -> 9600bps

BANK0              ; 

SELECCIONA BANCO 0 DA RAM      ; 
MOVLW
B'10010000'
MOVWF RCSTA
;
; CONFIGURA USART
; HABILITA RX
; RECEPÇÃO DE
8 BITS             ; RECEPÇÃO
CONTÍNUA          ; DESABILITA
ADDRESS DETECT    ; 
MOVLW
B'01001001'
MOVWF ADCON0
; CONFIGURA
CONVERSOR A/D     ; VELOCIDADE -
> Fosc/8          ; CANAL 1
; MÓDULO
LIGADO             ; 
; AS INSTRUÇÕES A SEGUIR FAZEM COM QUE O PROGRAMA TRAVE QUANDO HOUVER UM
; RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT. DESTA FORMA,
; SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA UM ESTOURO DE WDT
; E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO START-UP DO PIC.
BTFS C
STATUS,NOT_TO
; RESET POR
ESTOURO DE WATCHDOG TIMER ? GOTO   $ 
; NÃO -
AGUARDA ESTOURO DO WDT         ; SIM
;*****
;*          INICIALIZAÇÃO DA RAM      *
;*****
; ESTA ROTINA IRÁ LIMPAR TODA A RAM DO BANCO 0, INDO DE 0X20 A 0X7F
MOVLW 0X20
MOVWF FSR
;
APONTA O ENDEREÇAMENTO INDIRETO PARA      ; A PRIMEIRA
; POSIÇÃO DA RAM
LIMPA_RAM          CLRF   INDF
; LIMPA
A POSIÇÃO          INCF   FSR,F
;
INCREMENTA O PONTEIRO P/ A PRÓX. POS.      MOVF   FSR,W

```

```

XORLW 0X80
;

COMPARA O PONTEIRO COM A ÚLT. POS. +1
BTFSS
STATUS,Z
; JÁ LIMPOU

TODAS AS POSIÇÕES?
GOTO
LIMPA_RAM
; NÃO - LIMPA A

PRÓXIMA POSIÇÃO
; SIM

*****
; *      CONFIGURAÇÕES INICIAIS DO DISPLAY      *
*****
; ESTA ROTINA INICIALIZA O DISPLAY P/ COMUNICAÇÃO DE 8 VIAS, DISPLAY PARA 2
; LINHAS, CURSOR APAGADO E DESLOCAMENTO DO CURSOR À DIREITA.

INICIALIZACAO_DISPLAY
BCF      RS
;

SELECCIONA O DISPLAY P/ COMANDOS
MOVLW 0X30
;

ESCREVE COMANDO 0X30 PARA
CALL
ESCREVE
; INICIALIZAÇÃO
MOVLW .3
CALL
DELAY_MS
; DELAY DE 3MS

(EXIGIDO PELO DISPLAY)
MOVLW 0X30
;
;

ESCREVE COMANDO 0X30 PARA
CALL
ESCREVE
; INICIALIZAÇÃO
MOVLW 0X30
;
;

ESCREVE COMANDO 0X30 PARA
CALL
ESCREVE
; INICIALIZAÇÃO
MOVLW
B'00111000'
; ESCREVE

COMANDO PARA
CALL
ESCREVE
; INTERFACE DE

8 VIAS DE DADOS
MOVLW
B'00000001'
; ESCREVE

COMANDO PARA
CALL
ESCREVE
; LIMPAR TODO

O DISPLAY
MOVLW .1
CALL
DELAY_MS
; DELAY DE 1MS

MOVLW
B'00001100'

```

```

; ESCREVE
COMANDO PARA CALL
ESCREVE ; LIGAR O
DISPLAY SEM CURSOR MOVLW
B'00000110'
; ESCREVE
COMANDO PARA INCREM. CALL
ESCREVE ; AUTOMÁTICO À
DIREITA *****

; ROTINA DE ESCRITA DA TELA PRINCIPAL *
***** ; ESTA ROTINA ESCREVE A TELA PRINCIPAL DO PROGRAMA, COM AS FRASES:
; LINHA 1 - "USART:9600,8,n,1"
; LINHA 2 - "TX: d h RX:" MOVLW 0X80
;

COMANDO PARA POSICIONAR O CURSOR CALL
ESCREVE ; LINHA 0/
COLUNA 0 BSF RS
;

SELECONA O DISPLAY P/ DADOS ; COMANDOS
PARA ESCREVER AS ; LETRAS DE
"USART:9600,8,n,1" MOVLW 'U'
CALL
ESCREVE MOVLW 'S'
CALL
ESCREVE MOVLW 'A'
CALL
ESCREVE MOVLW 'R'
CALL
ESCREVE MOVLW 'T'
CALL
ESCREVE MOVLW ':'
CALL
ESCREVE MOVLW '9'
CALL
ESCREVE MOVLW '6'
CALL
ESCREVE MOVLW '0'
CALL
ESCREVE MOVLW '0'
CALL
ESCREVE MOVLW ''
CALL
ESCREVE MOVLW '8'
CALL
ESCREVE MOVLW ''

```

```

CALL
ESCREVE
MOVLW 'n'
CALL
ESCREVE
MOVLW ','
CALL
ESCREVE
MOVLW '1'
CALL
ESCREVE
BCF     RS
;
SELECCIONA O DISPLAY P/ COMANDO
MOVLW 0XC0
;
COMANDO PARA POSICIONAR O CURSOR
CALL
ESCREVE
; LINHA 1 /
COLUNA 0
BSF     RS
;
SELECCIONA O DISPLAY P/ DADOS
; COMANDOS
PARA ESCREVER AS
; LETRAS DE
"TX: d h RX: "
MOVLW 'T'
CALL
ESCREVE
MOVLW 'X'
CALL
ESCREVE
MOVLW ':'
CALL
ESCREVE
MOVLW ""
CALL
ESCREVE
MOVLW ""
CALL
ESCREVE
MOVLW ""
CALL
ESCREVE
MOVLW ""
CALL
ESCREVE
MOVLW "d"
CALL
ESCREVE
MOVLW ""
CALL
ESCREVE
MOVLW ""
CALL
ESCREVE
MOVLW ""
CALL
ESCREVE
MOVLW "h"
CALL
ESCREVE
MOVLW ""
CALL
ESCREVE
MOVLW 'R'
CALL
ESCREVE
MOVLW 'X'
CALL
ESCREVE
MOVLW ':'
CALL
ESCREVE

```

```

;*****
;*          LOOP PRINCIPAL      *
;*****  

; A ROTINA PRINCIPAL FICA AGUARDANDO O FINAL DA CONVERSÃO A/D E VERIFICANDO  

; SE ALGUM DADO FOI RECEBIDO PELA USART

CONVERSÃO A/D                                BSF  

                                                ADCON0,GO  

                                                ; INICIA  

                                                ; EXECUTADA

APENAS UMA VEZ  

LOOP                                         CLRWDT  

                                                ; LIMPA

WATCHDOG TIMER  

CONVERSÃO ?  

_GD  

; NÃO  

BTFS  

ADCON0,GO  

; FIM DA  

GOTO FIM_CONVERSAO  

; SIM  

; NÃO  

BTFS  

PIR1,RCIF  

; RECEBEU  

ALGUM DADO NA SERIAL ?  

GOTO DADO_RECEBIDO  

; SIM  

; NÃO  

GOTO    LOOP  

;  

VOLTA P/ LOOP

;*****
;*          MOSTRA A/D NO DISPLAY E TRANSMITE      *
;*****  

; ESTA ROTINA MOSTRA O VALOR DA CONVERSÃO A/D NO DISPLAY LCD TANTO EM DECIMAL  

; COMO EM HEXADECIMAL. O VALOR DA CONVERSÃO TAMBÉM É TRANSMITIDO PELA USART.  

; AO FINAL, A ROTINA REQUISITA UMA NOVA CONVERSÃO A/D.

FIM_CONVERSAO_AD  

; ***** MOSTRA VALOR DA CONVERSÃO A/D EM DECIMAL *****

WORK COM VALOR DO A/D                         MOVF  

                                                ADRESH,W  

                                                ; CARREGA  

CALL AJUSTE_DECIMA  

; CHAMA

L  

ROTINA DE AJUSTE DECIMAL  

BCF      RS  

;  

SELECCIONA O DISPLAY P/ COMANDO               MOVLW 0XC3  

;  

COMANDO PARA POSICIONAR O CURSOR              CALL  

                                                ESCREVE  

; LINHA 1 /  

COLUNA 3  

BSF      RS  

;  

SELECCIONA O DISPLAY P/ DADOS                MOVF

```

```

        CENTENA,W
        ADDLW 0X30
        ;
CONVERTE BCD DA CENTENA EM ASCII
        CALL
        ESCREVE
        ; ENVIA AO LCD

        MOVF
        DEZENA,W
        ADDLW 0X30
        ;
CONVERTE BCD DA DEZENA EM ASCII
        CALL
        ESCREVE
        ; ENVIA AO LCD

        MOVF
        UNIDADE,W
        ADDLW 0X30
        ;
CONVERTE BCD DA UNIDADE EM ASCII
        CALL
        ESCREVE
        ; ENVIA AO LCD

; ***** MOSTRA VALOR DA CONVERSÃO A/D EM HEXADECIMAL *****
        BCF      RS
        ;
SELECCIONA O DISPLAY P/ COMANDO
        MOVLW 0XC8
        ;
COMANDO PARA POSICIONAR O CURSOR
        CALL
        ESCREVE
        ; LINHA 1 /
COLUNA 8
        BSF      RS
        ;
SELECCIONA O DISPLAY P/ DADOS
        SWAPF
        ADRESH,W
        ; INVERTE NIBLE
DO ADRESH
        ANDLW
        B'00001111'
        ; MASCARA BITS
MAIS SIGNIFICATIVOS
        MOVWF AUX
        ;
SALVA EM AUXILIAR
        MOVLW 0X0A
        SUBWF AUX,W
        ; AUX -
10d (ATUALIZA FLAG DE CARRY)
        MOVLW 0X30
        ;
CARREGA WORK COM 30h
        BTFSC
        STATUS,C
        ; RESULTADO É
POSITIVO? (É UMA LETRA?)
        MOVLW 0X37
        ; SIM -
CARREGA WORK COM 37h
        ; NÃO - WORK
FICA COM 30h (NÚMERO)
        ADDWF AUX,W
        ; SOMA
O WORK AO AUXILIAR
        ; (CONVERSÃO
ASCII)

```

CARACTER AO DISPLAY LCD	CALL ESCREVE ; ENVIA
WORK COM ADRESH	MOVF ADRESH,W ; CARREGA
MAIS SIGNIFICATIVOS	ANDLW B'00001111' ; MASCARA BITS
SALVA EM AUXILIAR	MOVWF AUX ;
10d (ATUALIZA FLAG DE CARRY)	MOVLW 0X0A SUBWF AUX,W ; AUX -
CARREGA WORK COM 30h	MOVLW 0X30 ;
POSITIVO? (É UMA LETRA?)	BTFS C STATUS,C ; RESULTADO É
CARREGA WORK COM 37h	MOVLW 0X37 ; SIM -
FICA COM 30h (NÚMERO)	; NÃO - WORK
O WORK AO AUXILIAR	ADDWF AUX,W ; SOMA
ASCII)	; (CONVERSÃO
CARACTER AO DISPLAY LCD	CALL ESCREVE ; ENVIA
; ***** TRANSMITE VALOR DA CONVERSÃO A/D PELA USART *****	
WORK COM O VALOR DO A/D	MOVF ADRESH,W ; CARREGA
ALTERA P/ BANCO 1 DA RAM	BANK1 ;
TX ESTÁ VAZIO ?	BTFS S TXSTA,TRMT ; O BUFFER DE
AGUARDA ESVAZIAR	GOTO \$-1 ; NÃO -
VOLTA P/ BANCO 0 DA RAM	BANK0 ; SIM -
SALVA WORK EM TXREG (INICIA TX)	MOVWF TXREG ;
; ***** INICIA UMA NOVA CONVERSÃO *****	
NOVA CONVERSÃO A/D	BSF ADCON0,GO ; PEDE UMA
	GOTO LOOP ;

VOLTA PARA LOOP

```
;*****  
;*          ROTINA DE RECEPÇÃO DE DADOS NA USART      *  
;*****  
; ESTA ROTINA É EXECUTADA TODA VEZ QUE UM NOVO DADO É RECEBIDO PELA USART.  
; O DADO RECEBIDO É MOSTRADO NO LCD (EM ASCII).
```

DADO_RECEBIDO

BCF RS

;

SELECCIONA O DISPLAY P/ COMANDO

```
MOVlw 0XCF           ; COMANDO PARA POSICIONAR O CURSOR  
CALL   ESCREVE       ; LINHA 1 / COLUNA 15  
BSF    RS             ; SELECCIONA O DISPLAY P/ DADOS
```

```
MOVf   RCREG,W        ; CARREGA DADO RECEBIDO NO WORK  
CALL   ESCREVE       ; ENVIA AO LCD  
; AO LER O REGISTRADOR RCREG O BIT  
; RCIF DA INTERRUPÇÃO É LIMPO  
; AUTOMATICAMENTE.
```

GOTO LOOP ; VOLTA P/ LOOP PRINCIPAL

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

EN; FIM DO PROGRAMA

Dicas e comentários

A rotina de conversão Hex >>> Decimal deste exemplo é mais completa pois trabalha com três dígitos (CENTENA, DEZENA e UNIDADE). Desta forma, ela pode converter todo o range do argumento de entrada (W) que vai de 0 a 255.

O sistema de conversão A/D é o mesmo apresentado no capítulo 7, onde utilizamos o conversor interno e só consideramos os 8 bits mais significativos. Com isso nosso valor já fica limitado a um byte.

Devido à simplicidade do sistema, não foi necessário o uso das interrupções, deixando-as desabilitadas. Para o caso da recepção, o bit **RCIF** é testado toda vez dentro do loop principal. Quanto a transmissão, sempre que um novo valor foi convertido, checamos se o buffer de saída está vazio para podermos escrever o novo valor.

Exercícios propostos

Agora que o exemplo já foi estudado e esclarecido, aproveite para gerar novos problemas e soluções, seguindo os exercícios propostos:

1. Ative o uso da interrupção de recebimento. Quanto à transmissão, em vez de deixá-la contínua, crie uma interrupção de timer como base de tempo. Por exemplo, transmita o valor atual convertido a cada 1 segundo.
2. Crie um programa no PC (pode ser em Basic mesmo) que receba o valor convertido, efetue alguma operação e devolva outro valor. Por exemplo, divida o valor por 25, pegue a parte inteira e some 30h para imprimir no LCD um valor de 0 a 9.
3. Mude a rotina de recepção e escrita no LCD para poder receber um número de 0 a 50 e mostrá-lo como 0.0 a 5.0. Altere o programa do PC para evituar a regra de três necessária para converter um valor de 0 a 255 para 0 a 50. Com isso você voltou ao multímetro do Exemplo 4, só que com as contas de multiplicação e divisão não mais sendo feitas no PIC. Como você ainda tem os dados sendo enviados ao PC, é possível plotá-los como se fosse um minioscópio.

Outras Características

Introdução

Este capítulo é destinado à apresentação de alguns recursos ou sistemas que não foram abordados no decorrer dos capítulos anteriores. Estes assuntos serão divididos em tópicos para facilitar a didática e a compreensão.

Comunicação paralela (PSP)

O sistema de comunicação paralela em 8-bits do PIC 16F877A é muito simples, e por isso nem mereceu um capítulo dedicado a ele.

Sua primeira limitação é que ele só trabalha em modo S/ave, isto é, controlado por alguma outra unidade. Por isso, esse sistema recebe o nome de PSP (Parallel Slave Port).

O controle da ativação da PSP encontra-se em **TRISE<PSPPMODE>**:

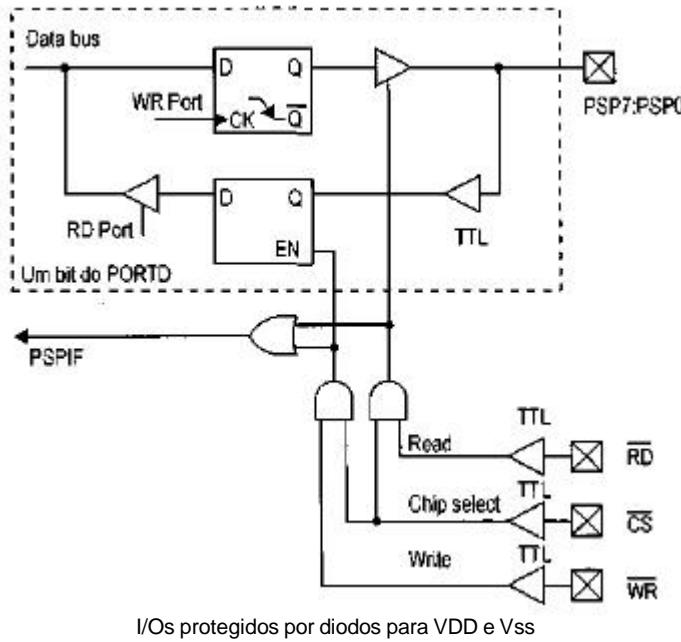
PSPPMODE	Descrição
0	PSP desativada. Pinos como I/Os convencionais.
1	PSP ativada. Pinos controlados internamente.

Os pinos utilizados pela porta paralela são:

Pino	Descrição
RD7... RD0	Barramento de dados (8-bits). Será lido/escrito diretamente nos latchets do PORTD.
RE0/RD	Efetua uma leitura (Read). Configurado como entrada.
RE1/WR	Efetua uma escrita (Write). Configurado como entrada.
RE2/CS	Ativa/Desativa a PSP externamente. Configurado como entrada.

Apesar do controle interno da porta paralela, os pinos **/RD**, **/WR** e **/CS** devem ser configurado; anteriormente com entrada (**TRISE**) e com funcionamento digital (desligar o A/D nesses pinos).

O **PORTD** possui dois Latches, um para a saída e outro para a entrada. Por causa disso, o estado do **TRISD** não é considerado. Na figura seguinte, pode-se visualizar o hardware interno do PIC para o **PORTD** quando configurado para a PSP.



O pino **/CS** funciona como um Chip Select, e a porta somente operará quando esse pino estiver em nível baixo (0). Quando ele está em nível alto (1), os pinos do **PORTD** ficam como entrada, o latch de entrada não é alterado e nenhuma interrupção é gerada.

/CS	Descrição
0	PSP ativada externamente.
1	PSP desativada externamente.

Para uma operação de escrita, o Mestre deverá manter **/CS=0** e **/WR=0**. Neste caso, o latch de entrada será atualizado para todo o barramento de dados (8-bits). O bit **PIR1<PSPIF>** é selado e a interrupção ativada (se as chaves estiverem corretamente ligadas). O bit **TRISE<IBF>** também = ativado, indicando que existe um dado recebido em **PORTD**. Este bit será limpo automaticamente quando **PORTD** for lido. Caso seja efetuada uma nova operação de escrita sem o sistema ter lido : **PORTD** anteriormente, temos ainda **IBF=1** e, então, um erro de overflow ocorrerá, sendo o mesmo indicado através do bit **TRISE<IBOV>=1**. Neste caso, o dado anterior terá sido perdido. O bit **IBOV** deve ser limpo manualmente pelo programa.

/WR	Descrição
0	Operação de escrita.
1	Situação normal.

IBF	Descrição
0	Nenhum dado novo escrito no latch de entrada
1	Um novo dado foi escrito no latch de entrada. Aconteceu uma operação de escrita.

IBOV	Descrição
0	Sem erro de overflow.
1	Houve um erro de overflow.

Uma operação de leitura será feita através da situação **/CS=0** e **/RD=0**. Todo o **PORTD** será transformado em saída e o valor previamente escrito no registrador **PORTD** (latch de saída) será imposto ao barramento. Aqui a interrupção também pode acontecer, pois o flag **PIR1<PSPIF>** é selado. Sempre que algum dado é escrito no registrador **PORTD** (latch de saída), teremos **TRISE<OBF>=1**. Após uma operação de leitura da porta paralela, esse bit é limpo automaticamente.

/RD	Descrição
0	Operação de leitura.
1	Situação normal.

OBF	Descrição
0	Aconteceu uma operação de leitura.
1	Aconteceu uma escrita interna no latch de saída.

Os pinos **/WR** e **/RD** precisam receber somente um pulso para ativar as operações de escrita e leitura. O tempo mínimo desse pulso é de ($1 / \text{Fosc}$), mas recomendamos a utilização de T_{CY} . O tempo máximo é indefinido e depende da lógica do sistema. Caso **/CS** não precise ser controlado pelo Master (só existe um Slave no barramento), então este pino pode ser diretamente aterrado.

Watchdog Timer (WDT)

O Watchdog Timer é uma espécie de temporizador que, de tempos em tempos, reseta o microcontrolador. É muito utilizado para evitar que o sistema trave, seja por algum tipo de falha ou porque o programa entrou em um loop infinito. Para que o PIC não seja resetado, o contador do WDT deve ser periodicamente zerado. Essa função é tão importante que existe uma instrução especialmente criada para isso (CLRWD).

O contador do WDT é independente do ciclo de máquina do PIC. Existe dentro da pastilha um circuito oscilador tipo RC que incrementa o contador do WDT, de forma que ele trabalha independentemente do oscilador do PIC. Isso significa que o WDT continua operando mesmo em modo SLEEP.

O período de estouro do WDT é de aproximadamente 18ms. Esse tempo pode ser aumentado, setando-se o bit **OPTION_REG<PSA>** e ajustando-se o valor do prescaler nos bits **OPTION_REG<PS2:PS0>**.

Os bits **STATUS</TO>** e **STATUS</PD>** serão selados sempre que a instrução CLRWD for executada. O bit **/TO** será zerado sempre que ocorrer o estouro do Watchdog Timer.

O ideal é que apenas uma instrução CLRWDT seja utilizada no software todo. Por isso, geralmente, essa instrução é encontrada apenas no loop principal do programa. Dependendo da aplicação, em algumas rotinas de delay também se faz necessário o uso da instrução CLRWDT, apesar disso não ser muito aconselhável.

Não existe controle por software para ligar ou desligar o contador do WDT. Ele somente pode ser habilitado no momento da gravação do microcontrolador. Quando ele está ligado, o consumo é maior, devido ao funcionamento do circuito RC.

Resumo dos registradores associados ao Watchdog Timer

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
03h...	STATUS	IRP	RP1	RP0	/TO	/PD	Z	DC	C
81h...	OPTION_REG	/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

—
Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Power-on Reset (POR)

Um pulso de Power-on Reset é gerado internamente no PIC sempre que a borda de subida do VDD for detectada, ou seja, somente na energização do microcontrolador. O POR é utilizado para garantir que o microcontrolador opere corretamente e não sofra problemas durante a energização. Esse pulso dispara o temporizador do Power-up Timer, caso este esteja habilitado.

Quando um POR ocorre, o bit **PCON</POR>** é limpo (zero).

Power-up Timer (PWRT)

O Power-up Timer é um temporizador utilizado para garantir que o microcontrolador somente comece a operar depois que a fonte de alimentação estiver estabilizada. O PWRT faz com que o PIC permaneça em estado de reset durante um intervalo de 72ms. Esse tempo é contabilizado após o pulso do POR e pode variar em função da pastilha e da temperatura.

O PWRT opera com um circuito oscilador RC interno dedicado que somente pode ser habilitado no momento da gravação do PIC. Deve obrigatoriamente estar habilitado caso se deseje utilizar o Brown-out Reset.

Oscillator Start-up Timer (OST)

O Oscilator Start-up Timer fornece um delay de 1.024 ciclos do cristal (ou do oscilador ligado ao pino **OSC1**) após o final do PWRT para iniciar a execução do software. É importante para garantir que o oscilador já esteja operando e estável quando o software começar a ser executado.

O Oscilator Start-up Timer é habilitado automaticamente quando o PIC estiver utilizando osciladores do tipo LP, XT ou HS e seu delay (1.024 ciclos do oscilador) será gerado sempre em três diferentes situações: na energização (POR), após um Brown-out Reset e após um wake-up ("acordar") do modo SLEEP.

Brown-out Reset (BOR)

O Brown-out Reset força um reset do PIC sempre que a tensão de alimentação cair abaixo de 4V (típico) durante um intervalo de tempo maior que 100µs (típico). Caso a queda de tensão seja durante um intervalo de tempo menor que 100µs, o reset não será gerado. A duração do reset é de 72ms (lembra do PWRT?) e, enquanto a tensão de alimentação não for superior aos 4V, esse temporizador não será inicializado.

Quando um BOR ocorre, o bit **PCON<BOR>** é limpo (zero).

Para o PIC 16F877A o valor do BOR é fixo e não pode ser alterado (alguns modelos permitem valores programáveis), além de somente poder ser habilitado na gravação do microcontrolador.

SLEEP (Power-down Mode)

O PIC entra em SLEEP (Power-down Mode) sempre que a instrução SLEEP for executada. Neste modo de operação, o oscilador é desligado, os pinos de I/Os mantêm o estado anterior à instrução SLEEP e o WDT é resetado embora continue operando.

Para minimizar o consumo, deve-se desligar o conversor A/D, desabilitar clocks externos, colocar todos os pinos em entrada e desligar os pull-ups do PORTB. Para casos extremos, deve-se levar em conta também o uso do Watchdog Timer (o circuito oscilador RC permanece consumindo), Power-up Timer (também utiliza um oscilador RC) e Brown-out Reset.

Para que o PIC volte a operar e "acorde" da situação de SLEEP existem três formas:

- por meio de um reset externo na entrada **/MCLR**;
- pelo estouro do WDT, se habilitado;
- ou pelo evento de alguma interrupção.

Um reset externo na entrada **/MCLR** causará o reset do PIC (PC=0x0000). Todas as outras situações são consideradas uma continuação do programa e causarão um wake-up na linha seguinte à instrução SLEEP. Os bits **STATUS<TO:/PD>** poderão ser testados para descobrir a causa do reset.

/TO	Descrição		
0	Indica que ocorreu um		
1	estouro do WDT	Indica que ocorreu um Power-up ou foram executadas as instruções CLRWDT ou SLEEP.	

/PD	Descrição
0	Indica que a instrução SLEEP foi executada.
1	Indica que ocorreu um Power-up ou foi executada a instrução CLRWDT.

Quando a instrução SLEEP for executada, a próxima instrução (PC+1) já estará sendo processada (lembre-se do conceito do Pipeline). Para que o PIC "acorde" por um evento de interrupção, a chave individual da interrupção deve estar ligada. Ao "acordar" nessa situação, existem duas possibilidades. Se o bit **GIE** estiver desabilitado, o programa continuará a ser executado no ponto seguinte à instrução SLEEP. Se o bit **GIE** estiver habilitado, a instrução seguinte à instrução SLEEP será processada e, em seguida, o PC será desviado para o endereço 0x0004 (vetor de interrupção). Por esse motivo é recomendado que logo após a instrução SLEEP seja adicionado um NOP.

Se a chave geral de interrupções (**GIE=0**) estiver desabilitada e caso nenhuma interrupção habilitada individualmente tenha ocorrido, ao ser executada a instrução SLEEP duas situações podem ocorrer:

- Se o evento da interrupção ocorrer antes da execução da instrução SLEEP, a instrução SLEEP será executada como um NOP. Entretanto, o WDT não será resetado, o bit **/TO** não será selado e o bit **/PD** não será limpo.
- Se o evento da interrupção ocorrer durante ou após a execução da instrução SLEEP, a instrução será executada normalmente, porém o microcontrolador imediatamente "acordará" do SLEEP. Neste caso, o WDT será resetado, o bit **/TO** será setado e o bit **/PD** será limpo.

Desta maneira, não adianta testar esses bits antes da execução da instrução SLEEP, pois os mesmos podem ser alterados enquanto a instrução está sendo executada. Para saber se a instrução foi corretamente executada, deve-se testar o bit **/PD**. Se ele estiver setado, a instrução foi executada como um NOP. Quanto ao WDT, recomenda-se executar um CLRWDT antes da instrução SLEEP.

Controle de Resets

Como pode ser observado, o PIC16F877A apresenta diversas formas de reset. A combinação de alguns bits/flags especiais podem nos informar precisamente qual dessas formas ocorreu. Vejamos a tabela seguinte:

/POR	/BOR	/TO	/PD	Causa
0	x	1	1	Ocorreu um <i>Power-on Reset</i> .
0	x	0	x	Não pode ocorrer. /TO é setado em POR.
0	x	x	0	Não pode ocorrer. /PD é setado em POR.
1	0	1	1	Ocorreu um <i>Brown-out Reset (BOR)</i> .
1	1	0	1	Ocorreu um WDT Reset.
1	1	0	0	Ocorreu um WDT <i>Wake-up</i> (acordou).
1	1	u	u	Ocorreu um MCRL Reset durante operação normal.
1	1	1	0	Ocorreu um MCRL Reset durante SLEEP ou acordou SLEEP por interrupção.

Resumo dos registradores associados aos Resets.

Endereço	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
03h	STATUS	IRP	RP1	RP0	/TO	/PD	Z	DC	C
8Eh	PCON	-	-	-	-	-	-	/POR	/BOR

■ Não usado para essa finalidade. Para obter mais informações, consulte apêndice A

Oscilador

O PIC 16F877A pode operar com quatro tipos diferentes de osciladores:

- LP - Cristal de baixa potência
- XT - Cristal / Ressonador
- HS - Cristal / Ressonador de alta freqüência
- RC - Oscilador RC

Junto ao cristal ou ressonador devem ser colocados dois capacitores cerâmicos para terra. Porém, alguns modelos de ressonadores já incorporam esses capacitores internamente. Como regra geral, podemos dizer que ressonadores de três pinos já possuem os capacitores.

O tipo de cristal deve ser configurado antes do PIC ser gravado. A tabela a seguir mostra o tipo de configuração do oscilador em função da freqüência de trabalho, além do valor sugerido para os capacitores.

Tipo de Oscilador	Freqüência do Cristal	Capacitor
LP	32 kHz	33pF
	200 kHz	15pF
	200 kHz	47 - 68pF
XT	1 MHz	15pF
	4MHz	15pF
	4 MHz	15pF
HS	8 MHz	15-33pF
	20 MHz	15-33pF

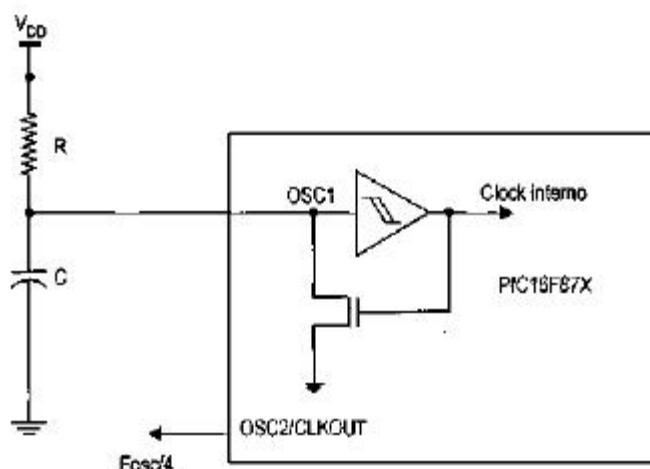
Para o caso do ressonador deve ser seguida a seguinte tabela:

'Tipo de Oscilador Freqüência do Ressonador Capacitor		
XT	455 kHz	68-100pF
	2 MHz	15-68pF
	4 MHz	15-68pF
HS	8 MHz	10-68pF
	16 MHz	10-22pF

Para aplicações onde a precisão não é critica, pode-se recorrer ao oscilador tipo RC.

Recomenda-se utilizar resistor entre 3k μ e 100K Ω . O capacitor deve ser maior do que 20pF.

A freqüência de trabalho do oscilador RC depende da tensão de alimentação e da temperatura ambiente. Por exemplo: com um resistor de 4,7K Ω , um capacitor de 100pF e uma temperatura de 25°C, o PIC deverá trabalhar próximo a 1 MHz. O data sheet apresenta gráficos auxiliares para cálculo do RC.



Sistema de proteção do código (Code Protection)

Depois que seu sistema estiver pronto e um protótipo e/ou produto for entregue a terceiros, é muito importante que essa opção seja ativada, pois ela impedirá que qualquer pessoa consiga ter acesso ao programa gravado no PIO. Para o PIC 16F877A, que é do tipo regravável (FLASH), o sistema de proteção impedirá a leitura, mas não a regravação. Entretanto, toda vez que uma nova gravação for feita quando o code protection estava ativado, primeiramente toda a memória de programa será limpa, evitando assim que qualquer tipo de alteração indevida seja efetuada no software. Esse sistema de proteção é interno ao PIC, não dependendo dos programas externos utilizados para a gravação/leitura. Isso garante a eficiência do mesmo.

Durante a configuração da gravação, existem diversas opções para proteção da memória de programa, considerando uma proteção total ou parcial.

Registradores de identificação (IDs)

O PIC possui quatro registradores especiais que se encontram nos endereços de 0x2000 a 0x2003 e não podem ser acessados diretamente pelo programa. Entretanto, esses registradores podem ser lidos e escritos por meio do sistema de gravação. Podemos utilizá-los, por exemplo, para a identificação da versão do programa gravado. A grande vantagem dos IDs é que podem ser lidos mesmo quando o Code Protection está ativado, possibilitando a rastreabilidade do software em casos de problemas futuros.

Entretanto, esses 4 bytes são mais limitados do que parecem, pois só devem ser utilizados os 4 bits menos significativos de cada um. Desta forma, apesar de 4 bytes, só possuímos 16 bits disponíveis.

O valor a ser gravado nos IDs pode ser escolhido diretamente durante as configurações de gravação. O sistema do MpLab possui incremento automático para a geração de número de série.

Sistema de emulação In-Circuit (Debugger Mode)

O PIC 16F877A, assim como muitos outros modelos FLASH mais recentes (isso é uma tendência da Microchip), possui um sistema interno para operar com debugadores de baixo custo. Quando essa função está habilitada (através das configurações de gravação), alguns recursos do PIC são perdidos:

Pinos de I/Os	RB6 e RB7 não operam mais como I/Os convencionais.
Pilha (Stack)	Perde-se 1 nível da pilha para o sistema de emulação.
Memória de programa	0 endereço 0000h deve possuir um NOP. Não podem ser utilizadas as últimas 100h words.
Memória de dados	Não podem ser utilizados os endereços 0x070 (0x0F0, 0x170 e 0x1F0) e de 0x1EB a 0x1EF.

Além dessas precauções, seu sistema deve ainda disponibilizar um sistema de interligação (conector) para um emulador externo. A própria Microchip possui uma família de emuladores para operar com o sistema In-Circuit Debugger Mode chamado MpLab ICD. A conexão entre o ICD e o PIC será feita por cinco vias: MCLFWPP, V_{DD}, V_{SS}, RB6 e RB7. Mais detalhes sobre esse sistema podem ser adquiridos através da documentação do MpLab ICD.

Proteção de escrita interna da FLASH

Esse sistema é conhecido como Flash Program Memory Write Enable e serve para proteger a E2PROM interna, impedindo eventuais escritas.

Gravação In-Circuit (ICSP)

O PIC 16F877A, assim como a maioria dos modelos da Microchip, possuem a possibilidade de ser gravado diretamente nos circuitos, isto é, já montados na placa final. Essa gravação é conhecida como in-circuit e pode possibilitar muito ganho de produtividade.

Como a gravação é feita de forma serial, utilizando-se somente cinco pinos, é possível a instalação de um soquete na placa para introduzir os sinais necessários à gravação. Obviamente uma série de cuidados especiais no projeto do hardware devem ser tomados, a fim de que as condições de gravação não prejudiquem os demais circuitos na placa.

Está completamente fora do objetivo deste livro entrar nos detalhes do protocolo de gravação (a Microchip possui um data sheet específico para isso), mas um exemplo de aplicação é a própria placa McLab2, que possui um conector para gravação in-circuit por meio dos gravadores McFlash ou McPlus (Mosaico).

Como pode ser observado no esquema elétrico dessa placa, os pinos necessários à gravação são:

- **V_{dd}**: Alimentação convencional do PIC (5V).
- **V_{ss}**: GND.
- **MCRL/V_{pp}**: Tensão de programação. Para que o PIC entre em modo de programação, a tensão nesse pino deve ser elevada a aproximadamente 13V. Por isso, o resto do circuito ligado a ele deve ser protegido.
- **RB6**: Clock da comunicação serial imposto pelo gravador.
- **RB7**: Dados da comunicação serial, que podem ser impostos pelo gravador (escrita) ou pelo próprio PIC (leitura).

Gravação em baixa tensão (Low Voltage Programming)

Além do sistema de gravação in-circuit descrito anteriormente, o PIC 16F877A possui um segundo modo de gravação. No sistema convencional pudemos observar que a gravação só pode ser efetuada por intermédio da presença de uma tensão especial em torno de 13V. Isso impossibilita a gravação em nível TTL. Com o sistema de gravação em baixa tensão (LVP), o PIC pode ser colocado em modo de programação com apenas 5V. Isso torna possível, por exemplo, que um PIC seja diretamente gravado por outro PIC.

O único problema é que, para poder utilizar o LVP, o pino RB3 é perdido como I/O convencional. Por isso, a habilitação desse sistema é feita por meio das configurações de gravação.

Anotações

Implementando um Sistema de Medição de Temperatura

Introdução

Parabéns, você chegou ao final do nosso estudo. Isso significa que a batalha foi ganha. Não deve ter sido muito fácil, mas esperamos que também não tenha sido tão difícil. O fato é que agora você possui muito mais conhecimento do que quando começou, e esse conhecimento será capaz de lançá-lo a um desafio muito maior, o desenvolvimento de projetos cada vez mais complexos. A batalha foi ganha, mas a luta está apenas começando.

Para ajudar nessa nova jornada, vamos apresentar a solução de um pequeno problema, utilizando os recursos da placa do Apêndice F (McLab2) que não foram vistos nos capítulos anteriores. A partir de agora, montaremos um pequeno sistema de medição de temperatura.

O sistema

O monitoramento da temperatura é um dos problemas mais clássicos enfrentado pela maioria dos projetistas. Como nossa placa possui um sensor de temperatura (diodo) e dois atuadores: aquecimento (resistência) e resfriamento (ventilador), nada melhor do que implementarmos um sistema capaz de obter a temperatura atual para mostrá-la no LCD.

O sensor de temperatura

Para podermos obter a temperatura ambiente, usaremos um diodo. Como o diodo é um componente que apresenta uma queda de tensão sobre ele proporcional à temperatura do mesmo, estaremos monitorando a tensão para encontrarmos a temperatura.

Para isso, nosso circuito eletrônico faz uso de um diodo de sinal convencional (1N4148) ligado a um amplificador e a uma porta analógica do PIC. Ligamos ao amplificador também um potenciômetro para podermos alterar o off-set da curva, ajustando assim a temperatura com uma referência externa.

Internamente, o sistema trabalhará com uma conversão A/D de 8 bits, gerando 256 possíveis valores de tensão para o diodo. Para cada valor obtido, teremos uma temperatura relacionada. A rotina TABELA_TEMP, que se encontra no final do código apresentado neste capítulo, efetuará a conversão entre a tensão lida (unidades de A/D) e a temperatura real. Nada mais é que uma tabela de conversão/linearização.

Essa tabela foi construída com base na curva de resposta do diodo utilizado em função da temperatura. Caso seja construído um sensor de temperatura com outro tipo de componente que gere uma tensão variável, basta refazer a tabela de conversão.

Uma vez convertida, a temperatura é, então, mostrada no LCD, na unidade de °C (lado direito).

O aquecimento

Possibilitaremos que o usuário aumente a temperatura sobre o diodo por meio do controle manual da resistência existente na placa. Faremos isso por intermédio de um dos PWMs do PIC, que se encontra ligado ao resistor.

Por Intermédio dos botões S1 e S2 poderemos aumentar e diminuir o duty cycle do PWM, variando de 0 a 100%. Mantendo-se os botões pressionados, o incremento/decremento será automático. O valor atual para o aquecimento será mostrado no LCD (lado esquerdo).

O resfriamento

Inversamente ao aquecimento, possilitaremos também o resfriamento do sistema por meio do ventilador, que é controlado pelo outro canal de PWM do PIC. Obviamente só seremos capazes de obter temperaturas levemente abaixo do valor ambiente, mas nossa intenção é podermos criar variáveis diferentes de aquecimento e resfriamento.

Controlaremos o ventilador pelo duty cycle do PWM, mas, para o sistema ficar um pouco mais completo, mostraremos no LCD (centro) o valor da rotação do motor, em RPS (rotações por segundo). Isso será feito através do sensor óptico que se encontra instalado na base das hélices do ventilador. Os botões S3 e S4 são usados para aumentar e diminuir o duty cycle do PWM, variando de 0 a 100%. Mantendo-se os botões pressionados, o incremento/decremento será automático.

Cada vez que uma das pás da hélice passa em frente ao sensor óptico, um pulso é transmitido ao PIC. Como esse sinal está ligado ao pino RC1, utilizamos o TMR1 com incremento externo para contabilizar a quantidade de pulsos gerados. A cada segundo (base de tempo gerada pela interrupção de TMR2), o total de pulsos é transferido para a variável CONT_VENT. Antes de mostrarmos o valor correto no LCD, devemos dividir o total de pulsos durante 1 segundo (CONT_VENT) pelo número de paletas (pulsos por volta). Neste caso CONT_VENT será dividido por sete.

Comunicação serial

O sistema possibilita ainda que maiores implementações sejam feitas com o tratamento externo da temperatura, pois os valores obtidos são enviados automaticamente pela porta serial, a cada segundo. Com isso, é possível criarmos um pequeno programa de computador capaz de plotar essa temperatura no decorrer do tempo. Os dados também podem ser armazenados para cálculos ou consultas futuras. Use a imaginação.

Os dados são transmitidos pela USART por meio do conector DB-9, respeitando-se o padrão RS-232 com 8N1 e baud rate de 9.600bps. A cada um segundo é transmitido um byte com o valor da temperatura já convertido para °C.

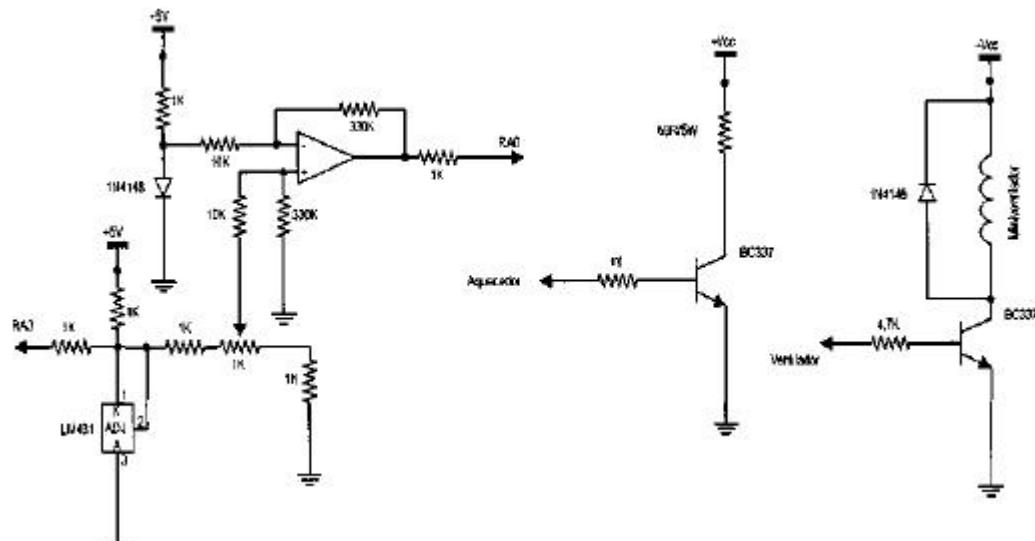
Considerações gerais

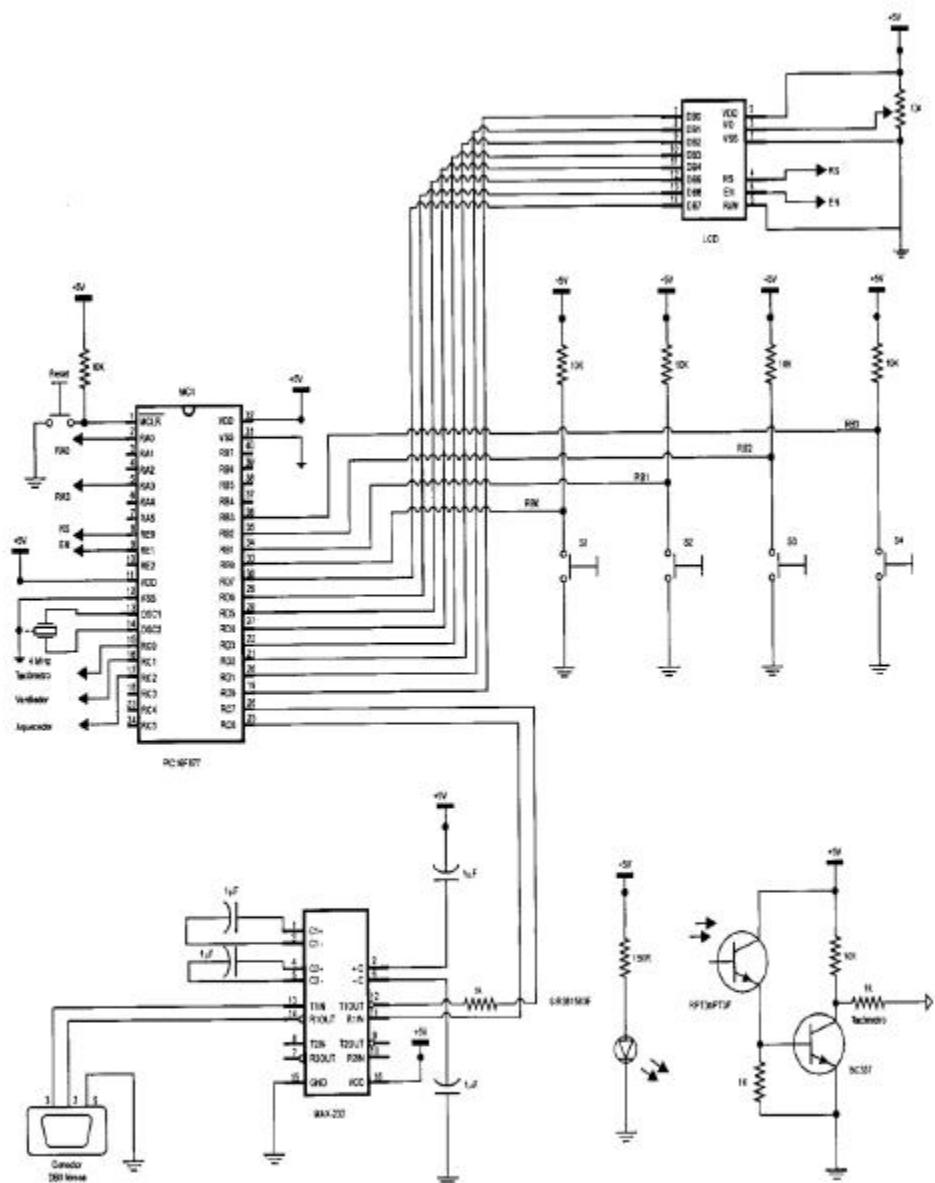
Ambos os PWMs são controlados pelo TMR2, que está regulado para 1 ms. Por isso, o período dos 2 PWMs é de 1ms, ou seja, eles operam a 1 kHz.

O postscale do TMR2 foi regulado em 1:10, gerando uma interrupção a cada 10ms. Utilizamos um contador auxiliar (TEMPO_1S) para contabilizarmos cem interrupções, gerando a base de tempo de um segundo. Essa base é utilizada para capturar a rotação do ventilador, efetuar uma conversão de temperatura e transmitir dados pela USART.

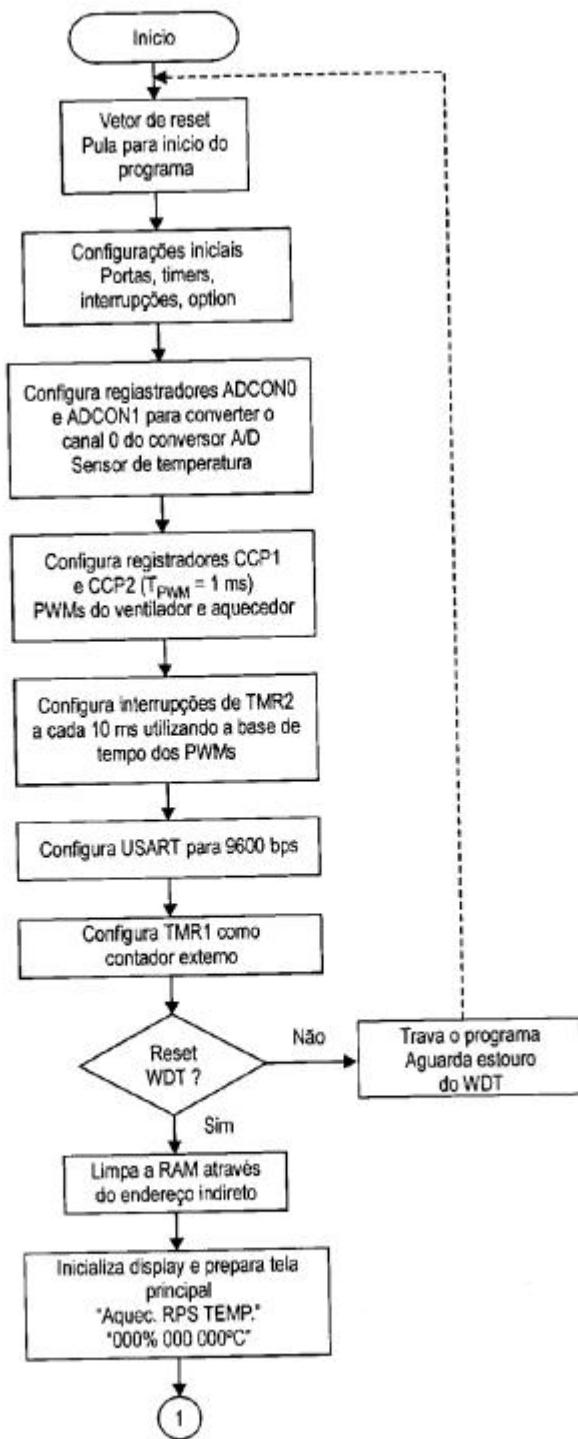
Esquema elétrico

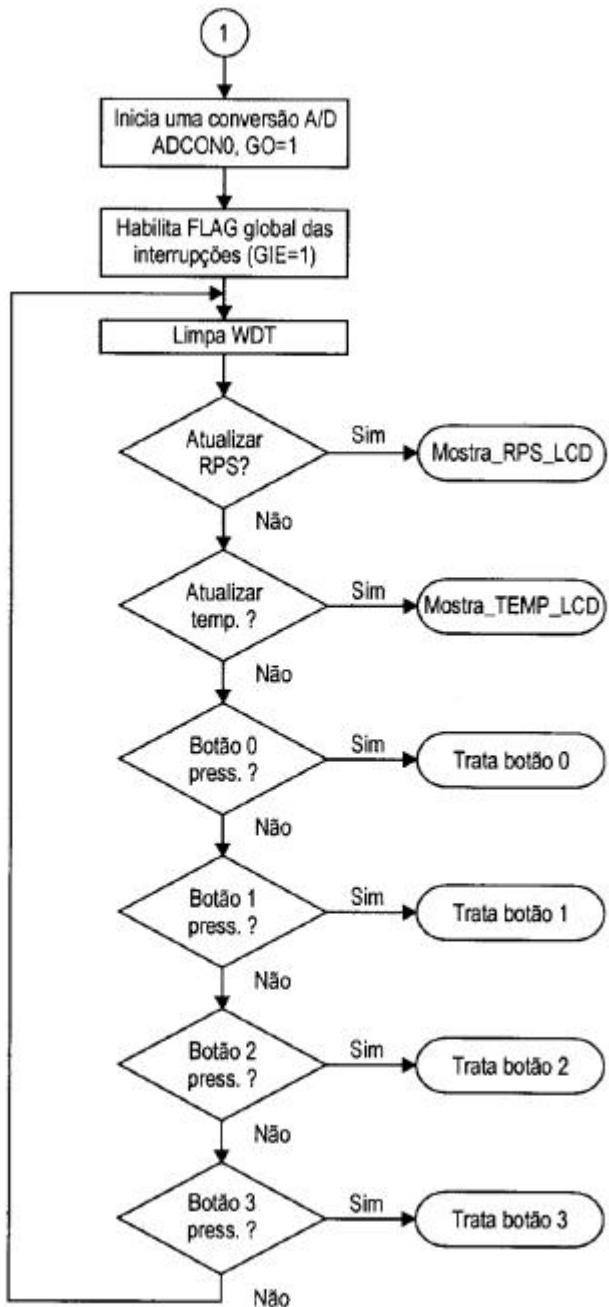
Esquema elétrico

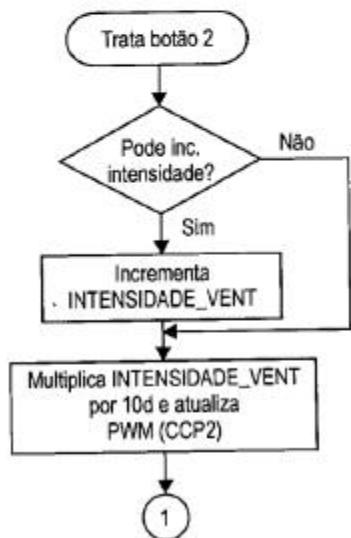
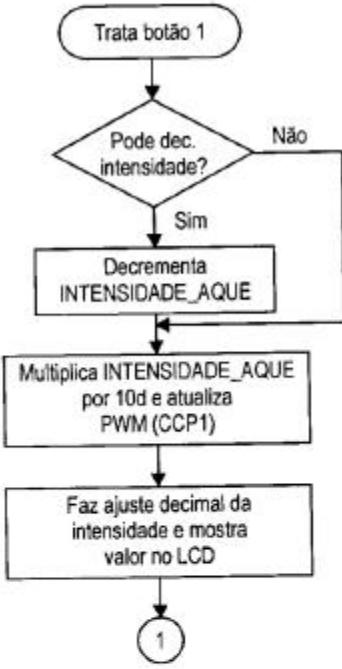
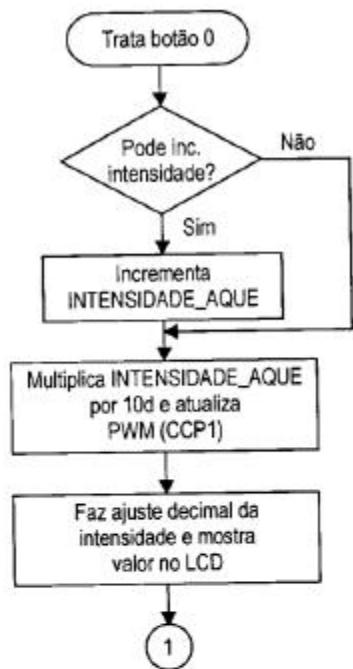


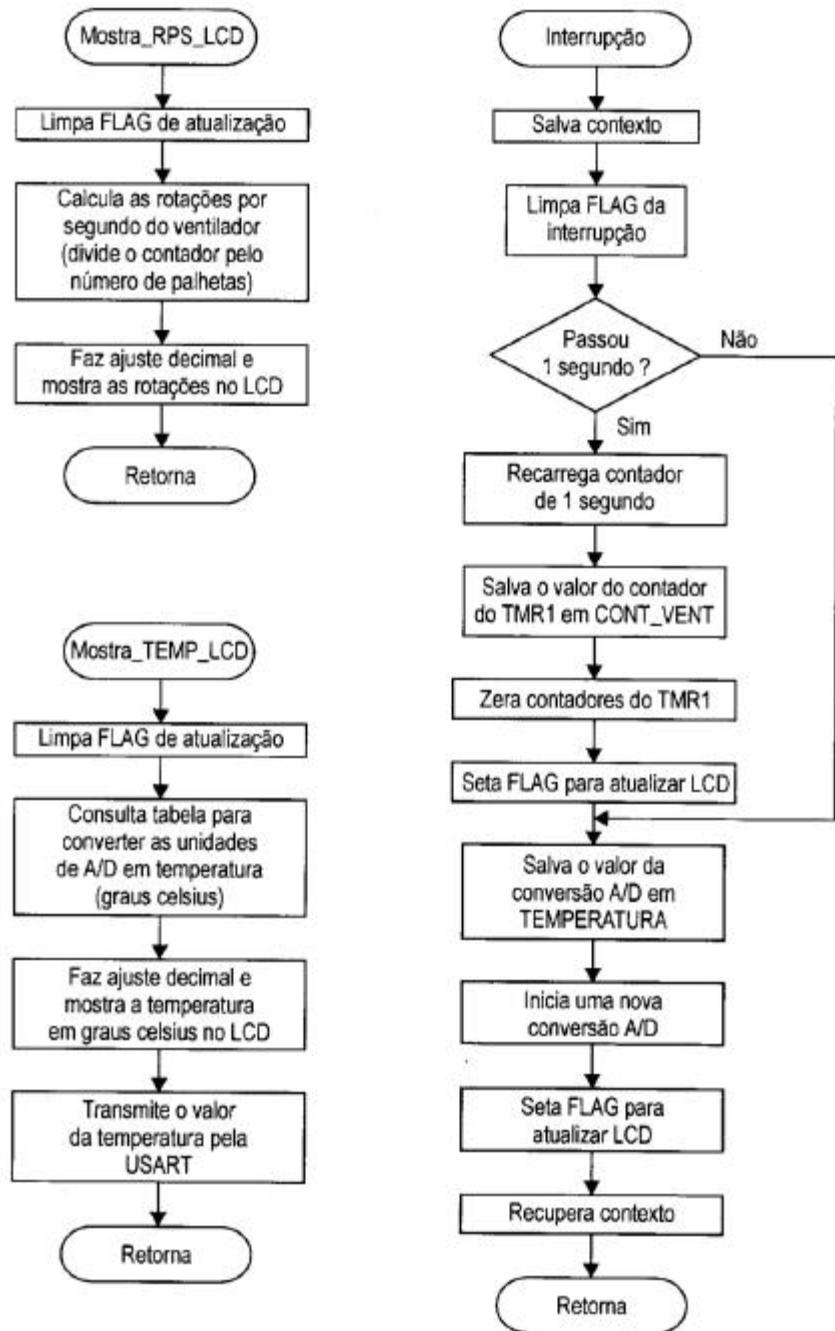


Fluxograma









```
;*****  
;*      CONECTANDO O PIC - RECURSOS AVANÇADOS      *  
;*  
;*      EXEMPLO 10          *  
;  
;*      NICOLÁS CÉSAR LAVINIA e DAVID JOSÉ DE SOUZA      *  
;  
;*  
;*****  
;* VERSÃO : 3.0          *  
;* DATA : 24/02/2003          *  
;*****
```

```
;*****  
;*      DESCRIÇÃO GERAL      *  
;  
; ESTE EXEMPLO FOI ELABORADO PARA EXPLICAR O FUNCIONAMENTO DO TMR1 COMO  
; CONTADOR, UTILIZADO NA PLACA MCLAB2 PARA CONTAR AS ROTAÇÕES DO VENTILADOR.  
; O SOFTWARE CONVERTE O CANAL 0 DO CONVERSOR A/D (SENSOR DE TEMPERATURA).  
; DOIS PWMs FORAM UTILIZADOS, UM PARA MODULAR A RESISTÊNCIA DE AQUECIMENTO  
; E OUTRO PARA A VELOCIDADE DO VENTILADOR.  
; COM AS TECLAS S1 E S2 PODE-SE VARIAR O PWM DO AQUECEDOR E COM AS TECLAS  
; S3 E S4 O PWM DO VENTILADOR.  
; NO LCD SÃO MOSTRADOS OS VALORES DO PWM DO AQUECEDOR, O NÚMERO DE ROTAÇÕES  
; POR SEGUNDO DO VENTILADOR E A TEMPERATURA DO DIODO JÁ CONVERTIDA EM GRAUS  
; CELSIUS. ALÉM DISSO, O VALOR ATUAL DA TEMPERATURA DO DIODO É TRANSMITIDO  
; PERIODICAMENTE ATRAVÉS DA USART.
```

```
;  
;*****  
;*      CONFIGURAÇÕES PARA GRAVAÇÃO      *  
;  
;*****
```

```
__CONFIG_CPD_OFF & _CPD_OFF & _DEBUG_OFF & _LVP_OFF & _WRT_OFF & _BODEN_OFF &  
_PWRTE_ON & _WDT_ON & _XT_OSC
```

```
;*****  
;*      DEFINIÇÃO DAS VARIÁVEIS      *  
;
```

;*****

; ESTE BLOCO DE VARIÁVEIS ESTÁ LOCALIZADO LOGO NO INÍCIO DO BANCO 0

CBLOCK 0X20	; POSIÇÃO INICIAL DA RAM
TEMPO0	
TEMPO1	; TMEPORIZADORES P/ ROTINA DE DELAY
AUX	; REGISTRADOR AUXILIAR DE USO GERAL
UNIDADE	; ARMAZENA VALOR DA UNIDADE
DEZENA	; ARMAZENA VALOR DA DEZENA
CENTENA	; ARMAZENA VALOR DA CENTENA
INTENSIDADE_VENT	; INTENSIDADE DO VENTILADOR
INTENSIDADE_AQUE	; INTENSIDADE DO AQUECEDOR
ACCaHI	; ACUMULADOR a DE 16 BITS UTILIZADO
ACCaLO	; NA ROTINA DE DIVISÃO
ACCbHI	; ACUMULADOR b DE 16 BITS UTILIZADO
ACCbLO	; NA ROTINA DE DIVISÃO
ACCcHI	; ACUMULADOR c DE 16 BITS UTILIZADO
ACCcLO	; NA ROTINA DE DIVISÃO
ACCdHI	; ACUMULADOR d DE 16 BITS UTILIZADO
ACCdLO	; NA ROTINA DE DIVISÃO
temp	; CONTADOR TEMPORÁRIO UTILIZADO
	; NA ROTINA DE DIVISÃO
H_byte	; ACUMULADOR DE 16 BITS UTILIZADO
L_byte	; P/ RETORNAR O VALOR DA ROTINA
	; DE MULTIPLICAÇÃO

Conectando o PIC 16F877A - Recursos Avançados

```

mulpr           ; OPERADOR P/ ROTINA DE MULTIPLICAÇÃO
mulcnd           ; OPERADOR P/ ROTINA DE MULTIPLICAÇÃO

TEMPERATURA      ; TEMPERATURA DO DIODO EM UNIDADES DE A/D
TEMP_CELSIUS      ; TEMPERATURA DO DIODO JÁ CONVERTIDO
                  ; PARA GRAUS CELSIUS

FILTRO_BOTOES    ; FILTRO P/ DEBOUNCE DOS BOTOES
TEMPO_TURBO       ; TEMPORIZADOR P/ TUBO DO TECLADO

TEMPO_1S          ; TEMPORIZADOR DE 1 SEGUNDO

CONT_VENT_HIGH
CONT_VENT_LOW      ; CONTADORES PARA ROTAÇÃO DO
VENTILADOR

FLAG              ; FLAG DE USO GERAL

WORK_TEMP
STATUS_TEMP
PCLATH_TEMP
FSR_TEMP          ; REGISTRADORES UTILIZADOS P/ SALVAR
                  ; O CONTEXTO DURANTE AS INTERRUPÇÕES

ENDC

```

```

;*****
;*      DEFINIÇÃO DAS VARIÁVEIS INTERNAS DO PIC      *
;*****                                                 *
; O ARQUIVO DE DEFINIÇÕES DO PIC UTILIZADO DEVE SER REFERENCIADO PARA QUE
; OS NOMES DEFINIDOS PELA MICROCHIP POSSAM SER UTILIZADOS, SEM A NECESSIDADE
; DE REDIGITAÇÃO.

```

```
#INCLUDE <P16F877A.INC>      ; MICROCONTROLADOR UTILIZADO
```

```
;*****  
;*          DEFINIÇÃO DOS BANCOS DE RAM      *  
;*****  
;  
; OS PSEUDOS-COMANDOS "BANK0" E "BANK1", AQUI DEFINIDOS, AJUDAM A COMUTAR  
; ENTRE OS BANCOS DE MEMÓRIA.
```

```
#DEFINE BANK1  BSF      STATUS,RP0      ; SELECCIONA BANK1 DA MEMORIA RAM  
#DEFINE BANK0  BCF      STATUS,RP0      ; SELECCIONA BANK0 DA MEMORIA RAM
```

```
;*****  
;*          CONSTANTES INTERNAS      *  
;*****  
;  
; A DEFINIÇÃO DE CONSTANTES FACILITA A PROGRAMAÇÃO E A MANUTENÇÃO.
```

```
FILTRO_TECLA  EQU      .200          ; FILTRO P/ EVITAR RUIDOS DOS BOTÕES  
  
TURBO_TECLA  EQU      .70           ; TEMPERIZADOR P/ TURBO DO TECLADO
```

```
;*****  
;*          DECLARAÇÃO DOS FLAGs DE SOFTWARE      *  
;*****  
;  
; A DEFINIÇÃO DE FLAGs AJUDA NA PROGRAMAÇÃO E ECONOMIZA MEMÓRIA RAM.
```

```
#DEFINE MOSTRA_RPS  FLAG,0        ; FLAG PARA MOSTRAR A ROTAÇÃO NO LCD  
                                ; 1 -> DEVE MOSTRAR A ROTAÇÃO  
                                ; 0 -> NAO DEVE MOSTRAR A ROTAÇÃO
```

```
#DEFINE MOSTRA_TEMP  FLAG,1        ; FLAG PARA MOSTRAR A TEMPERATURA NO LCD  
                                ; 1 -> DEVE MOSTRAR A TEMPERATURA  
                                ; 0 -> NAO DEVE MOSTRAR A TEMPERATURA
```

```
;*****  
;*          ENTRADAS      *  
;*****  
;  
; AS ENTRADAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E  
; Conectando o PIC 16F877A - Recursos Avançados
```

; FUTURAS ALTERAÇÕES DO HARDWARE.

```
#DEFINE BOTAO_0           PORTB,0      ; ESTADO DO BOTÃO 0
                                ; 1 -> LIBERADO
                                ; 0 -> PRESSIONADO
```

```
#DEFINE BOTAO_1           PORTB,1      ; ESTADO DO BOTÃO 1
                                ; 1 -> LIBERADO
                                ; 0 -> PRESSIONADO
```

```
#DEFINE BOTAO_2           PORTB,2      ; ESTADO DO BOTÃO 2
                                ; 1 -> LIBERADO
                                ; 0 -> PRESSIONADO
```

```
#DEFINE BOTAO_3           PORTB,3      ; ESTADO DO BOTÃO 3
                                ; 1 -> LIBERADO
                                ; 0 -> PRESSIONADO
```

; ESTE PROGRAMA UTILIZA UMA ENTRADA P/ O CONVERSOR A/D.

; ESTA ENTRADA NÃO PRECISA SER DECLARADA, POIS O SOFTWARE NUNCA FAZ
; REFERÊNCIA A ELA DE FORMA DIRETA, POIS O CANAL A/D A SER CONVERTIDO É
; SELECIONADO NO REGISTRADOS ADCON0 DE FORMA BINÁRIA E NÃO ATRAVÉS DE
; DEFINES. PORÉM PARA FACILITAR O ENTENDIMENTO DO HARDWARE VAMOS DECLARAR
; ESTA ENTRADA NORMALMENTE.

```
#DEFINE CAD_TEMP          PORTA,0      ; ENTRADA A/D PARA TEMPERATURA
```

; ALÉM DA ENTRADA DO CONVERSOR A/D, TEMOS A ENTRADA DA USART (RECEPÇÃO).
; NOVAMENTE ESTA ENTRADA NÃO NECESSITA SER DECLARADA, PORÉM, PARA
; FACILITAR O ENTENDIMENTO DO HARDWARE VAMOS DECLARAR ESTA ENTRADA
; NORMALMENTE.

```
#DEFINE RXUSART           PORTC,7      ; ENTRADA DE RX DA USART
```

;*****

```

;*          SAÍDAS          *
;*****;
; AS SAÍDAS DEVEM SER ASSOCIADAS A NOMES PARA FACILITAR A PROGRAMAÇÃO E
; FUTURAS ALTERAÇÕES DO HARDWARE.

#DEFINE DISPLAY      PORTD      ; BARRAMENTO DE DADOS DO DISPLAY

#DEFINE RS           PORTE,0    ; INDICA P/ O DISPLAY UM DADO OU COMANDO
                           ; 1 -> DADO
                           ; 0 -> COMANDO

#DEFINE ENABLE        PORTE,1    ; SINAL DE ENABLE P/ DISPLAY
                           ; ATIVO NA BORDA DE DESCIDA

; TEMOS TAMBÉM AS SAÍDAS DE TX DA USART, PWM1 E PWM2.

; PARA FACILITAR O ENTENDIMENTO DO HARDWARE VAMOS DECLARAR ESTAS SAÍDAS
; NORMALMENTE APESAR DE NÃO SEREM UTILIZADAS.

#DEFINE TXUSART       PORTC,6    ; SAÍDA DE TX DA USART

#DEFINE VENTILADOR     PORTC,1    ; SAÍDA P/ VENTILADOR

#DEFINE AQUECEDOR      PORTC,2    ; SAÍDA P/ AQUECEDOR

;*****;
;*          VETOR DE RESET DO MICROCONTROLADOR          *
;*****;

; POSIÇÃO INICIAL PARA EXECUÇÃO DO PROGRAMA

ORG 0X0000      ; ENDEREÇO DO VETOR DE RESET
GOTO CONFIG      ; PULA PARA CONFIG DEVIDO A REGIÃO
                  ; DESTINADA AS ROTINAS SEGUINTEIS

;*****;
;*          VETOR DE INTERRUPÇÃO          *
; Conectando o PIC 16F877A - Recursos Avançados

```

;*****
; POSIÇÃO DE DESVIO DO PROGRAMA QUANDO UMA INTERRUPÇÃO ACONTECE

ORG 0X0004 ; ENDEREÇO DO VETOR DE INTERRUPÇÃO

; É MUITO IMPORTANTE QUE OS REGISTRADORES PRIORITÁRIOS AO FUNCIONAMENTO DA
; MÁQUINA, E QUE PODEM SER ALTERADOS TANTO DENTRO QUANTO FORA DAS INTS SEJAM
; SALVOS EM REGISTRADORES TEMPORÁRIOS PARA PODEREM SER POSTERIORMENTE
; RECUPERADOS.

SALVA_CONTEXTO

MOVWF WORK_TEMP ; SALVA REGISTRADOR DE TRABALHO E
SWAPF STATUS,W ; DE STATUS DURANTE O TRATAMENTO
MOVWF STATUS_TEMP ; DA INTERRUPÇÃO.
MOVF FSR,W
MOVWF FSR_TEMP ; SALVA REGISTRADOR FSR
MOVF PCLATH,W
MOVWF PCLATH_TEMP ; SALVA REGISTRADOR PCLATH

CLRF PCLATH ; LIMPA REGISTRADOR PCLATH
; (SELECCIONA PÁGINA 0)
CLRF STATUS ; LIMPA REGISTRADOR STATUS
; (SELECCIONA BANCO 0)

;*****
;* TRATAMENTO DA INTERRUPÇÃO DE TIMER 2 *
;*****
; A INTERRUPÇÃO DE TMR2 É UTILIZADA PARA FORNECER A BASE DE TEMPO PARA AS
; MEDIDAS DAS ROTAÇÕES POR SEGUNDO DO VENTILADOR E DA TEMPERATURA DO DIODO.
; ALÉM DISSO, ELA SETA OS FLAGS PARA QUE ESTES SEJAM ATUALIZADOS NO LCD.
; O TMR2 ESTÁ CONFIGURADO PARA POSTSCALE DE 1:10 E PORTANTO A CADA 10ms A
; INTERRUPÇÃO É GERADA.
; O CONVERSOR A/D É LIDO A CADA INTERRUPÇÃO, OU SEJA, A CADA 10ms.
; A CADA 100 INTERRUPÇÕES, OU SEJA, A CADA 1 SEGUNDO, O VALOR DO CONTADOR DO
; TMR1 É SALVO NA VARIÁVEL CONT_VENT (HIGH E LOW), DESTA FORMA, O VALOR DE
Conectando o PIC 16F877A - Recursos Avançados

; CONT_VENT É O NÚMERO DE ROTAÇÕES DO VENTILADOR POR SEGUNDO. NA VERDADE ESTE
; VALOR ENCONTRA-SE MULTIPLICADO PELO NÚMERO DE PALHETAS DO VENTILADOR.

INT_TMR2

BCF PIR1,TMR2IF ; LIMPA FLAG DA INTERRUPÇÃO

DECFSZ TEMPO_1S,F ; FIM DO 1 SEGUNDO ?

GOTO INT_TMR2_2 ; NÃO - PULA P/ INT_TMR2_2

; SIM

MOVlw .100

MOVWF TEMPO_1S ; RECARREGA TEMPORIZADOR DE 1 SEGUNDO

BCF T1CON,TMR1ON ; PARALIZA CONTADOR DO TMR1

MOVF TMR1H,W

MOVWF CONT_VENT_HIGH

MOVF TMR1L,W

MOVWF CONT_VENT_LOW ; SALVA VALOR DO TMR1 EM CONT_VENT

CLRF TMR1H

CLRF TMR1L ; RESETA CONTADORES

BSF T1CON,TMR1ON ; LIBERA CONTADORES DO TMR1

BSF MOSTRA_RPS ; SETA FLAG P/ MOSTRAR VALOR

; DAS RPS DO VENTILADOR

INT_TMR2_2

MOVF ADRESH,W

MOVWF TEMPERATURA ; SALVA VALOR DA CONVERSÃO A/D

; NA VARIÁVEL TEMPERATURA

BSF ADCON0,GO ; INICIA UMA NOVA CONVERSÃO

BSF MOSTRA_TEMP ; SETA FLAG P/ ATUALIZAR VALOR

; DA TEMPERATURA NO LCD

```

;*****
;*          SAÍDA DA INTERRUPÇÃO      *
;*****  

;  

; ANTES DE SAIR DA INTERRUPÇÃO, O CONTEXTO SALVO NO INÍCIO DEVE SER  

; RECUPERADO PARA QUE O PROGRAMA NÃO SOFRA ALTERAÇÕES INDESEJADAS.

```

SAI_INT

```

    MOVF   PCLATH_TEMP,W
    MOVWF PCLATH           ; RECUPERA REG. PCLATH (PAGINAÇÃO)
    MOVF   FSR_TEMP,W
    MOVWF FSR              ; RECUPERA REG. FSR (END. INDIRETO)
    SWAPF STATUS_TEMP,W
    MOVWF STATUS            ; RECUPERA REG. STATUS
    SWAPF WORK_TEMP,F
    SWAPF WORK_TEMP,W      ; RECUPERA REG. WORK
    RETFIE                 ; RETORNA DA INTERRUPÇÃO (HABILITA GIE)

```

```

;*****
;
```

ROTINA DE DIVISÃO

```

;*****  

;  

;          Double Precision Division
;  

;*****  

;  

; Division : ACCb(16 bits) / ACCa(16 bits) -> ACCb(16 bits) with
;  

;           Remainder in ACCc (16 bits)
;  

;           (a) Load the Denominator in location ACCaHI & ACCaLO ( 16 bits )
;  

;           (b) Load the Numerator in location ACCbHI & ACCbLO ( 16 bits )
;  

;           (c) CALL D_divF
;  

;           (d) The 16 bit result is in location ACCbHI & ACCbLO
;  

;           (e) The 16 bit Remainder is in locations ACCcHI & ACCcLO
;  

;*****  

;
```

D_divF

MOVlw .16

Conectando o PIC 16F877A - Recursos Avançados

MOVWF temp ; CARREGA CONTADOR PARA DIVISÃO

MOVF ACCbHI,W

MOVWF ACCdHI

MOVF ACCbLO,W

MOVWF ACCdLO ; SALVA ACCb EM ACCd

CLRF ACCbHI

CLRF ACCbLO ; LIMPA ACCb

CLRF ACCcHI

CLRF ACCcLO ; LIMPA ACCc

DIV

BCF STATUS,C

RLF ACCdLO,F

RLF ACCdHI,F

RLF ACCcLO,F

RLF ACCcHI,F

MOVF ACCaHI,W

SUBWF ACCcHI,W ;check if a>c

BTFS S STATUS,Z

GOTO NOCHK

MOVF ACCaLO,W

SUBWF ACCcLO,W ;if msb equal then check lsb

NOCHK

BTFS S STATUS,C ;carry set if c>a

GOTO NOGO

MOVF ACCaLO,W ;c-a into c

SUBWF ACCcLO,F

BTFS S STATUS,C

DECF ACCcHI,F

MOVF ACCaHI,W

SUBWF ACCcHI,F

BSF S STATUS,C ;shift a 1 into b (result)

Conectando o PIC 16F877A - Recursos Avançados

NOGO

RLF ACCbLO,F

RLF ACCbHI,F

DECFSZ temp,F ; FIM DA DIVISÃO ?

GOTO DIV ; NÃO - VOLTA P/ DIV

; SIM

RETURN ; RETORNA

* ROTINA DE MULTIPLICAÇÃO *

; 8x8 Software Multiplier

; (Fast Version : Straight Line Code)

;

; The 16 bit result is stored in 2 bytes

; Before calling the subroutine " mpv ", the multiplier should

; be loaded in location " mulpr ", and the multiplicand in

; " mulcnd ". The 16 bit result is stored in locations

; H_byte & L_byte.

; Performance :

; Program Memory : 37 locations

; # of cycles : 38

; Scratch RAM : 0 locations

; Define a macro for adding & right shifting

mult MACRO bit ; Begin macro

```

        BTFSC  mulplr,bit
        ADDWF  H_byte,F
        RRF    H_byte,F
        RRF    L_byte,F

        ENDM           ; End of macro

; *****
; Begin Multiplier Routine
; *****

mpy_F

        CLRF  H_byte
        CLRF  L_byte
        MOVF  mulcnd,W      ; move the multiplicand to W reg.
        BCF   STATUS,C       ; Clear carry bit in the status Reg.

        mult  0
        mult  1
        mult  2
        mult  3
        mult  4
        mult  5
        mult  6
        mult  7

        RETURN          ; RETORNA

; *****
;*          ROTINA DE DELAY (DE 1MS ATÉ 256MS)          *
;*****


; ESTA É UMA ROTINA DE DELAY VARIÁVEL, COM DURAÇÃO DE 1MS X O VALOR PASSADO
; EM WORK (W).

DELAY_MS

```

```

MOVWF TEMPO1           ; CARREGA TEMPO1 (UNIDADES DE MS)
MOVLW .250
MOVWF TEMPO0           ; CARREGA TEMPO0 (P/ CONTAR 1MS)

CLRWDT                ; LIMPA WDT (PERDE TEMPO)
DECFSZ TEMPO0,F        ; FIM DE TEMPO0 ?
GOTO    $-2             ; NÃO - VOLTA 2 INSTRUÇÕES
                        ; SIM - PASSOU-SE 1MS
DECFSZ TEMPO1,F        ; FIM DE TEMPO1 ?
GOTO    $-6             ; NÃO - VOLTA 6 INSTRUÇÕES
                        ; SIM
RETURN                 ; RETORNA

```

```

;*****
;*          ROTINA DE ESCRITA DE UM CARACTER NO DISPLAY      *
;*****
; ESTA ROTINA ENVIA UM CARACTER PARA O MÓDULO DE LCD. O CARACTER A SER
; ESCRITO DEVE SER COLOCADO EM WORK (W) ANTES DE CHAMAR A ROTINA.

```

ESCREVE

```

MOVWF DISPLAY           ; ATUALIZA DISPLAY (PORTD)
NOP                   ; PERDE 1US PARA ESTABILIZAÇÃO
BSF      ENABLE         ; ENVIA UM PULSO DE ENABLE AO DISPLAY
GOTO    $+1             ;.
BCF      ENABLE         ;.

```

```

MOVLW .1
CALL     DELAY_MS       ; DELAY DE 1MS
RETURN               ; RETORNA

```

```

;*****
;*          AJUSTE DECIMAL          *
;*      W [HEX] = CENTENA [DEC] : DEZENA [DEC] ; UNIDADE [DEC]   *
;*****
; ESTA ROTINA RECEBE UM ARGUMENTO PASSADO PELO WORK E RETORNA NAS VARIÁVEIS
Conectando o PIC 16F877A - Recursos Avançados

```

; CENTENA, DEZENA E UNIDADE O NÚMERO BCD CORRESPONDENTE AO PARÂMETRO PASSADO.

AJUSTE_DECIMAL

```
MOVWF AUX ; SALVA VALOR A CONVERTER EM AUX

CLRF UNIDADE
CLRF DEZENA
CLRF CENTENA ; RESETA REGISTRADORES

MOVF AUX,F

BTFSZ STATUS,Z ; VALOR A CONVERTER = 0 ?
RETURN ; SIM - RETORNA
; NÃO

INCF UNIDADE,F ; INCREMENTA UNIDADE

MOVF UNIDADE,W
XORLW 0X0A
BTFSZ STATUS,Z ; UNIDADE = 10d ?
GOTO $+3 ; NÃO
; SIM

CLRF UNIDADE ; RESETA UNIDADE
INCF DEZENA,F ; INCREMENTA DEZENA

MOVF DEZENA,W
XORLW 0X0A
BTFSZ STATUS,Z ; DEZENA = 10d ?
GOTO $+3 ; NÃO
; SIM

CLRF DEZENA ; RESETA DEZENA
INCF CENTENA,F ; INCREMENTA CENTENA

DECFSZ AUX,F ; FIM DA CONVERSÃO ?
GOTO $-.14 ; NÃO - VOLTA P/ CONTINUAR CONVERSÃO
RETURN ; SIM
```

```

;*****
;*      ROTINA PARA MOSTRAR A ROTAÇÃO DO VENTILADOR NO LCD      *
;*****  

; ESTA ROTINA ATUALIZA O VALOR DAS ROTAÇÕES POR SEGUNDO DO VENTILADOR NO LCD.  
  

MOSTRA_RPS_LCD  

    BCF      MOSTRA_RPS          ; LIMPA FLAG DE ATUALIZAÇÃO DA RPS  

    BCF      RS                 ; SELECCIONA O DISPLAY P/ COMANDOS  

    MOVLW 0XC7                  ; COMANDO PARA POSICIONAR O CURSOR  

    CALL     ESCREVE           ; LINHA 1 / COLUNA 7  

    BSF      RS                 ; SELECCIONA O DISPLAY P/ DADOS  

    MOVF    CONT_VENT_HIGH,W  

    MOVWF ACCbHI  

    MOVF    CONT_VENT_LOW,W  

    MOVWF ACCbLO                ; CARREGA ACCb COM VALOR DO CONTADOR  

    CLRF    ACCaHI  

    MOVLW .7  

    MOVWF ACCaLO                ; CARREGA ACCa COM NÚMERO DE PALHETAS  

                                ; DO VENTILADOR  

    CALL     D_divF             ; CHAMA ROTINA DE DIVISÃO  

    MOVF    ACCbLO,W  

    CALL     AJUSTE_DECIMAL     ; FAZ O AJUSTE DECIMAL DO RESULTADO  

                                ; (ROTAÇÕES POR SEGUNDO)  

    MOVF    CENTENA,W  

    ADDLW 0X30                  ; CONVERTE CENTENA EM ASCII  

    CALL     ESCREVE            ; ESCREVE VALOR NO LCD  

    MOVF    DEZENA,W  

    ADDLW 0X30                  ; CONVERTE DEZENA EM ASCII  

    CALL     ESCREVE            ; ESCREVE VALOR NO LCD

```

Conectando o PIC 16F877A - Recursos Avançados

```
MOVF UNIDADE,W  
ADDLW 0X30 ; CONVERTE UNIDADE EM ASCII  
CALL ESCREVE ; ESCREVE VALOR NO LCD  
  
RETURN ; RETORNA
```

```
;* ROTINA PARA MOSTRAR A TEMPERATURA NO LCD *  
*****  
; ESTA ROTINA CONSULTA UMA TABELA P/ CONVERTER O VALOR DO CANAL A/D DO SENSOR  
; DE TEMPERATURA EM GRAUS CELSIUS, MOSTRA ESTE NO LCD E TRANSMITE PELA USART.
```

MOSTRA_TEMP_LCD

```
BCF MOSTRA_TEMP ; LIMPA FLAG DE ATUALIZAÇÃO DA TEMP.
```

```
CALL TABELA_TEMPERATURA ; CONVERTE A/D EM GRAUS CELSIUS  
MOVWF TEMP_CELSIUS ; SALVA VALOR EM TEMP_CELSIUS  
CALL AJUSTE_DECIMAL ; FAZ O AJUSTE DECIMAL
```

```
BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS  
MOVLW 0xcb ; COMANDO PARA POSICIONAR O CURSOR  
CALL ESCREVE ; LINHA 1 / COLUNA 11  
BSF RS ; SELECCIONA O DISPLAY P/ DADOS
```

```
MOVF CENTENA,W  
ADDLW 0X30 ; CONVERTE CENTENA EM ASCII  
CALL ESCREVE ; ESCREVE VALOR NO LCD
```

```
MOVF DEZENA,W  
ADDLW 0X30 ; CONVERTE DEZENA EM ASCII  
CALL ESCREVE ; ESCREVE VALOR NO LCD
```

```
MOVF UNIDADE,W  
ADDLW 0X30 ; CONVERTE UNIDADE EM ASCII  
Conectando o PIC 16F877A - Recursos Avançados
```

```

CALL    ESCREVE           ; ESCREVE VALOR NO LCD

MOVF   TEMP_CELSIUS,W      ; CARREGA EM WORK A TEMPERATURA
BANK1
BTFSS  TXSTA,TRMT         ; O BUFFER DE TX ESTÁ VAZIO ?
GOTO   $-1                 ; NÃO - AGUARDA ESVAZIAR
BANK0
MOVWF TXREG                ; CARREGA TXREG COM O VALOR DO WORK
; TRANSMITE A TEMPERATURA EM GRAUS
; CELSIUS PELA USART
RETURN                         ; RETORNA

```

```

;*****
;*          CONFIGURAÇÕES INICIAIS DE HARDWARE E SOFTWARE
;*
;*****
; NESTA ROTINA SÃO INICIALIZADAS AS PORTAS DE I/O DO MICROCONTROLADOR E AS
; CONFIGURAÇÕES DOS REGISTRADORES ESPECIAIS (SFR). A ROTINA INICIALIZA A
; MÁQUINA E AGUARDA O ESTOURO DO WDT.

```

CONFIG

```

CLRF   PORTA               ; GARANTE TODAS AS SAÍDAS EM ZERO
CLRF   PORTB
CLRF   PORTC
CLRF   PORTD
CLRF   PORTE

```

```
BANK1                   ; SELECCIONA BANCO 1 DA RAM
```

```
MOVLW B'11111111'
```

```
MOVWF TRISA              ; CONFIGURA I/O DO PORTA
```

```
MOVLW B'11111111'
```

```
MOVWF TRISB              ; CONFIGURA I/O DO PORTB
```

```
MOVLW B'10111001'
```

MOVWF TRISC ; CONFIGURA I/O DO PORTC

MOVLW B'00000000'

MOVWF TRISD ; CONFIGURA I/O DO PORTD

MOVLW B'00000100'

MOVWF TRISE ; CONFIGURA I/O DO PORTE

MOVLW B'11011011'

MOVWF OPTION_REG ; CONFIGURA OPTIONS
; PULL-UPS DESABILITADOS
; INTER. NA BORDA DE SUBIDA DO RB0
; TIMER0 INCREM. PELO CICLO DE MÁQUINA
; WDT - 1:8
; TIMER - 1:1

MOVLW B'01000000'

MOVWF INTCON ; CONFIGURA INTERRUPÇÕES
; HABILITA INTER. DE PERIFÉRICOS

MOVLW B'00000010' ; CONFIGURA INTER. DE PERIFÉRICOS

MOVWF PIE1 ; HABILITA A INTERRUPÇÃO DE TMR2

MOVLW B'00000100'

MOVWF ADCON1 ; CONFIGURA CONVERSOR A/D
; RA0, RA1 E RA3 COMO ANALÓGICO
; RA2, RA4 E RA5 COMO I/O DIGITAL
; PORTE COMO I/O DIGITAL
; JUSTIFICADO À ESQUERDA
; 8 BITS EM ADRESH E 2 BITS EM ADRESL
; Vref+ = VDD (+5V)
; Vref- = GND (0V)

MOVLW B'00100100'

Conectando o PIC 16F877A - Recursos Avançados

```
MOVWF TXSTA           ; CONFIGURA USART
                        ; HABILITA TX
                        ; MODO ASSINCRONO
                        ; TRANSMISSÃO DE 8 BITS
                        ; HIGH SPEED BAUD RATE
MOVlw .25
MOVWF SPBRG          ; ACERTA BAUD RATE -> 9600bps
```

```
MOVlw .249
MOVWF PR2            ; CONFIGURA PERÍODO DO PWM
                        ; T=((PR2)+1)*4*Tosc*TMR2 Prescale
                        ; T=((249)+1)*4*250ns*4
                        ; T=1,000ms -> 1.000Hz = 1KHz
```

```
BANK0                ; SELECCIONA BANCO 0 DA RAM
```

```
MOVlw B'10010000'
MOVWF RCSTA          ; CONFIGURA USART
                        ; HABILITA RX
                        ; RECEPÇÃO DE 8 BITS
                        ; RECEPÇÃO CONTÍNUA
                        ; DESABILITA ADDRESS DETECT
```

```
MOVlw B'01000001'
MOVWF ADCON0         ; CONFIGURA CONVERSOR A/D
                        ; VELOCIDADE -> Fosc/8
                        ; CANAL 0
                        ; MÓDULO LIGADO
```

```
CLRF    TMR1L
CLRF    TMR1H          ; ZERA CONTADOR DO TMR1
```

```
MOVlw B'00000011'
MOVWF T1CON          ; CONFIGURA TMR1
                        ; PRESCALE DE 1:1
Conectando o PIC 16F877A - Recursos Avançados
```

```

; TMR1 INCREM. PELO PINO T1CKI (RC0)
; NÃO SINCRONIZADO COM CLOCK INTERNO
; CONTADOR HABILITADO

MOVLW B'01001101'

MOVWF T2CON           ; CONFIGURA TMR2
; TMR2 HABILITADO
; POSTSCALE 1:10
; PRESCALE 1:4

CLRF    CCPR2L         ; ZERA PWM DO CCP2 (RC1 - VENTILADOR)

MOVLW B'00001111'

MOVWF CCP2CON          ; CONFIGURA CCP2 P/ PWM

CLRF    CCPR1L         ; ZERA PWM DO CCP1 (RC2 - AQUECEDOR)

MOVLW B'00001111'

MOVWF CCP1CON          ; CONFIGURA CCP1 P/ PWM

; AS INSTRUÇÕES A SEGUIR FAZEM COM QUE O PROGRAMA TRAVE QUANDO HOUVER UM
; RESET OU POWER-UP, MAS PASSE DIRETO SE O RESET FOR POR WDT. DESTA FORMA,
; SEMPRE QUE O PIC É LIGADO, O PROGRAMA TRAVA, AGUARDA UM ESTOURO DE WDT
; E COMEÇA NOVAMENTE. ISTO EVITA PROBLEMAS NO START-UP DO PIC.

BTFS  STATUS,NOT_TO      ; RESET POR ESTOURO DE WATCHDOG TIMER ?
GOTO   $                  ; NÃO - AGUARDA ESTOURO DO WDT
                           ; SIM

;*****
;*          INICIALIZAÇÃO DA RAM
;*****
; ESTA ROTINA IRÁ LIMPAR TODA A RAM DO BANCO 0, INDO DE 0X20 A 0X7F

```

MOVLW 0X20

Conectando o PIC 16F877A - Recursos Avançados

```

MOVWF FSR           ; APONTA O ENDEREÇAMENTO INDIRETO PARA
                     ; A PRIMEIRA POSIÇÃO DA RAM

LIMPA_RAM

    CLRF INDF          ; LIMPA A POSIÇÃO
    INCF FSR,F          ; INCREMENTA O PONTEIRO P/ A PRÓX. POS.
    MOVF FSR,W
    XORLW 0X80          ; COMPARA O PONTEIRO COM A ÚLT. POS. +1
    BTFSS STATUS,Z      ; JÁ LIMPOU TODAS AS POSIÇÕES?
    GOTO LIMPA_RAM     ; NÃO - LIMPA A PRÓXIMA POSIÇÃO
                     ; SIM

;*****
;*
;***** CONFIGURAÇÕES INICIAIS DO DISPLAY *
;*****

; ESTA ROTINA INICIALIZA O DISPLAY P/ COMUNICAÇÃO DE 8 VIAS, DISPLAY PARA 2
; LINHAS, CURSOR APAGADO E DESLOCAMENTO DO CURSOR À DIREITA.

INICIALIZACAO_DISPLAY

    BCF RS              ; SELECCIONA O DISPLAY P/ COMANDOS

    MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
    CALL ESCREVE        ; INICIALIZAÇÃO

    MOVLW .3
    CALL DELAY_MS       ; DELAY DE 3MS (EXIGIDO PELO DISPLAY)

    MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
    CALL ESCREVE        ; INICIALIZAÇÃO

    MOVLW 0X30          ; ESCREVE COMANDO 0X30 PARA
    CALL ESCREVE        ; INICIALIZAÇÃO

    MOVLW B'00111000'   ; ESCREVE COMANDO PARA
    CALL ESCREVE        ; INTERFACE DE 8 VIAS DE DADOS

```

```

        MOVLW B'00000001'          ; ESCREVE COMANDO PARA
        CALL    ESCREVE           ; LIMPAR TODO O DISPLAY

        MOVLW .1
        CALL    DELAY_MS          ; DELAY DE 1MS

        MOVLW B'00001100'          ; ESCREVE COMANDO PARA
        CALL    ESCREVE           ; LIGAR O DISPLAY SEM CURSOR

        MOVLW B'00000110'          ; ESCREVE COMANDO PARA INCREM.
        CALL    ESCREVE           ; AUTOMÁTICO À DIREITA

;*****
;*          ROTINA DE ESCRITA DA TELA PRINCIPAL          *
;*****

; ESTA ROTINA ESCREVE A TELA PRINCIPAL DO PROGRAMA, COM AS FRASES:
; LINHA 1 - "AQUEC. RPS TEMP."
; LINHA 2 - " 000% 000 000°C"

        MOVLW 0X80          ; COMANDO PARA POSICIONAR O CURSOR
        CALL    ESCREVE           ; LINHA 0 / COLUNA 0
        BSF    RS              ; SELECCIONA O DISPLAY P/ DADOS
                                ; COMANDOS PARA ESCREVER AS
                                ; LETRAS DE "AQUEC. RPS TEMP."
        MOVLW 'A'
        CALL    ESCREVE
        MOVLW 'Q'
        CALL    ESCREVE
        MOVLW 'U'
        CALL    ESCREVE
        MOVLW 'E'
        CALL    ESCREVE
        MOVLW 'C'
        CALL    ESCREVE
        MOVLW '.'


```

CALL ESCREVE
MOVLW ''
CALL ESCREVE
MOVLW 'R'
CALL ESCREVE
MOVLW 'P'
CALL ESCREVE
MOVLW 'S'
CALL ESCREVE
MOVLW ''
CALL ESCREVE
MOVLW 'T'
CALL ESCREVE
MOVLW 'E'
CALL ESCREVE
MOVLW 'M'
CALL ESCREVE
MOVLW 'P'
CALL ESCREVE
MOVLW ''
CALL ESCREVE

BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS
MOVLW 0XC1 ; COMANDO PARA POSICIONAR O CURSOR
CALL ESCREVE ; LINHA 1 / COLUNA 1
BSF RS ; SELECCIONA O DISPLAY P/ DADOS
; COMANDOS PARA ESCREVER AS
; LETRAS DE "000%"
MOVLW '0'
CALL ESCREVE
MOVLW '0'
CALL ESCREVE
MOVLW '0'
CALL ESCREVE
MOVLW '%'

CALL ESCREVE

```
BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS
MOVLW 0XC7 ; COMANDO PARA POSICIONAR O CURSOR
CALL ESCREVE ; LINHA 1 / COLUNA 7
BSF RS ; SELECCIONA O DISPLAY P/ DADOS
; COMANDOS PARA ESCREVER AS
; LETRAS DE "000"
MOVLW '0'
CALL ESCREVE
MOVLW '0'
CALL ESCREVE
MOVLW '0'
CALL ESCREVE

BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS
MOVLW 0XCB ; COMANDO PARA POSICIONAR O CURSOR
CALL ESCREVE ; LINHA 1 / COLUNA 7
BSF RS ; SELECCIONA O DISPLAY P/ DADOS
; COMANDOS PARA ESCREVER AS
; LETRAS DE "000°C"
MOVLW '0'
CALL ESCREVE
MOVLW '0'
CALL ESCREVE
MOVLW '0'
CALL ESCREVE
MOVLW 0XDF
CALL ESCREVE
MOVLW 'C'
CALL ESCREVE
```

;*****

* LOOP PRINCIPAL *
Conectando o PIC 16F877A - Recursos Avançados

;*****

; A ROTINA PRINCIPAL FICA AGUARDANDO O MOMENTO DE ESCREVER O VALOR DAS
; ROTAÇÕES DO VENTILADOR E A TEMPERATURA NO LCD ALÉM DE VARRER O TECLADO
; PARA MANIPULAR O VALOR DO PWM.

BSF ADCON0,GO ; INICIA CONVERSÃO A/D
; EXECUTADA APENAS UMA VEZ

BSF INTCON,GIE ; HABILITA FLAG GLOBAL DAS
; INTERRUPÇÕES

VARRE

CLRWDAT ; LIMPA WATCHDOG TIMER

BTFSC MOSTRA_RPS ; DEVE MOSTRAR RPS NO LCD ?

CALL MOSTRA_RPS_LCD ; SIM - CHAMA ROTINA P/ ATUALIZAR RPS
; NÃO

BTFSC MOSTRA_TEMP ; DEVE MOSTRAR A TEMP. NO LCD ?

CALL MOSTRA_TEMP_LCD ; SIM - CHAMA ROTINA P/ ATUALIZAR TEMP.
; NÃO

BTFSS BOTAO_0 ; O BOTÃO 0 ESTÁ PRESSIONADO ?

GOTO TRATA_BOTAO_0 ; SIM - PULA P/ TRATA_BOTAO_0
; NÃO

BTFSS BOTAO_1 ; O BOTÃO 1 ESTÁ PRESSIONADO ?

GOTO TRATA_BOTAO_1 ; SIM - PULA P/ TRATA_BOTAO_1
; NÃO

BTFSS BOTAO_2 ; O BOTÃO 2 ESTÁ PRESSIONADO ?

GOTO TRATA_BOTAO_2 ; SIM - PULA P/ TRATA_BOTAO_2
; NÃO

BTFSS BOTAO_3 ; O BOTÃO 3 ESTÁ PRESSIONADO ?

GOTO TRATA_BOTAO_3 ; SIM - PULA P/ TRATA_BOTAO_3
; NÃO

MOVlw FILTRO_TECLA ; CARREGA NO WORK O VALOR DE FILTRO_TECLA

MOVwf FILTRO_BOTOES ; SALVA EM FILTRO_BOTOES
Conectando o PIC 16F877A - Recursos Avançados

```

; RECARREGA FILTRO P/ EVITAR RUIDOS

MOVlw .1

MOVwf TEMPO_TURBO ; CARREGA TEMPO DO TURBO DAS TECLAS
; COM 1 - IGNORA O TURBO A PRIMEIRA
; VEZ QUE A TECLA É PRESSIONADA

GOTO VARRE ; VOLTA PARA VARRER TECLADO

;*****
;* TRATAMENTO DOS BOTÕES *
;*****

;***** TRATAMENTO DO BOTÃO 0 *****

TRATA_BOTAO_0

DECFSZ FILTRO_BOTOES,F ; FIM DO FILTRO ? (RUIDO?)

GOTO VARRE ; NÃO - VOLTA P/ VARRE
; SIM - BOTÃO PRESSIONADO

DECFSZ TEMPO_TURBO,F ; FIM DO TEMPO DE TURBO ?

GOTO VARRE ; NÃO - VOLTA P/ VARRE
; SIM

MOVlw TURBO_TECLA

MOVwf TEMPO_TURBO ; RECARREGA TEMPORIZADOR DO TURBO
; DAS TECLAS

MOVlw .100

XORwf INTENSIDADE_AQUE,W

BTFSZ STATUS,Z ; PODE INCREMENTAR PWM DO AQUECEDOR ?

INCF INTENSIDADE_AQUE,F ; SIM - INCREMENTA INTENSIDADE_AQUE
; NÃO

MOVF INTENSIDADE_AQUE,W ; CARREGA INTENSIDADE_AQUE NO WORK

MOVWF mulpr ; CARREGA WORK EM mulpr

```

MOVlw .10

MOVWF mulcnd ; CARREGA 10d EM mulcnd

CALL mpy_F ; CHAMA ROTINA DE MULTIPLICAÇÃO

SWAPF L_byte,W

ANDLW B'00110000'

IORLW B'00001111'

RRF H_byte,F

RRF L_byte,F

RRF H_byte,F

MOVWF CCP1CON

RRF L_byte,W

MOVWF CCPR1L ; ATUALIZA REGISTRADORES DO DUTY CYCLE
; DO MÓDULO CCP1 - PWM DO AQUECEDOR

MOVF INTENSIDADE_AQUE,W ; FAZ O AJUSTE DECIMAL DA INTENSIDADE

CALL AJUSTE_DECIMAL ; DO PWM DO AQUECEDOR

BCF RS ; SELECCIONA O DISPLAY P/ COMANDOS

MOVlw 0XC1 ; COMANDO PARA POSICIONAR O CURSOR

CALL ESCREVE ; LINHA 1 / COLUNA 1

BSF RS ; SELECCIONA O DISPLAY P/ DADOS

MOVF CENTENA,W

ADDLW 0X30 ; FAZ AJUSTE ASCII DA CENTENA

CALL ESCREVE ; ESCREVE VALOR NO LCD

MOVF DEZENA,W

ADDLW 0X30 ; FAZ AJUSTE ASCII DA DEZENA

CALL ESCREVE ; ESCREVE VALOR NO LCD

MOVF UNIDADE,W

ADDLW 0X30 ; FAZ AJUSTE ASCII DA UNIDADE

CALL ESCREVE ; ESCREVE VALOR NO LCD

Conectando o PIC 16F877A - Recursos Avançados

GOTO VARRE ; VOLTA P/ VARREDURA DOS BOTÕES

; ***** TRATAMENTO DO BOTÃO 1 *****

TRATA_BOTAO_1

DECFSZ FILTRO_BOTOES,F ; FIM DO FILTRO ? (RUIDO?)

GOTO VARRE ; NÃO - VOLTA P/ VARRE

; SIM - BOTÃO PRESSIONADO

DECFSZ TEMPO_TURBO,F ; FIM DO TEMPO DE TURBO ?

GOTO VARRE ; NÃO - VOLTA P/ VARRE

; SIM

MOVLW TURBO_TECLA

MOVWF TEMPO_TURBO ; RECARREGA TEMPORIZADOR DO TURBO

; DAS TECLAS

MOVF INTENSIDADE_AQUE,F

BTFSZ STATUS,Z ; PODE DECREMENTAR PWM DO AQUECEDOR ?

DECF INTENSIDADE_AQUE,F ; SIM - DECREMENTA INTENSIDADE_AQUE

; NÃO

MOVF INTENSIDADE_AQUE,W ; CARREGA INTENSIDADE_AQUE NO WORK

MOVWF mulplr ; CARREGA WORK EM mulplr

MOVLW .10

MOVWF mulcnd ; CARREGA 10d EM mulcnd

CALL mpy_F ; CHAMA ROTINA DE MULTIPLICAÇÃO

SWAPF L_byte,W

ANDLW B'00110000'

IORLW B'00001111'

RRF H_byte,F

RRF L_byte,F

Conectando o PIC 16F877A - Recursos Avançados

```

        RRF      H_byte,F
        MOVWF CCP1CON
        RRF      L_byte,W
        MOVWF CCP1R1L           ; ATUALIZA REGISTRADORES DO DUTY CYCLE
                                ; DO MÓDULO CCP1 - PWM DO AQUECEDOR

        MOVF    INTENSIDADE_AQUE,W   ; FAZ O AJUSTE DECIMAL DA INTENSIDADE
        CALL    AJUSTE_DECIMAL      ; DO PWM DO AQUECEDOR

        BCF    RS                  ; SELECCIONA O DISPLAY P/ COMANDOS
        MOVLW 0XC1                ; COMANDO PARA POSICIONAR O CURSOR
        CALL    ESCREVE             ; LINHA 1 / COLUNA 1
        BSF    RS                  ; SELECCIONA O DISPLAY P/ DADOS

        MOVF    CENTENA,W
        ADDLW 0X30                ; FAZ AJUSTE ASCII DA CENTENA
        CALL    ESCREVE             ; ESCREVE VALOR NO LCD

        MOVF    DEZENA,W
        ADDLW 0X30                ; FAZ AJUSTE ASCII DA DEZENA
        CALL    ESCREVE             ; ESCREVE VALOR NO LCD

        MOVF    UNIDADE,W
        ADDLW 0X30                ; FAZ AJUSTE ASCII DA UNIDADE
        CALL    ESCREVE             ; ESCREVE VALOR NO LCD

        GOTO    VARRE              ; VOLTA P/ VARREDURA DOS BOTÕES

; ***** TRATAMENTO DO BOTÃO 2 *****
TRATA_BOTAO_2
        DECFSZ FILTRO_BOTOES,F      ; FIM DO FILTRO ? (RUIDO?)
        GOTO    VARRE              ; NÃO - VOLTA P/ VARRE
                                ; SIM - BOTÃO PRESSIONADO

```

```

DECFSZ TEMPO_TURBO,F      ; FIM DO TEMPO DE TURBO ?

GOTO    VARRE           ; NÃO - VOLTA P/ VARRE

                                ; SIM

MOVlw  TURBO_TECLA

MOVWF TEMPO_TURBO        ; RECARREGA TEMPORIZADOR DO TURBO

                                ; DAS TECLAS

MOVlw .100

XORWF INTENSIDADE_VENT,W

BTFSZ  STATUS,Z          ; PODE INCREMENTAR PWM DO VENTILADOR ?

INCF   INTENSIDADE_VENT,F ; SIM - INCREMENTA INTENSIDADE_VENT

                                ; NÃO

MOVf   INTENSIDADE_VENT,W ; CARREGA INTENSIDADE_VENT NO WORK

MOVWF mulplr               ; CARREGA WORK EM mulplr

MOVlw .10

MOVWF mulcnd               ; CARREGA 10d EM mulcnd

CALL   mpy_F              ; CHAMA ROTINA DE MULTIPLICAÇÃO

SWAPF L_byte,W

ANDlw B'00110000'

IORlw B'00001111'

RRF   H_byte,F

RRF   L_byte,F

RRF   H_byte,F

MOVWF CCP2CON

RRF   L_byte,W

MOVWF CCP2R2L               ; ATUALIZA REGISTRADORES DO DUTY CYCLE

                                ; DO MÓDULO CCP2 - PWM DO VENTILADOR

GOTO    VARRE           ; VOLTA P/ VARREDURA DOS BOTÕES

; ***** TRATAMENTO DO BOTÃO 3 *****

```

TRATA_BOTAO_3

```
DECFSZ FILTRO_BOTOES,F           ; FIM DO FILTRO ? (RUIDO?)  
GOTO    VARRE                   ; NÃO - VOLTA P/ VARRE  
                                ; SIM - BOTÃO PRESSIONADO  
  
DECFSZ TEMPO_TURBO,F           ; FIM DO TEMPO DE TURBO ?  
GOTO    VARRE                   ; NÃO - VOLTA P/ VARRE  
                                ; SIM  
MOVLW  TURBO_TECLA  
MOVWF  TEMPO_TURBO            ; RECARREGA TEMPORIZADOR DO TURBO  
                                ; DAS TECLAS  
  
MOVF   INTENSIDADE_VENT,F  
BTFSZ  STATUS,Z                ; PODE DECREMENTAR PWM DO VENTILADOR ?  
DECF   INTENSIDADE_VENT,F      ; SIM - DECREMENTA INTENSIDADE_VENT  
                                ; NÃO  
  
MOVF   INTENSIDADE_VENT,W      ; CARREGA INTENSIDADE_VENT NO WORK  
MOVWF  mulplr                 ; CARREGA WORK EM mulplr  
  
MOVLW  .10  
MOVWF  mulcnd                 ; CARREGA 50d EM mulcnd  
  
CALL   mpy_F                  ; CHAMA ROTINA DE MULTIPLICAÇÃO  
  
SWAPF L_byte,W  
ANDLW  B'00110000'  
IORLW  B'00001111'  
RRF    H_byte,F  
RRF    L_byte,F  
RRF    H_byte,F  
MOVWF CCP2CON  
RRF    L_byte,W  
MOVWF CCPR2L                   ; ATUALIZA REGISTRADORES DO DUTY CYCLE  
                                ; DO MÓDULO CCP2 - PWM DO VENTILADOR  
Conectando o PIC 16F877A - Recursos Avançados
```

GOTO VARRE ; VOLTA P/ VARREDURA DOS BOTOES

;*****

.* TABELAS DE CONVERSAO P/ TEMPERATURA DO DIODO *

;*****

ORG 0X200 ; POSICIONA O INICIO DA TABELA EM 0X400

RADIX DEC ; CONFIGURA RADIX EM DECIMAL

TABELA_TEMPERATURA

MOVLW HIGH TABELA_TEMP

MOVWF PCLATH ; ACERTA VALOR DO PCLATH

MOVLW LOW TABELA_TEMP ; CARREGA NO WORK PARTE BAIXA DO PC

ADDWF TEMPERATURA,W ; SOMA AO DESLOCAMENTO (TEMPERATURA)

BTFS C STATUS,C ; HOUVE ESTOURO ?

INCF PCLATH,F ; SIM - INCREMENTA PCLATH (PARTE ALTA DO PC)

MOVWF PCL ; MOVE WORK P/ PCL

; PROVOCA UM SALTO

TABELA_TEMP

DT 000,000,000,000,000,000,000,000,000,000,000,000,000,000,010,000 ;15

DT 000,000,000,000,000,000,000,000,000,000,000,000,000,000,001,001 ;31

DT 002,002,003,003,004,004,005,005,006,006,007,007,008,008,009,009 ;47

DT 010,010,011,011,012,012,013,013,014,014,015,015,016,016,017,017 ;63

DT 018,018,019,019,020,020,021,021,022,022,023,023,023,024,024,025 ;79

DT 025,026,026,027,027,028,028,029,029,030,030,031,031,032,032,033 ;95

DT 033,034,034,035,035,036,036,037,037,038,038,039,039,040,040,041 ;111

DT 041,042,042,043,043,044,044,045,045,046,046,047,047,047,048,048,049 ;127

DT 049,050,050,051,051,052,052,053,053,054,054,055,055,056,056,057 ;143

DT 057,058,058,059,059,060,060,061,061,062,062,063,063,064,064,065 ;159

DT 065,066,066,067,067,068,068,069,069,070,070,071,071,072,072,073 ;175

DT 073,074,074,075,075,076,076,077,077,078,078,079,079,080,080,081 ;191

DT 081,082,082,083,083,084,084,085,085,086,086,087,087,088,088,089 ;207
DT 089,090,090,091,091,092,092,093,093,094,094,095,095,096,096,097 ;223
DT 097,098,098,099,099,100,100,101,101,102,102,103,103,104,104,104 ;239
DT 105,105,106,106,107,107,108,108,109,109,110,110,111,111,112,112 ;255

;*****

;* FIM DO PROGRAMA *

;*****

END

; FIM DO PROGRAMA



Detalhamento dos Registradores Especiais (SFRs)

Introdução

Este apêndice destina-se a apresentar um detalhamento de todos os registradores especiais (SFRs) disponíveis no PIC 16F877A, para auxiliá-lo na programação dos seus sistemas.

Para efeito de padronização, cada bit dentro destes 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.
Agrupamento e localização

Gerais

STATUS.....	303
OPTION_REG.....	304
PCON.....	305
PCLePCLATH	305
FSReoINDF	305

Portas

PORTAeTRISA	306
PORTBeTRISB	306
PORTCeTRISC	306
PORTDeTRISD	307
PORTEeTRISE	307

Timers

TMRO.....	308
T1CON,TMR1LeTMR1H	308
T2CON, TMR2 e PR2.....	309

Interrupções

INTCON.....	310
PIE1	311
PIR1.....	312
PIE2.....	313
PIR2.....	313

Conversor A/D

ADCON0.....	314
ADCON1	315

ADRESLeADRESH.....	315
Compare / Capture / PWM	
CCP1CON.....	316
CCP1LeCCPR1H	316
CCP2CON	317
CCPR2LeCCPR2H	317
E²PROM	
EECON1 e EECON2	318
EEADReEEADRH	318
EEDATAeEEDATH	319
SSP	
SSPCON.....	320
SSPCON2.....	321
SSPSTAT	321
SSPADD	322
SSPBUF	322
USART	
TXSTA.....	322
RCSTA	323
TXREGeRCREG	324
SPBRG	325
Comparadores	
CMCON	324
CVRCON	326

STATUS

Registrador: STATUS				Endereços: 03h, 103h, 183h			
Bit 7	Bit 6	Bit5	Bit 4	Bit 3	Bit 2	Bit1	Bit0
R/W	R/W	R/W	R	R	R/W	R/W	R/W
IRP	RP1	RP0	/TO	/PD	Z	DC	C
Condição en Power-On Reset (PQR)							
0	0	0	1	1	X	X	X

IRP: Seletor de banco de memória de dados usado para endereçamento indireto:

0 = Banco O e 1 (00h - FFh).
1 = Banco2e3(100h-1FFh).

RP1

RPO: Seletor de banco de memória de dados usado para endereçamento direto:

00 = Banco O (00h - 7Fh).
01 = Banco 1 (80h - FFh).
10 = Banco2(100h-17Fh).
11 = Banco3(180h-1FFh).

/TO:

Indicação de *Time-out*:
0 = Indica que ocorreu um estouro do *WatchDog* (WDT).
1 = Indica que ocorreu um power-up ou foram executadas instruções CLRWDI 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 CLRWDI.

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_REG

Registrador: OPTION_REG				Endreços: 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
/RBPU	INTEDG	T0CS	TOSE	PSA	PS2	PS1	PS0
Condição em Power-On Reset (POR)							
1	1	1	1	1	1	1	1

/RBPU: 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 **RB0**:
 0 = A interrupção ocorrerá na borda de descida.
 1 = A interrupção ocorrerá na borda de subida.

TOCS: Configuração do incremento para o **TMR0**:
 0 = **TMR0** será incrementado internamente pelo clockda máquina.
 1 = **TMR0** será incrementado externamente pela mudança no pino **RA4/TOCKI**.

TOSE: Configuração da borda que incrementará o **TMR0** no pino **RA4/TOCKI**, quando **TOCS=1**:
 0 = O incremento ocorrerá na borda de subida de **RA4/TOCKI**.
 1 = O incremento ocorrerá na borda de descida de **RA4/TOCKI**.

PSA: Configuração de aplicação do *prescaler*:
 0 = 0 prescaler será aplicado ao **TMR0**.
 1 = O prescaler será aplicado ao WDT.

PS2

PS1

PS0: Configuração de valor de prescaler:

PS		
2/1/0	TMR0	WDT
000	1:21:1	
000	1:41:2	
010	1:81:4	
011	1:16 1:8	
100	1:32 1:16	
101	1:64 1:32	
110	1:128 1:64	
111	1:256 1:128	

PCON

Registrador PCON				Endereços: 8EH			
Bit 7 U -	Bit 6 U -	Bit 5 U -	Bit 4 U -	Bit 3 U -	Bit 2 U -	Bit 1 R/W -	Bit 0 R/W /POR /BOR
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	1

/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, 182h			
Bit 7 R/W	Bit 6 R/W	Bit 5 R/W	Bit 4 R/W	Bit 3 R/W	Bit 2 R/W	Bit 1 R/W	Bit 0 R/W
Parte baixa do PC							
0	0	0	0	0	0	0	0

Registrador: PCLATH				Endereços: 0Ah, 8Ah, 10Ah e 18Ah			
Bit 7 U	Bit 6 U	Bit 5 U	Bit 4 R/W	Bit 3 R/W	Bit 2 R/W	Bit 1 R/W	Bit 0 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 R/W	Bit 6 R/W	Bit 5 R/W	Bit 4 R/W	Bit 3 R/W	Bit 2 R/W	Bit 1 R/W	Bit 0 R/W
Ponteiro para endereçamento							
Condição em Ponteiro de 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	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
U	U	R/W	R/W	R/W	R/W	R/W	R/W
-	-	RA5	RA4	RA3	RA2	RA1	RA0
Condição em Power-On Reset (POR)							
-	-	0	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
U	U	R/W	R/W	R/W	R/W	R/W	R/W
-	-	Ref. RA5	Ref. RA4	Ref. RA3	Ref. RA2	Ref. RA1	Ref. RA0
Condição em Power-On reset (POR)							
-	-	1	1	1	1	1	1

PORTB e TRISB

Registrador: PORTB				Endereços: 06h e 106h			
Bit 7	Bit 6	BitS	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
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: 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/W	R/W	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

PORTC e TRISC

Registidor: PORTC				Endereços: 07h			
Bit 7	Bit 6	BitS	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RW	R/W	R/W	R/W	R/W	R/W	R/W	R/W
RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

Registrador: TRISC				Endereços: 87h			
Bit 7	Bit 6	BitS	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
Ref. RC7	Ref. RC6	Ref. RC5	Ref. RC4	Ref. RC3	Ref. RC2	Ref. RC1	Ref. RC0
Condição em Power-On Reset (POR)							
1	1	1	1	1	1	1	1

PORTD e TRISD

Registrador: PORTD								Endereços: 08h							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	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	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
X	X	X	X	X	X	X	X	Condição em Power On-Reset							

Registrador: TRISD								Endereços: 88h							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	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	Ref. RD7	Ref. RD6	Ref. RD5	Ref. RD4	Ref. RD3	Ref. RD2	Ref. RD1	Ref. RD0
Condição em Power-On Reset (POR)															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

PORTE e TRISE

Registrador: PORTE								Endereços: 09h							
Bit 7	Bit 6	BitS	Bit 4	Bit 5	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	BitS	Bit 4	Bit 5	Bit 2	Bit 1	Bit 0
U	U	U	U	U	R/W	R/W	R/W	-	-	-	-	RE2	RE1	RE0	
Condição em Power-On Reset (POR)															
-	-	-	-	-	X	X	X	-	-	-	-	X	X	X	

Registrado: TRISE								Endereços: 89h							
Bit 7	Bit 6	BitS	Bit 4	Bit 5	Bit 2	Bit 1	Bit 0	Bit 7	Bit 6	BitS	Bit 4	Bit 5	Bit 2	Bit 1	Bit 0
R	R	R/W	R/W	U	R/W	R/W	R/W	IBF	OBF	IBOV	PSPMOD	-	Ref. RE2	Ref. RE1	Ref. RE0
Condição em Power-On Reset (POR)															
0	0	0	0	-	1	1	1	-	-	-	-	-	-	-	

IBF: Indicação de buffer de entrada da porta paralela (PSP):
 0 = 0 buffer de entrada está vazio.
 1 = Uma palavra foi recebida e está aguardando para ser lida.

OBF: Indicação de buffer de saída da porta paralela (PSP):
 1 = 0 buffer de saída foi lido.
 0 = Existe uma palavra não lida no buffer de saída.

IBOV: Indicação de Overflow no buffer de entrada:
 1 = Não houve Overflow.
 0 = Uma nova palavra chegou sem que a anterior tivesse sido lida do buffer. (este bit deve ser limpo por software)

PSPMODE: Modo de operação dos pinos da porta paralela (PSP):
 1 = PSP desabilitada. Pinos como I/Os convencionais.
 0 = PSP habilitada. Pinos controlados internamente pelo sistema.

TMR0

Registrados TMR0				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 TMR0 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	BitS	Bit 2	Bit 1	Bit 0
U	U	R/W	R/W	R/W	R/W	R/W	R/W
-	-	T1CKPS	T1CKPS T10SCE /T1SYNC TMR1CS TMR1ON				
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

T1CKPS1

T1CKPS0: Ajuste do *prescale* do Timer 1:

- 00 = prescale de 1:1.
- 01 = prescale de 1:2.
- 10 = prescale de 1:4.
- 11 = prescale 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 TMR1 CS=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 (Fosc/4)
- 1 = Clock externo no pino **T10SO/T1CKI**.

TMR1ON: Habilitação do Timer 1:

- 0 = Timer 1 desabilitado. Paralisa o contador do Timer 1.
- 1 = Timer 1 habilitado.

Registradre: TMR1L				Endereços: 0Eh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Contador TMR1 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 TMR1 de 16 bits - Parte Alta							
Condição em Power On-Reset (POR)							
X	X	X	X	X	X	X	X

T2CON,TMR2ePR2

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	TOUTP O	TRM20N	T2CKPS1	T2CKPS0
Condição em Power-On Reset (POR)							
-	0	0	0	0	0	0	0

TOUTPS3

TOUTPS2

TOUTPS1

TOUTPS0

Ajuste do postscale:

- | | |
|-------------------------|--------------------------|
| 000= postscale de 1:1 | 1000 = postscale de 1:9 |
| 001= postscale de 1:2 | 1001 = postscale de 1:10 |
| 010= postscale de 1:3 | 1010 = postscale de 1:11 |
| 0011= postscale de 1:4 | 1011 = postscale de 1:12 |
| 0100 = postscale de 1:5 | 1100 = postscale de 1:13 |
| 0101= postscale de 1:6 | 1101 = postscale de 1:14 |
| 0110 = postscale de 1:7 | 1110 = postscale de 1:15 |
| 0111= postscale de 1:8 | 1111 = postscale de 1:16 |

TMR20N:

Habilitação do Timer 2:

- 0 = Timer 2 desabilitado. Paralisa o contador do Timer 2.
1 = Timer 2 habilitado.

T2CKPS1

T2CKPS0:

Ajuste do prescale:

- | |
|----------------------|
| 00 = presca/ede1:1 |
| 01 = prescale de 1:4 |
| 10 = presca/ede1:16 |
| 11 =presca/ede1:16 |

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 TMR2 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	Bit3	Bit 2	Bit 1	Bit0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Límite superior para contagem do TMR2							
Condição em Power-On Reset (POR)							
1	1	1	1	1	1	1	1

INTCON

Registrador: INTCON				Endereços: 0Bh, 8Bh, 10Bh e 18Bh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit3	Bit 2	Bit1	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 geral p/ 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 **TMR0** (chave individual):

0 = Interrupção de **TMR0** desabilitada.

1 = Interrupção de **TMR0** 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 TMRO (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).

PIE1

Registrador: PIE1				Endereços: 8Ch			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit1	Bit 0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

PSPIE: Habilitação da interrupção da porta paralela (chave individual):
0 = Interrupção da porta paralela desabilitada.
1 = Interrupção da porta paralela habilitada.

ADIE: Habilitação da interrupção do conversor A/D (chave individual):
0 = Interrupção do conversor A/D desabilitada.
1 = Interrupção do conversor A/D 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.

SSPIE: Habilitação da interrupção da porta serial SSP (chave individual):
0 = Interrupção da porta serial SSP desabilitada.
1 = Interrupção da porta serial SSP 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.

PIR1

Registrador: PIE1				Endereços: 0Ch			
Bit 7	Bit 6	Bit5	Bit 4	Bit 3	Bit 2	Bit1	Bit 0
R/W	R/W	R	R	R/W	R/W	R/W	R/W
PSPIE	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

- PSPIF:** Identificação da interrupção da porta paralela (chave individual):
 0 = Não ocorreu evento de leitura/escrita na porta paralela.
 1 = Ocorreu evento de leitura/escrita na porta paralela (este bit deve ser limpo por software).
- ADIF:** Identificação da interrupção do conversor A/D (chave individual):
 0 = Conversão A/D não foi completada.
 1 = Conversão A/D foi completada.
- RCIF:** Identificação da interrupção de recepção da USART (chave individual):
 0 = Buffer de recepção da USART está vazio.
 1 = Buffer de recepção da USART está cheio.
- TXIF:** Identificação da interrupção de transmissão da USART (chave individual):
 0 = Buffer de transmissão da USART está cheio.
 1 = Buffer de transmissão da USART está vazio.
- SSPIF:** Identificação da interrupção da porta serial SSP (chave individual):
 0 = Não ocorreu condição de interrupção no módulo SSP.
 1 = Ocorreu condição de interrupção no módulo SSP (este bit deve ser limpo por software).
- CCP1IF:** Identificação da interrupção do módulo CCP1 (chave individual):
 0 = Não ocorreu condição de interrupção no módulo CCP1.
 1 = Ocorreu condição de interrupção no módulo CCP1 (este bit deve ser limpo por software).
- TMR2IF:** Identificação da interrupção de Timer 2 (chave individual):
 0 = Não ocorreu evento da interrupção de Timer 2.
 0 = Ocorreu evento da interrupção de Timer 2 (este bit deve ser limpo por software).
- 1=TMR1IF:** Identificação da interrupção de estouro do Timer 1 (chave individual):
 0 = Não ocorreu estouro do Timer 1.
 1 = Ocorreu estouro do Timer 1 (este bit deve ser limpo por software).

PIE2

Registrador: PIE				Endereços: 8Dh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
U	R/W	U	R/W	R/W	U	U	R/W
-	-	-	EEIE	BCLIE	-	-	CCP2IE
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

EEIE: Habilitação da interrupção de fim de escrita na E²PRÓM (chave individual):

- 0 = Interrupção de fim de escrita na E²PRÓM desabilitada.
- 1 = Interrupção de fim de escrita na E²PRÓM habilitada.

BCLIE: Habilitação da interrupção de colisão na linha I²C (chave individual):

- 0 = Interrupção de colisão na linha I²C desabilitada.
- 1 = Interrupção de colisão na linha I²C habilitada.

CCP2IE: Habilitação da interrupção do módulo CCP2 (chave individual):

- 0 = Interrupção de CCP2 desabilitada.
- 1 = Interrupção de CCP2 habilitada.

Obs: O bit 6 é reservado e deve ser mantido sempre em 0.

PIR2

Registrador: PIR2				Endereço: 8Dh			
Bit7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
U	R/W	U	R/W	R/W	U	U	R/W
-	-	-	EEIF	BCLIF	-	-	CCP2IF
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

EEIF: Identificação da interrupção de fim de escrita na E²PRÓM (chave individual):

- 0 = Operação de escrita na E²PRÓM não foi completada e nova escrita não pode ser iniciada.
- 1 = Operação de escrita na E²PRÓM foi completada (este bit deve ser limpo por software).

BCLIF: Identificação da interrupção de colisão na linha I²C (chave individual):

- 0 = Não ocorreu colisão na linha I²C.²
- 1 = Ocorreu colisão na linha I²C.²

CCP2IF: Identificação da interrupção do módulo CCP2 (chave individual):

- 0 = Não ocorreu condição de interrupção no módulo CCP2
- 1 = Ocorreu condição de interrupção no módulo CCP2 (este bit deve ser limpo por software).

Obs: O bit 6 é reservado e deve ser mantido sempre em 0.

ADCON0

Registrador: ADCON0				Endereço: 1Fh			
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	U	R/W
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	-	ADON
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

ADCS1

ADCS0: Especificação do clock para o conversor A/D:

00 = $F_{osc} / 2$

01 = $F_{osc} / 8$

10 = $F_{osc} / 32$

11 = F_{RC} . Usado oscilador RC interno, específico para o conversor A/D.

CHS2

CHS1

CHS0: Seleção do canal para conversão:

000 = Canal0

001 = Canal1(RA1/AN1)

010 = Canal2(RA2/AN2)

011 = Canal3(RA3/AN3)

100 = Canal4(RA5/AN4)

101 = Canal5(RE0/AN5)

110 = Canal6(RE1/AN6)

111 = Canal7(RE2/AN7)

GO/DONE:

Situação da conversão A/D. Válido somente quando **ADON=1**:

0 = Conversão não sendo efetuada. Este bit é limpo automaticamente quando a conversão termina.

1 = Conversão em andamento. Este bit deve ser selado para iniciar a conversão.

ADON:

Ativa/Desativa o sistema de conversão A/D:

0 = Conversor desativado para economizar energia.

1 = Conversor ativado.

ADCON1

Registrador: ADCON1				Endereços: 9Fh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	U	U	U	R/W	R/W	R/W	R/W
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0
0							

ADFM:

Justificação para o resultado da conversão A/D de 10-bits:

1 = Justificado à direita. Os 6 bits mais significativos de ADRESH são lidos em 0 (zero). 0 = Justificado à esquerda. Os 6 bits menos significativos de ADRESL são lidos em 0 (zero).

PCFG3, PCFG2, PCFG1

PCFG0: Configuração dos pinos analógicos / digitais e as tensões de referência:

PCFG3 PCFG0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2	AN1	AN0			Anal.	Ext.
00000	A	A	A	A	A	A	A	A	V _{DD}	V _{ss}	8	0
00001	A	A	A	A	V _{REF+}	A	A	A	RA3	V _{ss}	7	1
00110	D	D	D	A	A	A	A	A	V _{DD}	V _{ss}	5	0
00111	D	D	D	A	V _{REF+}	A	A	A	RA3	V _{ss}	4	1
01000	D	D	D	D	A	D	A	A	V _{DD}	V _{ss}	3	0
01011	D	D	D	D	V _{REF+}	D	A	A	RA3	V _{ss}	2	1
01110	D	D	D	D	D	D	D	D	V _{DD}	V _{ss}	0	0
01111	D	D	D	D	D	D	D	D	V _{DD}	V _{ss}	0	0
10000	A	A	A	A	V _{REF+}	V _{REF+}	A	A	RA3	RA2	6	2
10001	D	D	A	A	A	A	A	A	V _{DD}	V _{ss}	6	0
10110	D	D	A	A	V _{REF+}	A	A	A	RA3	V _{ss}	5	1
10111	D	D	A	A	V _{REF+}	V _{REF-}	A	A	RA3	RA2	4	2
11000	D	D	D	A	V _{REF+}	V _{REF-}	A	A	RA3	RA2	3	2
11011	D	D	D	D	V _{REF+}	V _{REF-}	A	A	RA3	RA2	2	2
11110	D	D	D	D	D	D	D	A	V _{DD}	V _{ss}	1	0
11111	D	D	D	D	V _{REF+}	V _{REF-}	D	A	RA3	RA2	1	2

Pinos configurados como analógicos. A=Analógico / D=Digital

ADRESL e ADRESH

Registrados: ADRESH								Endereços: 1Eh				
Resultado da conversão A/D - Parte Alta								Endereços: 9Eh				
Condição em Power-On Reset (POR)												
X	X	X	X	X	X	X	X	X	X	X	X	X
X			X		X		X		X		X	
Registrador: ADRESL								Endereços: 9Eh				
Resultado da conversão A/D - Parte Baixa								Endereços: 9Eh				
Condição em Power-On Reset (POR)												
X	X	X	X	X	X	X	X	X	X	X	X	X

CCP1CON

Registrador: CCP1CON				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 *prescale* de 1:1.

0101 = Capture ligado para borda de subida com *prescale* de 1:1.

0110= Capture ligado para borda de subida com *prescale* de 1:4.

0111= Capture ligado para borda de subida com *prescale* de 1:16.

1000= Compare ligado. Pino de saída (**RC2**) será setado (1) quando o compare ocorrer.

1001= Compare ligado. Pino de saída (**RC2**) será zerado (0) quando o compare ocorrer.

1010= Compare ligado. Pino de saída (**RC2**) não será afetado.

1011= Compare ligado. Pino de saída (**RC2**) não será afetado. **TMR1** será resetado.

1100 = PWM ligado

1101= PWM ligado

1110= PWM ligado

1111= PWM ligado

CCPR1L e CCPR1H

Registrador: CCPR1H				Endereços: 16h							
Registrador do CCP1 - Parte Alta											
Condição em Power-On Reset (POR)											
X	X	X	X	X	X	X	X				

Registrador: CCP1L				Endereços: 15h							
Registrador do CCP1 - Parte Baixa											
Condição em Power-On Reset (POR)											
X	X	X	X	X	X	X	X				

CCP2CON

Registrador: CCP2CON				Endereços: 1Dh			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit1	Bit 0
U	U	R/W	R/W	R/W	R/W	r/w	R/W
-	-	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

CCP2X

CCP2Y: Parte baixa do PWM de 10 bits. A parte alta fica em CCPR2L. Válido somente quando em

PWM.

CCP2M3

CCP2M2

CCP2M1

CCP2M0: Seleção do modo CCP2 - Compare/Capture/PWM:

0000 = Modo desligado.

0100 = Capture ligado para borda de descida com presca/e de 1:1.

0101= Capture ligado para borda de subida com presca/e de 1:1.

0110 = Capture ligado para borda de subida com presca/e de 1:4.

0111= Capture ligado para borda de subida com presca/e de 1:16.

1000= Compare ligado. Pino de saída (**RC1**) será setado (1) quando o compare ocorrer.

1001= Compare ligado. Pino de saída (**RC1**) será zerado (0) quando o compare ocorrer.

1010= Compare ligado. Pino de saída (**RC1**) não será afetado.

1011= Compare ligado. Pino de saída (**RC1**) não será afetado. **TMR1** será resetado. Uma conversão A/D será iniciada.

1100= PWM ligado

1101= PWM ligado

1110 = PWM ligado

1111= PWM ligado

CCPR2L e CCPR2H

Registrador: CCPR2H				Endereços: 1Ch							
Registrador do CCP2 - Parte Alta											
Condição em Power-On Reset (POR)											
X	X	X	X	X	X	X	X				

Registrador: CCPR2L				Endereços: 1Bh							
Registrador do CCP2 - Parte Baixa											
Condição em Power-On Reset (POR)											
X	X	X	X	X	X	X	X				

EECON1 e EECON2

Registrador: EECON1				Endereços: 18Ch			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	U	U	U	R/W	R/W	R/W	R/W
EEPGD	-	-	-	WRERR	WREN	WR	RD
Condição em Power-On Reset (POR)							
X	0	0	0	X	0	0	0

- EEPGD:** Seleção do acesso da E²PROM:
 0 = Acessa a memória de dados.
 1 = Acessa a memória de programa.
- WRERR:** Identificação de erro durante a escrita na E²PROM:
 0 = Não ocorreu erro, a escrita foi completada.
 1 = Um erro ocorreu por uma escrita não terminada (um reset pode ter ocorrido).
- WREN:** Habilitação de escrita na E²PROM (bit de segurança):
 0 = Não habilita a escrita na E²PROM.
 1 = Habilita a escrita na E²PROM.
- WR:** Ciclo de escrita na E²PROM:
 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 E²PROM:
 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: 18Dh			
Registrador de proteção para escrita na E PROM (não é um registrador implementado)							
Condição em Power-On Reset (POR)							
-	-	-	-	-	-	-	-

EEADR e EEADRH

Registrador: EEADR				Endereços: 10Dh			
Registrador de endereço da E PROM - Parte Baixa							
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

Registrador: EEADRH				Endereços: 10Fh			
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
-	-	-	Registrador de endereço da E PROM - Parte Alta (só para memória de programa)				
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

EEDATA e EEDATH

Registrador: EEDATA				Endereços: 10Ch			
Registrador de dado da E PROM - Parte Baixa							
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X
Registrador: EEDATH				Endereços: I0Eh			
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
-	-	-	Registrador de dado da E2PROM - Parte Alta (só para memória de programa)				
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

SSPCON

Registrador: SSPCON				Endereços: 14h			
Bit 7	Bit 6	BitS	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
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

- WCOL:** Colisão na escrita do registrador **SSPBUF**:
 Master: 0 = Sem colisão.
 1 = Ocorreu uma escrita em momento indevido.
 Slave: 0 = Sem colisão.
 1 = Ocorreu uma escrita enquanto ainda estava transmitindo
 (deve ser limpo em software).
- SSPOV:** Erro de Overflow na recepção:
 0 = Sem erro.
 1 = Um novo byte foi recebido antes do registrador **SSPBUF** ser lido
 (deve ser limpo em software).
- SSPEN:** Habilitação da porta SSP:
 SPI: 0 = SSP desabilitada. Pinos **SCK**, **SDO**, **SDI** e **/SS** como I/Os convencionais.
 1 = SSP habilitada. Pinos **SCK**, **SDO**, **SDI** e **/SS** controlados internamente.
 I²C: 0 = SSP desabilitada. Pinos **SCL** e **SDA** como I/Os convencionais.
 1 = SSP habilitada. Pinos **SCL** e **SDA** controlados internamente.
- CKP:** Controle do Clock
 SPI: 0 = Clock padrão em nível alto (1).
 1 = Clock padrão em nível baixo (0).
 I²C Slave: 0 = Clock habilitado.
 1 = Clock travado (Pausa). Pino **SCL** mantido em nível baixo (0).
 I²C Master Sem uso

SSPM3

SSPM2

SSPM1

SSPM0:

Seleção do modo de trabalho da porta SSP:

0000 = SPI Master. Clock= Fosc/ 4.

0001 = SPI Master. Clock= Fosc/16.

0010 = SPI Master. Clock= Fosc/ 64.

0011 = SPI Master. Clock = **TMR2 / 2**.

0100 = SPI Slave. Pino **/SS** habilitado.

0101 = SPI Slave. Pino **/SS** desabilitado (pode ser usado como I/O).

0110 = I²C Slave. Endereçamento de 7-bits.

0111 = I²C Slave. Endereçamento de 10-bits.

1000 = PC Master. Clock = Fosc/ (4 x (**SSPADD+1**)) Outros =

Reservados

SSPCON2

Registrador : SSPCON2				Endereços: 91h				
Bit 7 R/W GCEN	Bit 6 R/W ACKSTA	Bit 5 R/W ACKDT	Bit 4 R/W ACKEN	Bit 3 R/W RCEN	Bit 2 R/W PEN	Bit 1 R/W RSEN	Bit 0 R/W SEN	
Condição em Power-On Reset (POR)								
0	0	0	0	0	0	0	0	0

GCEN: Habilitação do endereço global (somente para I²C Slave):
0 = Endereço global desativado.

1 = Endereço global ativado. Quando recebido endereço 0 (zero) o Slave aceita como endereço válido.

ACKSTAT: Valor recebido como ACK após a transmissão de um dado (somente para I²C Master):
0 = ACK recebido do Slave.
1 = ACK não recebido do Slave.

ACKDT: Valor a ser respondido como ACK após uma recepção (somente para I²C Master):
0 = Continua recebendo mais bytes do Slave.
1 = O Slave deve parar de transmitir dados para o Master.

ACKEN: Gera uma condição de acknowledge com o valor de **ACKDT** (somente para I²C Master)
0 = Não gera condição ou condição já finalizada.
1 = Inicia condição e indica condição ainda em execução. Limpo automaticamente pelo hardware.

RCEN: Ativa o sistema de recepção de dados (somente para I²C Master):
0 = Master não pode receber dados.
1 = Master pronto para receber dados do Slave.

PEN: Gera uma condição de Stop na linha (somente para I²C Master):
0 = Não gera condição ou condição já finalizada.
1 = Inicia condição e indica condição ainda em execução. Limpo automaticamente pelo hardware.

RSEN: Gera uma condição de Re-Star na linha (somente para I²C Master):
 0 = Não gera condição ou condição já finalizada.
 1 = Inicia condição e indica condição ainda em execução. Limpo automaticamente pelo hardware.

SEN: Gera uma condição de Start na linha (somente para I²C Master):
 0 = Não gera condição ou condição já finalizada.
 1 = Inicia condição e indica condição ainda em execução. Limpo automaticamente pelo hardware.

SSPSTAT

Registrador: SSPSTAT				Endereços: 94h			
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
R/W	R/W	R	R	R	R	R	R
SMP	CKE	D/A	P	S	R/W	UA	BF
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

SMP: Sistema de amostragem de dados:
 SPI: 0 = A recep. é feita na borda do meio do período. Modo obrigatório para Slave.
 1 = A recepção é feita na borda do fim do período. Pode ser usado no Master. PC:
 I²C 0 = Habilita controle de sinal para alta velocidade (400 kHz).
 1 = Desabilita controle de sinal para alta velocidade (100 kHz).

CKE: Sistema de amostragem de dados:
 SPI: 0 = Pulso do clock no início do período.
 1 = Pulso do clock no final do período.
 I²C: 0 = Entradas conforme especificações I²C.
 1 = Entradas conforme especificações SMBUS.

D/A: Indicação de Dados ou Endereço (somente para I²C):
 0 = Último byte recebido ou transmitido foi um endereço.
 1 = Último byte recebido ou transmitido foi um dado.

P: Indicação de Sfop (somente para I²C):
 0 = Não ocorreu uma condição de Stop.
 1 = Ocorreu uma condição de Sfop. Limpo quando SSP desabilitada.

S: Indicação de S/arí (somente para I²C):
 1 = Não ocorreu uma condição de Sfart.
 0 = Ocorreu uma condição de Sfart. Limpo quando SSP desabilitada.

R/W: Indicação de Leitura/Escrita (somente para I²C):
 Slave: 0 = Escrita.
 1 = Leitura.
 Master. 0 = Nenhuma transmissão em progresso.
 1 = Trans. em progresso. Limpo automaticamente pelo hardware no final.

UA: Indicação de atualização de endereço (somente para I²C com 10 bits):
 0 = Não precisa atualizar o endereço.
 1 = Deve atualizar o endereço (parte alta ou parte baixa) para nova comparação.

BF: Indicação de Buffer cheio:
 Recebe: 0 = Recepção ainda não foi completada.
 Buffer **SSPBUF** vazio.
 1 = Recepção terminada. Buffer SSPBUF cheio.

Transmite (I_C):
 0 = Transmissão terminada. Buffer SSPBUF vazio.
 1 = Transmissão em progresso. Buffer SSPBUF ainda cheio.

SSPADD

Registrador: SSPADD								Endereços: 93h							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Endereço da unidade para PC (parte alta e parte baixa)															
Condição em Power-On Reset (POR)														R/W	
0	0	0	0	0	0	0	0								

SSPBUF

Registrador: SSPBUB								Endereços: 13h							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Buffer para recepção e transmissão															
Condição em Power-On Reset (POR)															
X	X	X	X	X	X	X	X								
X															

TXSTA

Registrador: TXSTA				Endereços: 98h			
Bit 7	Bit 6	Bit5	Bit 4	Bits	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 99 bit. Pode ser usado como paridade ou endereçamento.

RCSTA

Registrador: RCSTA				Endereços: 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/W	R	R	R
SPEN	RX9	SREN	CREN	ADDEN	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 obit CREN).
- RX9D:** Valor recebido no 99 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
<i>Buffer para transmissão</i>							
<i>Condição em Power-On Reset (POR)</i>							
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
<i>Buffer para recepção</i>							
<i>Condição em Power-On Reset (POR)</i>							
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
<i>Ajuste do Baud Rate</i>							
<i>Condição em Power-On Reset (POR)</i>							
0	0	0	0	0	0	0	0

CMCON

Registrador: CMCON				Endereços: 9Ch			
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
C20UT	C10UT	C2INV	C1INV	CIS	CM2	CM1	CM0
<i>Condição em Power-On Reset (POR)</i>							
0	0	0	0	0	1	1	1

C20UT: Valor da saída do comparador 2:

Normal (C2INV=0):

0 = C2V_{IN+}<C2V_{IN-}

1=C2V_{IN+}>C2V_{IN-}

Inversa (C2INV=1):

0 = C2V_{IN+}>C2V_{IN-}

1=C2V_{IN+}<C2V_{IN-}

C1OUT: Valor da saída do comparador 1:

Normal (C1INV=0):

0 = C1 V_{IN+}<C1V_{IN-}

1 = C1 V_{IN+}>C1V_{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-} .

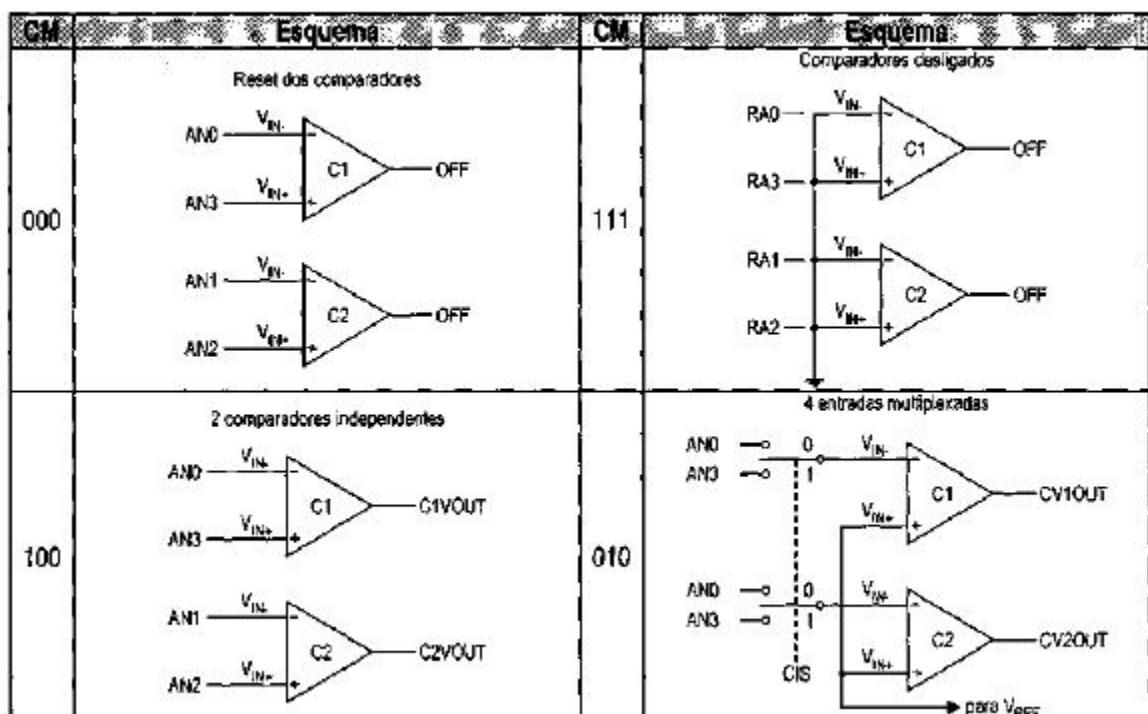
1 = RA3 conectado a C1 V_{IN-} .

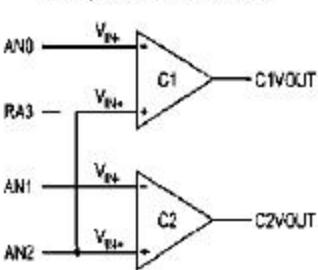
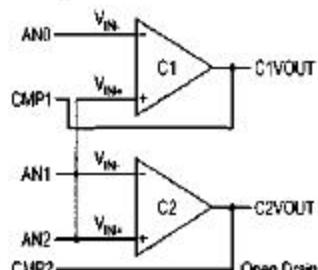
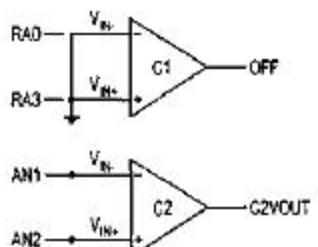
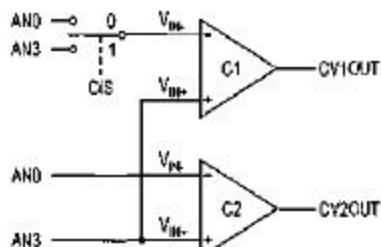
RA2 conectado a C2 V_{IN-} .

CM2

CM1

CM0: Configura a pinagem dos corriparadores (modo de operação):



OM	Esquema	OM	Esquema
011	2 comparadores com ref. comum 	110	2 comparadores com ref. comum e saídas 
011	1 comparador independente 	110	3 entradas multiplexadas 

CVRCON

Registrador: CVRCON				Endereços: 9Dh			
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
CVREN	CVRON	CVRR	-	CVR3	CVR2	CVR1	CVR0
Condição em Power-On Reset (POR)							
0	0	0	0	0	0	0	0

CVREN: Energização do sistema de tensão de referência:
0 = Circuito de vref desenergizado.
1 = Circuito de vref energizado.

CVRON: Habilitação da saída de vref:
0 = Tensão de referência desligada.
1 = Tensão de referência ligada ao pino RA2.

CVRR: Seleção do range de operação do sistema de vref:
0 = Range baixo.
1 = Range alto.

CVR3
CVR2
CVR1
CVR0: Seleção do valor da tensão de vref:
Se CVRR = 1:
 $V_{REF} = (VR/24) * V_{DD}$
Se CVRR = 0:
 $V_{REF} = 1/4 * V_{DD} + (VR/32) * V_{DD}$

CCP1CON, CCPR1L e CCPR1H

Registrador: CCP1CON				Endereços: 17h			
Bit 7 U -	Bit 6 U -	Bit 5 R/W CCP1X	Bit 4 R/W CCP1Y	Bit 3 R/W CCP1M3	Bit 2 R/W CCP1M2	Bit 1 R/W CCP1M1	Bit 0 R/W 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 presca/erde 1:1.

0101 = Capture ligado para borda de subida com prescaterde 1:1.

0110 = Capture ligado para borda de subida com presca/erde 1:4.

0111 = Capture ligado para borda de subida com presca/erde 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 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 R/W	Bit 6 R/W	Bit 3 R/W	Bit 4 R/W	Bit 3 R/W	Bit 2 R/W	Bit1 R/W	Bit 0 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 R/W	Bit 6 R/W	Bit 3 R/W	Bit 4 R/W	Bit 3 R/W	Bit 2 R/W	Bit1 R/W	Bit 0 R/W
Registrador de controle do CCP1 - Parte alta							
Condição em Power-On Reset (POR)							
X	X	X	X	X	X	X	X

Resumo e condições após reset

Hex	Nome	Bit 7	Bit6	BitS	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Power-on	Outros Resets
BANK 0											
00	INDF	Valor apontado pelo FSR (endereçamento indireto - não é um registro)								-----	
01	TMR0	Contador TMR0 de 8 bits								xxxx xxxx	uuuu uuuu
02	PCL	Parte baixa do PC (8 bits menos significativos)								0000 0000	0000 0000
03	STATUS	IRP	RP1	RP0	/T0	/PD	Z	DC	C	0001 1xxx	000q quuu
04	FSR	Ponteiro para endereçamento indireto								xxxx xxxx	uuuu uuuu
05	PORTA	-	-	RA5	RA4	RA3	RA2	RA1	RA0	--0x xxxx	--u uuuu
06	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
07	PORTC	RC7	RC5	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
08	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
09	PORTE	-	-	-	-	RE2	RE1	RE0		---- -xxxx	uuuu uuuu
0A	PCLATH	-	-	-	Parte alta do PC (5 bits mais significativos)					--0 0000	--0 0000
0B	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x 0000 000u	
0C	PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000 0000 0000	0000
0D	PIR2	-	-	-	EEIF	BCLIF	-	-	CCP2IF	-r-- 0--0	-r0 0-0-
0E	TMR1L	Contador TMR1 de 16-bits (parte baixa)								xxxx xxxx	uuuu uuuu
0F	TMR1H	Contador TMR1 de 16-bits (parte alta)								xxxx xxxx	uuuu uuuu
10	T1CON	-	-	T1CXP S1	T1CKPS 0	T10SCE N	T1SYNC	TMR1CS	TMR1ON	--00	0000 -uuu uuuu
11	TMR2	Contador TMR2 de 8-bits								0000 0000	0000 0000
12	T2CON	-	TOUT PS3	TOUTPS 2	TOUTPS 1	TOUTPS 0	TMR2ON 1	T2CKPS 1	T2CKPS 0	-000 0000	-000 0000
13	SSPBUF	Buffer de recepção/transmissão da porta serial SSP								xxxx xxxx	uuuu uuuu
14	SSPCON	WCOL	\$SPO V	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
15	CCPR1L	Registrador do CCP1 (parte baixa)								xxxx xxxx	uuuu uuuu
16	CCPR1H	Registrador do CCP1 (parte alta)								xxxx xxxx	uuuu uuuu
17	CCP1CON	-	-	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00	0000 --00 0000
18	RCSTA	\$PEN	RX9	SREN	CREN	ADDEN	FERR	QERR	RX9D	0000 000X	0000 000X
19	TXREG	Buffer de transmissão para a USART								0000 0000	0000 0000
1A	RCREG	Buffer de recepção para a USART								0000 0000	0000 0000
1B	CCPR2L	Registrador do CCP2 (parte baixa)								xxxx xxxx	uuuu uuuu
1C	CCPR2H	Registrador do CCP2 (parte alta)								xxxx xxxx	uuuu uuuu
1D	CCP2CON	-	-	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00	0000 --00 0000
1E	ADRESH	Resultado da conversão A/D (parte alta)								xxxx xxxx	uuuu uuuu
1F	ADCON	ADC S1	ADCSO	CHS2	CHS1	CHSO	GO/DON E	-	ADON	0000 00-0	0000 00-0

Hex	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Power-on	Outros Resets
BANK1											
80	INDF	Valor apontado pelo FSR (endereçamento indireto - não é um registro)									---- ----
81	OPTION_REG	/RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82	PCL	Parte baixa do PC (8 bits menos significativos)							0000 0000	0000 0000	
83	STATUS	IRP	RP1	RPO	/TO	/PD	Z	DC	C	0001 1xxx	
84	FSR	Ponteiro para endereçamento indireto							xxxx xxxx	uuuu uuuu	
85	TRISA	-	-	-	RA4	RA3	RA2	RA1	RA0	--- 1 1111	---1 1111
86	TRISB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	1111 1111	1111 1111
87	TRISC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	1111 1111	1111 1111
88	TRISD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	1111 1111	1111 1111
89	TRISE	IBF	OBF	BOV	PSPMO		RE2	RE1	RE0		
8A	PCLATH	-	-	-	Parte alta do PC (5 bits mais significativos)					0000 000x	0000 000u
8B	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
8C	PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8D	PIE2	-	-	-	EEIE	BCLIE	-	-	CCP2IE	-r-- 0--0	-r0 0-0-
8E	PCON	-	-	-	-	-	-	/POR	/BOR	-----qg	-----uu
8F	-	Não implementado							-	-	
90	-	Não Implementado							-	-	
91	SSPCON2	GCEN	ACKST _{AT}	ACKDT	ACKEN	RCEN	FEN	RSEN	SEN	0000 0000	0000 0000
92	PR2	Limite superior para contagem doTMR2							1111 1111	1111 1111	
93	SSPADD	Registrador de endereço para I ^C							000- 0000	0000 0000	
94	SSPSTAT	SMP	CKE	D/A	P	S	R/W	UA	BF	oooooooooooo	0000 0000
95	-	Não imptemendo							-	-	
96	-	Nêo implementado							-	-	
97	-	Não implementado							-	-	
98	TXSTA	CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D	0000 -010	0000 -010
99	SPBRG	Registrador de ajuste do Baud Rate da USART							0000	0000 0000	
9A	-	NaoImplementado							0000	0000	
9B	-	Não implementado							-	-	
9C	CMCON	C20UT	C10UT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	0000 0111
9D	CVRCN	CVREN	CVROE	CVRR	-	CVR3	CVR2	CVR1	CVR0	000- 0000	000- 0000
9E	ADRESL	Resultado da conversão A/D (parte baixa)							xxxx xxxx	uuuu uuuu	
9F	ADCON1	ADFM	-	-	-	PCFG ₃	PCFG ₂	PCFG1	PCFG0	0--- 00-0	0-- 0000

Hex	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	BIT 2	BIT 1	Bit 0	Power-on	Outros Resets	
BANK 2												
100	IDEF	Valor apontado pelo FSR (endereçamento indireto - não é um								-----	-----	
101	TMR0	Contador TMRO de 8 bits								xxxx xxxx	uuuu uuuu	
102	PCL	Parte baixa do PC (8 bits menos significativos)								0000 0000	0000 0000	
103	STATUS	IRP	RP1	RP0	/TO	/PD	Z	DC	C	0001 lxxx	000q quuu	
104	FSR	Ponteiro para endereçamento indireto								xxxx xxxx	uuuu uuuu	
105	-	Não Implementado								-	-	
106	PORTB	RB7	RB6	RB5	RB4	RB3	RB2		RB1	RB0	xxxx xxxx	uuuu uuuu
107	-	Não Implementado RBO								-	-	
108	-	Não Implementado								-	-	
109	-	Não Implementado								-	-	
10A	PCLATH	-	-	-	Parte alta do PC (5 bits mais significativos)					0000 0000	--0 0000	
10B	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u	
10C	EEDATA	Dado para escrita/leitura na E PROM (parte baixa)								xxxx xxxx	uuuu uuuu	
10D	EEADR	Endereço para escrita/leitura na E PROM (parte baixa)								xxxx xxxx	uuuu uuuu	
10E	EEDATH	-	-	Dado para escrita/leitura na E PROM (parte alta)						xxxx xxxx	uuuu uuuu	
10F	EEADRH	-	-	-	Endereço para escrita/leitura na E PROM (parte alta)					xxxx xxxx	uuuu uuuu	

Hex	Nome	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	BIT 1	Bit 0	Power-on	Outros Resets
BANK3											
180	INDF	Valor apontado pelo FSR (endereçamento indireto - não é um								-----	-----
181	OPTION_REG	/RBPU	INTED			PSA	PS2	PS1	PS0	1111 1111	1111 1111
182	PCL	Parte baixa do PC (8 bits menos significativos)								0000 0000	0000 0000
183	STATUS	IRP	RP1	RP0	/TO	/PD	Z	DC	C	0001 lxxx	0000 quuu
184	FSR	Ponteiro para endereçamento indireto								xxxx xxxx	uuuu uuuu
185	-	Não implementado								-	-
186	TRISB	RB7	RB6	RB5	RB4	RB3	RB2	RB1		1111 1111	11111111
187	-	Não Implementado RBO								-	-
188	-	Não implementado								-	-
189	-	Não implementado								-	-
18A	PCLATH	-	-	-	Parte alta do PC (5 bits mais significativos)					--0 0000	--0 0000
18B	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000000X	0000 000u
18C	EECON1	EEPG		-	-	WRE	WRE	WR	RD	0000 0000	0000 0000
18D	EECON2	Registrador de segurança para escrita na E PROM (não é um registrador implementado)								-----	-----
18E	-	Reservado, manter sempre zerado								0000 0000	0000 0000
18F	-	Reservado, manter sempre zerado								0000 0000	0000 0000

Legenda: x = desconhecido, u = não modificado, q = depende de outras condições, 1 = um, 0 = zero.

Set de Instruções Completo (14 bits)

ADDLW Soma uma literal ao W

Sintaxe: ADDLW k

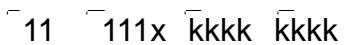
Descrição: O valor da literal passada no argumento k é somado ao valor de W e o resultado é armazenado no próprio W.

Limites: $0 \leq k \leq 255$

Operação: $(W) + (k) \rightarrow (W)$

Status afet.: C,DC,Z

Encoding:

 11 111x kkkk kkkk

Palavras: 1

Ciclos: 1

Exemplo: ADDLW 0x15

Antes da instrução:

W= 0x10

Após a instrução:

W= 0x25

ADWF Soma entre W e o registrador F

Sintaxe: ADWF 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:

 00 0111 dfff ffff

Palavras: 1

Ciclos: 1

Exemplo: ADWF 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.

Límites: $0 \leq k \leq 255$

Operação: (W) .AND. (k) → (W)

Statusafet.: Z

Encoding:

11 1001 kkkk kkkk

Palavras:

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.

Límites: $0 \leq f \leq 127$
 $d=0$ (W) ou $d=1$ (f)

Operação: (W) .AND. (f) → (d)

Statusafet.: 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).

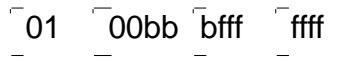
Limites: $0 \leq f \leq 127$

$0 \leq b \leq 7$

Operação: $0 \rightarrow (f,b)$

Status afet.: Nenhum

Encoding:



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: BTFSC 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 2 ciclos.

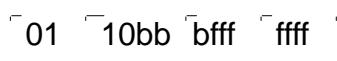
Limites: $0 \leq f \leq 127$

$0 \leq b \leq 7$

Operação: pula se $(f,b) = 0$

Status afet.: Nenhum

Encoding:



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

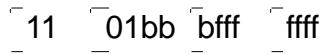
Descrição: O bit de número b do registrador f será selado (set).

Limites: $0 \leq f \leq 127$
 $0 \leq b \leq 7$

Operação: $1 \rightarrow (f,b)$

Statusafet.: Nenhum

Encoding:

 11 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 2 ciclos.

Limites: $0 \leq t \leq 127$
 $0 \leq b \leq 7$

Operação: pula se (f,b) = 1

Statusafet.: Nenhum

Encoding:

 01 11bb bfff ffff

Palavras: 1

Ciclos: 1 ou 2

Exemplo: BTFSS LED
GOTO ACENDEJ.ED
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 subrotma

Sintaxe: CALL k

Descrição: Chama a subrotina 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: $(PC) + 1 \rightarrow TOS$
 $(k) \rightarrow (PC, 10:0)$
 $(PCLATH,4:3) \rightarrow (PC,12:11)$

Status afet.: Nenhum

Encoding:

10 [Okkkk] [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: $0 \rightarrow (f)$
 $1 \rightarrow Z$

Statusafet.: Z

Encoding:

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 selado.

Limites: Nenhum

Operação: $0 \rightarrow (W)$
 $1 \rightarrow Z$

Status afet.: Z

Encoding

$\underline{00}$ $\underline{0001}$ $\underline{0000}$ $\underline{0011}$

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, rese-tando o valor do contador de prescaler. Os bits /TO e /PD são setados.

Limites: Nenhum

Operação: $0 \rightarrow (WDT)$
 0 → WDT prescaler
 1 → /TO
 1 → /PD

Statusafet.: /TO,/PD

Encoding:

$\underline{00}$ $\underline{0000}$ $\underline{0110}$ $\underline{0100}$

Palavras: 1

Ciclos: 1

Exemplo: CLRWDT

Antes da instrução:

WDT cont. = ?

WDT ps = ?

Após a instrução:

WDT cont. = 0

WDTp = 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:

00 1001 dfff ffff

Palavras: 1

Ciclos: 1

Exemplo: COMF REG1,F
Antes da instrução:

REG1 = 0x13

Após a instrução:
REG1= OxEC

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)

Statusafet.: Z

Encoding:

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 isto 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:

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# 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:

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:

00 1010 dfff ffff

Palavras: 1

Ciclos: 1

Exemplo: INCF CNT, F
Antes da instrução:

CNT = 0xFF
Z = 0

Após a instrução:

CNT = 0x00
FSR = Z

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 isto 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:


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 urna literal e W

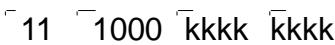
Sintaxe: IORLW k

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) → (W)

Status afet.: Z

Encoding:


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 dff 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 kk

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 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= ?

Após a instrução:

W= FSR

MOVWF Move o valor de W para F

Sintaxe: MOVWF f

Descrição: Move o valor do registrador W para o registrador f.

Limites: $0 \leq f \leq 127$

Operação: $(W) \rightarrow (f)$

Status afet.: Nenhum

Encoding:

 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:

 00 0000 0000 0000

Palavras: 1

Ciclos: 1

Exemplo: NOP

RETFIE Retorna da interrupção

Sintaxe: RETFIE

Descrição: Retorna da interrupção, recuperando o último endereço da pilha e selando obitGIE.

Limites: Nenhum Operação: TOS → (PC)

1-+GIE

Status afet.: Nenhum

Encoding:

 \u2014\u2014 00 \u2014\u2014\u2014\u2014 0000 \u2014\u2014\u2014\u2014 1001

Palavras: 1

Ciclos: 2

Exemplo: RETFIE

Após a instrução:

PC= TOS
GIE= 1

RETLW Retorna com uma literal em W

Sintaxe: RETLWk

Descrição: Retorna de uma subrotina, 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:

 \u2014\u2014 11 \u2014\u2014\u2014\u2014 01xx \u2014\u2014\u2014\u2014 kkkk \u2014\u2014\u2014\u2014 kkkk

Palavras: 1 Ciclos: 2

Exemplo: RETLW0x50

Após a instrução:

PC = TOS
W = 0x50

RETURN **Retorna de uma subrotina**

Sintaxe: RETURN

Descrição: Retorna de uma subrotina, recuperando o último endereço da pilha.

Limites: Nenhum

Operação: TOS → (PC)

Status afet.: Nenhum

Encoding:

 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 ≤ f ≤ 127
d=0 (W) ou d=1 (f)

Operação: Conforme descrição.

Status afet.: C

Encoding:

 00 1101 dff ffff

Palavras: 1

Ciclos: 1

Exemplo: RLF REG1.F

Antes da instrução:
C= 0
REG1= 11100110
Após a instrução:
C= 1
REG1= 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.

Status afet.: C

Encoding:

 00 1100 dfff ffff

Palavras: 1

Ciclos: 1

Exemplo: RRF REG1.F

Antes da instrução:
C= 0
REG1=11100110

Após a instrução:
C= 0
REG1= 01110011

SLEEP Entra em modo sleep

Sintaxe: SLEEP

Descrição: Coloca o microcontrolador em modo sleep.

Limites: Nenhum

Operação: 0 → WDT
0 → WDT prescaler
1 → /TO
0 → WPD

Status afet.: /TO, /PD

Encoding:

 00 0000 0110 0011

Palavras: 1

Ciclos: 1

Exemplo: SLEEP

SUBLW Subtração entre uma literal e W

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, PC, Z

Encoding:

11 110x kkkk kkkk

Palavras: 1

Ciclos: 1

Exemplo 1: 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 Subtração entre W e F

Sintaxe: SUBWF f,d

Descrição: Subtrai do registrador 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: $(f) - (W) \rightarrow (d)$

Statusafet.: C, PC, Z

Encoding:

 00 0010 dfff ffff

Palavras: 1

Ciclos: 1

Exemplo 1: 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)

Exemplo3: SUBLW REG.F

Antes da instrução:

REG= 1

W= 2

C= ?

Após a instrução:

REG = 255

W= 2

C= 0 (neg.)

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:

00 1110 dfff ffff

Palavras: 1

Ciclos: 1

Exemplo: SWAPF REG.F

Antes da instrução:

REG = 0xA5

Após a instrução:

REG = 0x5A

XORLW Operação "OU exclusivo" entre k-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:

11 1010 kkkk kkkk

Palavras: 1

Ciclos: 1

Exemplo: XORLW 0xAF

Antes da instrução:

W= 0xB5

Após a instrução:

W= 0x1A

XORWF Operação "OU" exclus. 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) → (d)

Statusafet.: 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

Anotações

Diretrizes da Linguagem MPASM

As diretrizes apresentadas neste apêndice são válidas para todos os PIC (12, 14 e 16 bits), exeto para a família 18.

BADRAM - Configura regiões da RAM não disponíveis

Sintaxe

_BADRAM <expr>[-<expr>[,<expr>-[<expr>]]]

Descrição

Utilizado para determinar 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 estes limites, uma mensagem de erro será gerada. O programador não precisa se preocupar com esta 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

Esta 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 RPO 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:O,END_H,END_L ;END=0X24, END_H=0X24,
                        END_L=0X25
    CÓDIGO:2        ;CÓDIGO=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

```

RESETcode   H'01FF'
            goto      START

```

Veja também

IDATA, UDATA, UDATAJDVR, 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 estas opções nos arquivos de INCLUDE de cada modelo de PIC. Basta dar uma conferida nestes arquivos para saber quais os símbolos pertinentes ao tipo de oscilador, WDT, etc. A <expr> deve ser montada pela junção destes símbolos através do operador & (AND).

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 TAMJTOTAL = TAM_NOME + TAM_CODIGO
```

Veja também

SET, VARIABLE

DA - Armazena uma string na memória de programa

Sintaxe

DA <expr> [, <expr2>, ... <exprn>]

Descrição

Preenche a memória de programa (14 bits) com o código ASCII reduzido (7 bits). Cada 2 caracteres da string são armazenados na mesma posição da memória de programa, sendo o primeiro na parte mais significativa e o segundo na menos significativa.

Exemplo

```
DA "abedef"  
Isto irá armazenar: 30E2 31E4 32E6 3380 na memória de programa.  
DA "12", 0  
Isto irá armazenar: 18B2 0000 na memória de programa.
```

DATA - Preenche a memória com números ou textos

Sintaxe

```
DATA <expr>[, <expr>, ..., <expr>]  
DATA "texto"[, "texto", ..., "texto"]
```

Descrição

Preenche a memória de programa com o valor determinado por <expr> ou "texto", começando na posição atual e avançando a quantidade 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 2 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 de programa 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 E2PROM 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 esta 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

Esta 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, esta 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

DT - Preenche a memória com uma tabela

Sintaxe

```
DT <expr> [<expr>,...,<expr>]  
DT "texto"
```

Descrição

Preenche a memória de programa com uma série de instruções RETLW, uma para cada <expr> ou caractere do "texto", começando na posição atual e avançando a quantidade 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 de programa com o valor determinado por <expr> ou "texto", começando na posição atual e avançando a quantidade 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 2 caracteres para cada posição, sendo o primeiro no byte mais significativo. Se o texto possui 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 condicionais

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. Esta 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

ENDIF - Fim dos blocos de teste condicional

Sintaxe

ENDIFⁱ

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/1/2/+- <msg_num>

Descrição

Determina que tipo de mensagens devem ser mostradas durante a compilação:

Nível "0" 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>."

Observação

Os números das mensagens, alertas e erros podem ser encontrados no apêndice C do manual do compilador MPASM. Nas versões mais novas do MPLab, esta diretriz pode ter seu nível acertado e gravado no arquivo de projeto.

Exemplo

ERRORLEVEL, -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 ela serão expandidas na listagem do LST.

Veja também

LIST, NOEXPAND, MACRO

EXTERN - Declara símbolos definidos externamente

Sintaxe

EXTERN <nome> [, <nome>]

Descrição

Usado para objetos. Declara símbolos que serão utilizados no código corrente mas que são definidos como GLOBAL em outros módulos.

Esta diretriz deve ser aplicada antes dos símbolos serem utilizados no código corrente. Mais de um símbolo pode ser definido na mesma linha, através 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 de programa 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 colocadas 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. Esta diretriz pode ser utilizada antes ou depois da definição do símbolo. Mais de um símbolo podem ser declarados na mesma linha através dos argumentos <nome>.

Exemplo

```
UDATA
VAR1 RES      .I
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 este 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. Esta diretriz não está disponível para PICs de 12 bits.

Exemplo

```
IDAT
LIMITEL     DW    .0
LIMITEH    DW    .300
GANHODW   .5
FLAGS DW    .0
TEXTODB    „MOSAICO”
```

Veja também

TEXT, UDATA, UDATA_OVR, UDATA_SHR, EXTERN, GLOBAL

IDLOCS - Especifica os dados para gravação do IP

Sintaxe

Conectando o PIC 16F877A - Recursos Avançados

IDLOCS <expr>

Descrição

Utilizado para especificar previamente os dados que serão gravados nas 4 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 esta diretriz e a ENDIF será executado. Caso contrário, ele será pulado, podendo ainda ser executado o bloco relacionado a 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 esta diretriz e a ENDIF será executado. Caso contrário, ele será pulado, podendo ser executado o bloco associado a diretriz ELSE, caso ela exista.

Exemplo

 #define TESTE

 IFDEF TESTE

(rotina que será executada para teste)
ENDIF

Veja também

IFNDEF, tfDEFINE, tfUNDEFINE, 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 esta diretriz e a ENDIF será executado. Caso contrário, ele será pulado, podendo ser executado o bloco associado a 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

IFNDEF, #DEFINE, tfUNDEFINE, ENDIF, ELSE

INCLUDE - Inclui ao programa um arquivo de definições

Sintaxe

INCLUDE <nome_do_arquivo> INCLUDE "nome_do_arquivo"

Descrição

Usado geralmente no início do programa, para incluir definições especificadas em arquivos auxiliares. Estes arquivos, com mesma formatação que os 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 destes 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 neste 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>

LIST - Opções para a listagem do programa

Sintaxe

LIST [<opção>, <opção>]

Conectando o PIC 16F877A - Recursos Avançados

Descrição

Configura uma série de opções que serão utilizadas para a criação da listagem de programa (arquivo com extensão 1ST). As opções possíveis encontram-se na tabela seguinte:

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.
p=<tipo>	Nenhu	Modelo de PIC (por exemplo: PIC16F84)
r=<radix>	H ^m ex	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.

Observação

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 esta 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

```
LOCALCOMP, LARG ; AS VARIÁVEIS LOCAIS NÃO GERAM  
CONFLITOS  
COMP SET .30  
LARG SET .50  
ENDM
```

;COMP=10 E LARG=20

Veja também
ENDM, MACRO

MACRO · Define uma Macro

Sintaxe

<nome> MACRO [<arg1>, ..., <argx>]

Descrição

Uma macro é uma sequência 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 a macro serão substituídos no corpo do código. Para terminar uma macro é necessário a diretiva ENDM. No entanto ela pode ser finalizada quando é encontrada a diretiva 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 RAM para o PIC que está sendo utilizado. A <expr> determina o último endereço de RAM disponível. O programador não precisa se preocupar com esta diretiva pois ela está definida nos arquivos de INCLUDE fornecidos pela Microchip.

Exemplo

```
; (para PIC 16F84)  
MAXRAM H'CF'  
BADRAMH'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 esta diretriz não serão expandidas na listagem do programa (1ST).

Veja também

LIST, EXPAND, MACRO

NOLIST - Desliga a geração do arquivo de listagem**Sintaxe**

NOLIST

Descrição

Inibe a geração automática do arquivo de listagem (1ST).

Veja também

LIST

ORG - Acerta ponto da memória de programação

Sintaxe

[<nome>] ORG <expr>

Descrição

Usado para determinar o ponto da memória de programação onde 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_10RG 0x10  
END_20RG 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 (1ST).

Veja também

LIST, TITLE, SUBTITLE, SPACE

PAGESEL - Seleciona a página de programação

Sintaxe

PAGESEL <nome>

Descrição

Esta 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 desta 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, esta 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<tip0>

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 que independentemente do RADIX padrão, qualquer número pode ser acertado através das pré--definições:

Radix	Formato 1	Formato 2
Decimal	D'xx'	.x
Hexadecimal	H'xx'	0Xxx
Octadecimal	O'xx'	
Binário	B'xxxxxxxx'	
ASCII	A'x'	'x'

Observação

Recomendamos que as pre-definições apresentadas acima sejam sempre utilizadas, para evitar problemas caso alguém altere o valor do radix em seus programas.

Nas versões mais novas do MPLab, esta diretriz não é mais necessária pois este valor é acertado e gravado no arquivo de projeto.

Exemplo

RADIX DEC

Veja também

LIST

RÉS - Reserva memória

Sintaxe

[<nome>] RES <tamanho>

Descrição

Reserva memória de programa de dados 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 RÉS 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
ÁREA SET COMP * LARG
VOL SET ÁREA * 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

```
SPACE3 ; 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 (1ST).

Exemplo

```
SUBTITLE " Desenvolvido pela MOSAICO ENGENHARIA"
```

Veja também

```
LIST, TITLE
```

TULE - 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 (1ST).

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 este argumento não seja especificado. Nenhum código será gerado nesta seção. A diretriz RÉS deve ser usada para reservar espaço para os dados.

Exemplo

```
UDATA  
VAR1 RÉS .1  
DOUBLE      RÉS .2
```

Veja também

```
TEXT, IDATA, UDATA_OVR, UDATA_SHR, EXTERN, GLOBAL
```

UDATA_OVR - Declara o início de uma seção de dados sobre carregados

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 UDATAJDVR. 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 sobre carregado por outras seções deste tipo que possuam o mesmo nome. Isto é ideal para declarar variáveis temporárias que devem ocupar o mesmo espaço na memória. Nenhum código será gerado nesta seção. A diretriz RÉS deve ser usada para reservar espaço para os dados.

Exemplo

```
TEMPS UDATA_OVR  
TEMP1  RÉS .1  
TEMP2  RÉS .1  
TEMP2  RÉS .1
```

```
TEMPS UDATA_OVR  
LONGTEMP1 RÉS .2  
LONGTEMP2 RÉS .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_SHR. 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. Isto 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 RÉS deve ser usada para reservar espaço para os dados.

Exemplo

```
TEMPS UDATA_SHR  
TEMP1 RÉS .1  
TEMP2 RÉS .1  
TEMP2 RÉS .1
```

Veja também

TEXT, IDATA, UDATA, EXTERN, GLOBAL, UDATA_SHR

UNDEFINE - Elimina uma substituição de texto

Sintaxe

```
#UNDEFINE  <nome>
```

Descrição

Elimina definição anteriormente criada pela diretriz tfDEFINE.

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 totalmente similar a 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 esta diretriz e a ENDW será executado. Caso contrário ele será pulado. Lembre-se que o valor 0 (zero) será considerado falso, enquanto qualquer outro número será considerado verdadeiro. O bloco poderá ter no máximo 100 linhas e poderá ser repetido até 256 vezes.

Exemplo

```
VARIABLE I  
CONSTANT CONTA = 5
```

1 = 0 WHILE I < CONTA
(bloco que será repetido 5 vezes)
I += 1 ENDW

Veja também

ENDW

Anotações

Instruções Especiais

A tabela abaixo 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 INCF	STATUS , C f,d
ADDDCF f , d	Se houve digit carry, incrementa o registrador f, guardando o resultado em d.	BTFSC INCF	STATUS, DC £,d
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 GOTO	STATUS , C k
BDC k	Pula (branch) para a posição definida por k, caso tenha ocorrido um digit carry.	BTFSC GOTO	STATUS, DC k
BNC k	Pula (branch) para a posição definida por k, caso não tenha ocorrido um carry.	BTFSS GOTO	STATUS, C k
BNDC k	Pula (branch) para a posição definida por k, caso não tenha ocorrido um digit carry.	BTFSS GOTO	STATUS, DC k
BNZ k	Pula (branch) para a posição definida por k, caso não tenha ocorrido um zero.	BTFSS GOTO	STATUS, Z k
BZ k	Pula (branch) para a posição definida por k, caso tenha ocorrido um zero.	BTFSC GOTO	STATUS, Z k
CLRC	Limpa o bit de carry.	BCF	STATUS , C
CLRDC	Limpa o bit de digit carry.	BCF	STATUS , DC
CLRZ	Limpa o bit de zero.	BCF	STATUS , Z
LCALL k	Chamada de rotina (k) distante. Acerta o PCLATH (bits 3 e 4) automaticamente.	BCF/BSF BCF/BSF CALL	PCLATH, 3 PCLATH, 4 k

instrução	Descrição	Instruções Equivalentes	Status
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.	MOVF 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 DECF f,d	z
SUBDCF f, d	Se houve digit carry, decrementa o registrador f, guardando o resultado em d.	BTFS C STATUS, DC DECF f,d	z
TSTF f	Testa o registrador f para saber se é zero.	MOVF f,F	z

Operadores do Compilador

Operado	Descrição	Exemplo
\$	Número da atual linha de programa (PC).	GOTO \$ + 3
(Abertura de parentesis.	1 + (R * 4)
)	Fechamento de parentesis.	(COMPR + 1) * 256
!	NÃO lógico.	IF ! (A - B)
-	Negativo.	-1 * COMPR
-	Complemento.	FLAGS = -FLAGS
high	Byte alto de um símbolo de dois bytes.	MOVLW high CTR_TABELA
low	Byte baixo de um símbolo de dois bytes.	MOVLW low CTR_TABELA
upper	Byte superior de um símbolo de três bytes.	MOVLW upper CTR_TABELA
*	bytes 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 ÍNDICE > NUM
>	^Maior que.	IF ÍNDICE > NUM
<	Menor que.	IF ÍNDICE < NUM
<=	Menor ou igual.	IF ÍNDICE <= NUM
==	Igual a.	IF ÍNDICE == NUM
!=	Diferente de (NÃO igual).	IF ÍNDICE != 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)

Operador	Descrição	Exemplo
=	Variável = valor.	ÍNDICE = 0
+=	Variável = ela mesma + valor.	ÍNDICE += 1
-=	Variável = ela mesma - valor.	ÍNDICE -= 1
* _	Variável = ela mesma * valor.	ÍNDICE *= TOTAL
/=	Variável = ela mesma / valor.	ÍNDICE /= TOTAL
%=	Variável = resto da divisão dela mesma novo valor	ÍNDICE %= 16
<=>	Variável = ela mesma relacionada à	ÍNDICE <=> 3
>=>	Variável = ela mesma rotacionada à direita.	ÍNDICE >=> 2
&=	Variável = ela mesma AND valor	ÍNDICE &= MASCARA
=	Variável = ela mesma IOR valor	ÍNDICE = MASCARA
^=	Variável = ela mesma XOR valor	ÍNDICE ^= MASCARA
++	Variável = ela mesma + 1 (incremento)	ÍNDICE ++
--	Variável = ela mesma - 1 (decremento)	ÍNDICE --



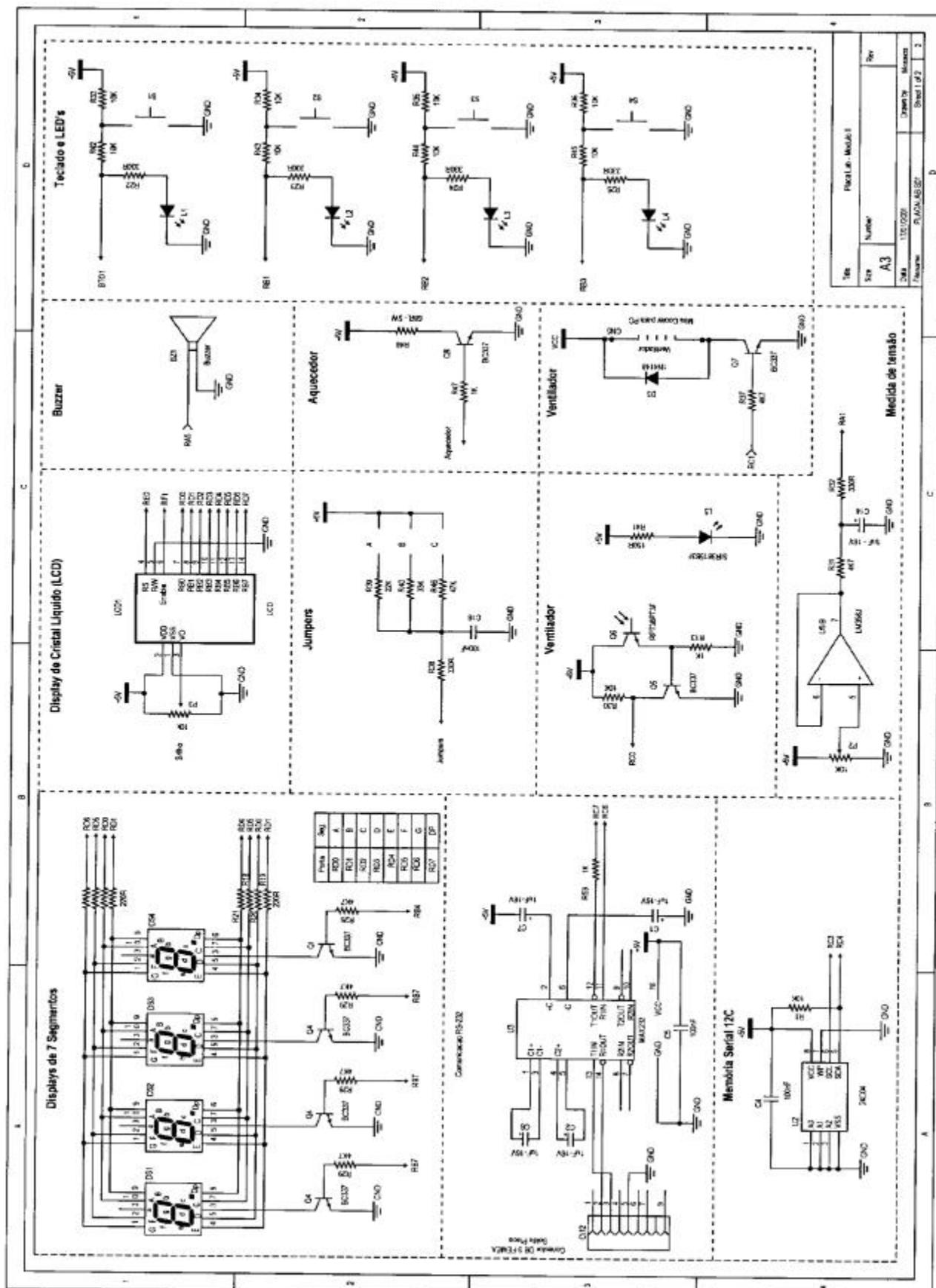
Esquema Elétrico da Placa Proposta (McLab2)

Sugerimos aqui um hardware para a realização dos treinamentos e projetos presentes neste livro. Com o esquema elétrico apresentado nas próximas páginas, será possível a montagem de uma placa capaz de realizar todas as experiências deste livro e muitos outros experimentos com o PIC 16F877A (ou qualquer outro de 40 pinos).

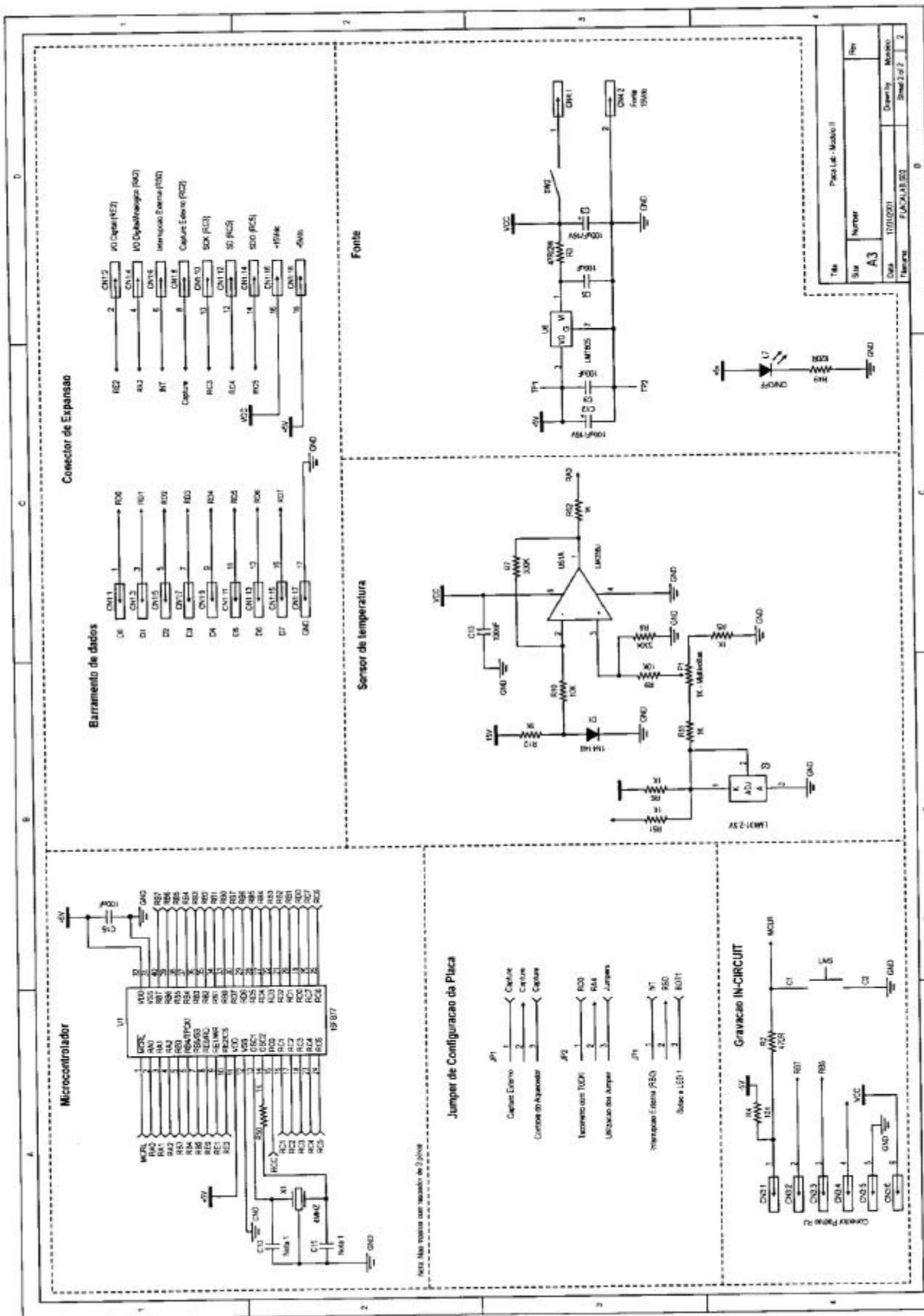
Sua placa possuirá então os seguintes recursos:

- Soquete para montagem do PIC de 40 pinos;
- 4 displays de 7 segmentos com controle por varredura;
- Display de cristal líquido (LCD) com 2 linhas e 16 colunas;
- 4 teclas e 4 Leds para entradas e saídas;
- Buzzer e jumpers;
- Memória E²PROM serial externa do tipo 24C04 (ou outra similar);
- Comunicação RS-232;
- Conector para gravação In-Circuit,
- Sensor de temperatura;
- Aquecedor via resistor de potência;
- Miniventilador para resfriamento da resistência ou experiências com sensor óptico;
- Sensor (transmissor e receptor) óptico do tipo infravermelho;
- Conector de expansão.

Informamos que para facilitar ainda mais a sua vida, este esquema é o mesmo de um produto denominado McLab2 (Mosaico), que pode ser facilmente encontrado no mercado, evitando que você seja obrigado a montar o hardware proposto para poder executar todos os exercícios apresentados neste livro.



Conectando o PIC16F877A - Recursos Avançados



Conectando o PIC16F877A - Recursos Avançados

Anotações

índice Remissivo

/PD, 274
/To, 274

A

A/D, 34, 117, 137
Acknowledge, 209
ACON0.125
ADCON1,47,50,125
ADDEN, 255
ADIE, 125
ADIF,37,125
ADON, 125
ADRESH,125

CKE, 207, 220
Clear, 41
Clock, 205
externo, 26

interno, 25

Compare, 34,151
Conversor analógico, 117
CREN, 255
Cristal,276
CS,50

D

D/A, 220
DC161,159

B

BANKSEL.31
Baud Rate, 247, 255
BCLIE, 37
BF, 206, 220

Destino, 41
Dufy cyc/e, 156,282

E

Bit, 41
BRGH.255
Bronw-out, 274
Brown-out Reset, 275
Buzzer,58

E²PROM, 27,33,175
EEADR.180
EEADRH, 180
EECON, 180
EEDATH, 180
EEIE, 37

C

CAD.117
CALL, 28, 29
Capture, 34,151,152
CCP, 34

EEIF, 180
EEPGD.33,180
Escravo, 202

F

CCP1CON.154
CCP1IE.36
CCP2IE'37
CCPR1L, 159
CGRAM, 99

FERR, 255
File, 41
FLASH, 27,175,178

FSR, 31
Fu//Dup/ex,247

G

GIE,36,37,125,153,207
GO/DONE, 125
GOTO, 28

H

Ha/f Duptex, 249

I

PIC,,201,214
In-Circuit, 279
INDF,31
INTCON, 36,125,153,180, 207, 220, 2 26, 255
Interrupções, 27

L

Led 58
Literai 41

M

Máquina, 26
Master, 201,205,255
Memória de
 dados, 30
de programa, 27
MSSP, 34,201

O

OERR.255
OPTION_REG, 47, 274
OPTION_REG<TOSE>; 50
Oscilador, 276
Overíow, 273

p

P, 220
Paralela, 271
Paridade, 249
PCF, 50
PCFGO, 47
PCFG1.47
PCFG2, 47
PCFG3,47

PEIE, 125,153,207, 255
PIE1.36,125
Pilha, 29
Pipeline, 27
PIR1.125.220
PQR, 274
PORTA, 23, 39, 45, 47
PORTB, 23, 39
PORTC, 23,39,48,153
PORTD, 24, 39
PORTE, 45

Postscale, 35, 54,158
Power-down Mode, 275
Prescaler, 35, 51,53,74,157
PSA, 52, 274
PSP, 34, 271
PSPMODE, 50
Pull-up, 47
PWM, 34,151,155,282
PWRT, 274

R

R/W, 220
RAO, 47
RA1.47
RA2, 47
RA3, 47
RA4, 47
RA5, 47
RAM, 30

RB4, 34
RB5, 34
RB6, 34
RB7, 34
RC, 137
RC0, 48
RC1.48
RC2, 48
RC3, 48
RC4, 48
RC5, 48
RC6, 48
RC7, 48
RCIE, 36, 255
RCIF, 37

SSPIE, 36
SSPIF, 207
SSPMO, 220
SSPM1.207
SSPM2, 207
SSPOV, 220
SSPSTAT, 207,
Start, 208
Start Bit, 248
STATUS, 274, 276
STATUS<IRP>, 31
STATUS<RP1:RP0>, 31
Stop, 208
Bit, 248

RCSTA, 255
RD.50,180
Registradoras especiais, 30
Reset, 55
Ressonador, 276
Re-Start, 209
RS-232, 282
RW,50
RX, 247

T
T0CKI, 50
T0CS, 47
T0IE, 52
T0IF, 36
T0SE, 52
Teste, 41
TMRO, 33, 50
TMR1.74

S

Schmitt Trigger, 57
Set, 41
 de instrução
SFRs, 32
Skip, 41
Slave, 201,205,256
SLEEP, 153,159,254,274,275
SMP, 220
SPI, 34
SSPADD,220
SSPBUF, 206, 207, 220
SSPCON, 207, 220
SSPCON2, 220
SSPEN, 206

TMR1IE.37
TMR2IE
Trigger, 46
TRISA.47,125
TRISC, 153
TRISE, 125
TX, 247
TX9D
TXIE, 255
TXIF, 37, 255

u

USART, 282

v

Vetor de Interrupção, 27 Vetor de Reset, 27

w

Wake-up, 274 WCOL, 207, 220 WDT, 273 Work, 41 WR, 180 WREN, 180 WRERR, 180

z

Zero, 41

Referências Bibliográficas

- SOUZA. D. J. Desbravando o PIC. São Paulo: Érica, 2000.
- MICROCHIP Technology. DataSheet PIC16F877A. USA, 2001.
- _____. MPASM Assembler User's Guide. USA, 2001.
- _____. MPLAB User's Guide V6.22. USA, 2003.
- _____. Home page (Internet) e Technical Library (CD-ROM). USA, 2003.
- _____. Product Line Card. Catálogo.USA, 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.



Microcontroladores HCS08 - Teoria e Prática
Autor: Fábio Pereira • Código: 0980 • 208 páginas • Formato: 17 x 24 cm

Com uma linha didática e prática, o livro apresenta noções básicas dos microcontroladores HCS08, o hardware utilizado, uma visão geral de sua arquitetura, algumas características da CPU e detalhes relativos à sua família. Utilizando o Codewarrior, ensina a escrever diversos exemplos de aplicação para as placas de demonstração utilizadas no livro, desde a base, passando pela simulação do programa, o download do código para o chip e a execução do programa diretamente na placa.

Apresenta instruções Assembly, detalhes sobre os periféricos e módulos internos, características do compilador C Integrado ao ambiente Codewarrior e exemplos de aplicação para facilitar o aprendizado.



Microcontrolador 8051 com linguagem C - Prático e Didático - Família AT89S8252 Atmel
Autores: Denys E. C. Nicolosi e Rodrigo B. Bronzeri • Código: 0794 • 224 páginas • Formato: 17 x 24 cm

A proposta do livro é ensinar a linguagem C para os microcontroladores da família 8051, mais especificamente com o ATTEL AT89S8252, largamente utilizado no Brasil e no mundo, gerando material didático teórico que abrange também o compilador SDCC (freeware).

Apresenta detalhes do hardware, experiências práticas do chip desenvolvidas em linguagem C (display 7 segmentos, display alfanumérico, interface serial com PC, timer, interrupção, DA, AD e comunicação I2C), além de detalhes sobre o Hyperterminal do Windows, displays alfanuméricos e instalação do compilador SDCC.



Programação C para Microcontroladores 8051
Autor: Maurício Cardoso de Sá • Código: 0778 • 336 páginas • Formato: 17,5 x 24,5 cm

Com uma linguagem prática descreve a arquitetura interna da família de microcontroladores Intel 8051/52, passando pelo compilador Keil com destaque para a linguagem C.

Possui exemplos práticos de programação dos periféricos internos, como: pisca LEDs, manipulação de teclas e teclados matriciais, comunicação serial, exibição de dados e varredura de displays de 7 segmentos, displays LCD caractere e tratamento e programação das diversas interrupções dos integrados da família 8051.

É indicado aos profissionais ou hobbistas que tenham a eletrônica como base, e também interesse em conhecer essa família de microcontroladores e aplicações embarcadas (embedded).



Microcontroladores MSP430 - Teoria e Prática
Autor: Fábio Pereira • Código: 00670 • 416 páginas • Formato: 17,5 x 24,5 cm

Aborda a maioria dos periféricos disponíveis nas famílias 1xx, 2xx e 4xx de microcontroladores MSP430 da Texas Instruments, além da arquitetura interna da CPU de 16 bits, modos de funcionamento e instruções Assembly.

O ambiente de programação Embedded Workbench da IAR é estudado, assim como as técnicas de simulação e depuração disponíveis nos chips. Apresenta as características particulares do compilador C IAR, além de uma breve revisão da linguagem C no padrão ANSI.

Possui diversos exemplos de configuração e utilização dos periféricos internos e um capítulo dedicado à aplicação utilizando módulos LCD de caractere e gráfico, utilização de integrados SPI, teclados, comunicação serial, etc.



Controlador Digital de Sinais - Família 56F800/E - Baseado no MC56F8013 - Microarquitetura e Prática
Autores: José Carlos de Souza Junior e Renato Rodrigues Paixão • Código: 0697 • 352 páginas • Formato: 17,5 x 24,5 cm

A arquitetura interna da família 56F800/E, seu set de instruções e as principais características técnicas são estudados neste livro.

Baseado no modelo MC56F8013, com exemplos de uso dos periféricos "on-chip" (PWM, ADC, Timers, SCI, IFC, SPI, GPIO's, entre outros), ilustra o embasamento teórico e prático, destacando a ferramenta de trabalho e o ambiente de programação CodeWarrior®, orientação para as etapas de um projeto e o kit de programação DEMO56F8013. Ensina também a escrever códigos nas linguagens Assembly e C, e a desenvolver um projeto completo a partir da ferramenta Processor Expert.

É indicado a estudantes e profissionais da área que possuam conhecimentos básicos de eletrônica analógica, digital e tenham noções de programação.

OUTROS LIVROS DA ÁREA

Microcontroladores



Microcontroladores PIC16F628A/648A - Uma abordagem prática e objetiva
Autor: Wagner da Silva Zanco • Código: 059X • 368 páginas • Formato: 17,5 x 24,5 cm

Fornecê um conteúdo didático, que possibilita desenvolver aplicações em linguagem assembly para os PIC16F628A/648A, utilizando todos os recursos desses microcontroladores. A primeira parte explica como estruturar programas em linguagem assembly, simular, debugar e gravar o programa na memória do microcontrolador. A segunda parte aborda os periféricos, cada um em um capítulo exclusivo, seguido por um programa exemplo, desenvolvido sob a ótica da programação estruturada, que possibilita comprovar, em circuito, o funcionamento do periférico estudado.

É destinado a estudantes ou profissionais que desejam estudar microcontroladores ou se aprofundar no assunto.



Internet Embedded - TCP/IP para Microcontroladores
Autores: Marcos P. Mokarzel e Karina P. Mokarzel Carneiro • 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 I^C. 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 EEPROM externa através do barramento serial I^C, 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 M68EV908Q: pisca-pisca com LED, controle PWM de brilho com leitura analógica, temporizador com display LED multiplexado e um voltímetro digital com display LCD.

Aplicações Práticas do Microcontrolador 8051

Autor: Vidal Pereira da Silva Jr. • Código: 9395 • 248 páginas • 17 x 24 cm

Conectando o PIC - Recursos Avançados

Autores: David José de Souza e Nicolás César Lavinia • Código: 7376 • 384 páginas • 17 x 24 cm

Desbravando o PIC - Ampliado e Atualizado para PIC 16F628A

Autor: David José de Souza • Código: 8674 • 272 páginas • 17 x 24 cm

Laboratório de Microcontroladores - Família 8051 - Treino de Instruções, Hardware e Software

Autor: Denys E. C. Nicolosi • Código: 8712 • 208 páginas • 17 x 24 cm

Microcontrolador 8051 - Detalhado

Autor: Denys E. Campion Nicolosi • Código: 721x • 256 páginas • 17 x 24 cm

Microcontroladores PIC - Programação em C

Autor: Fábio Pereira • Código: 9352 • 360 páginas • 17 x 24 cm

Microcontroladores PIC - Técnicas Avançadas

Autor: Fábio Pereira • Código: 7279 • 360 páginas • 17 x 24 cm

OUTROS LIVROS DA ÁREA



Microcontroladores

Conectando o PIC16F877A - Recursos Avançados



EDITORAS ÉRICA

www.editoraerica.com.br