Programando Microcontroladores PIC

Linguagem “C”

Renato A. Silva

Programando Microcontroladores PIC

Linguagem “C”

Ensino Profissional
EDITORA

Novembro de 2006

Produção: Ensino Profissional Editora
Editor Responssável: Fábio Luiz Dias
Organização: Julianna Alves Dias
Design da Capa: Renato A. Silva
Diagramação : Renato A. Silva
Correção ortográfica: Ligia Vaner da Silva

Direitos reservados por:
Jubela Livros Ltda.
Rua Maestro Bortolucci, _9_
Vila Albertina - São Paulo - SP
Cep: 0__57-0_0
Telefone: (__) 6_0_ - 6__8
Fax: (__) 6_6_ - _9__

S586p Silva, Renato A.
Programando microcontroladores PIC : Linguagem “C” / Renato A. Silva. – São Paulo : Ensino Profissional, 2006.
_7_p.

_. Microcontroladores. _. Microprocessadores. _. C (Linguagem de Programação de Computadores). I. Título.

CDU: 68_.__

Catalogação na publicação por: Onélia Silva Guimarães CRB-14/071

E-mail da Editora: editor@ensinoprofissional.com.br
Homepage: www.ensinoprofissional.com.br
Atendimento ao Consumidor: suporte@ensinoprofissional.com.br

Contato com o Autor:
autor.renato@ensinoprofissional.com.br
http://www.renato.silva.nom.br

"Quanto ao mais irmãos, fortaleci-vos no Senhor, pelo seu soberano poder. Revesti-vos da armadura de Deus para que possais resistir às ciladas do Demônio. Porque nós não temos que lutar contra a carne e o sangue, mas contra os Principados, Potestades, contra os Dominadores deste mundo tenebroso, contra os espíritos malignos espalhados pelos ares. Tomai portanto, a armadura de Deus para que possais resistir no dia mau, e ficar de pé depois de terdes cumprido todo o vosso dever. Ficai firmes, tendo os vossos rins congidos com a verdade, revestidos com a couraça da justiça, e os pés calçados, prontos para ir anunciar o Evangelho da Paz."

São Paulo, Efe 6,_0-_5

Sumário

Prefácio

Na atualidade uma diversidade de microcontroladores esta presente no mercado exigindo a efetiva busca por atualização, para fins aplicativos operacionais e/ou didáticos existe uma procura por aperfeiçoamento numa programação mais fácil. Cada dia se faz mais necessário um conjunto de instruções que não sofra variações bruscas e relevantes entre os microcontroladores. Logo, a sua aprendizagem deve possibilitar o entendimento dos demais. Com base nisto, o conteúdo da presente obra vem produzir uma documentação para uso como instrumento de aplicação pedagógica e operacional nos mais variados ambientes, bem como em desenvolver competências no âmbito das aplicações de microcontroladores e motivar desenvolvedores a projetar, desenvolver e implementar sistemas microcontrolados de pequeno e médio porte.

O conteúdo deste livro é apresenta a fundamentação teórica sobre o microcontrolador PIC _6F6_7 e _6F6_8, realiza experimentações práticas com esses microcontroladores e segue um tipo de metodologia cujo objetivo é permitir ao desenvolvedor a familiaridade com componentes eletrônicos, a montagem em matrizes de contato e posterior análise, testes e programação dos circuitos propostos.

Mais uma vez, este livro dedica-se ao aprendizado da tecnologia de automação e robótica, utilizando microcontroladores para executar tarefas específicas. No decorrer do livro o leitor terá a oportunidade de inteirar-se da tecnologia dos microcontroladores da família PIC da Microchip®, iniciando no módulo básico com o uso de transistores passando pelas portas lógicas e avançando passo-a-passo até os microcontroladores, onde aprenderá a fazer softwares em linguagem assembler e posteriormente utilizando a linguagem C.

Finalmente, cabe ao leitor sempre, o esforço para aprender a programar microcontroladores e usa-los com criatividade e imaginação para o desenvolvimento de novos projetos. Aqui reforçamos o pedido do autor no sentido de ter uma boa dose de paciência no aprendizado e não desistir frente às dificuldades, pois com certeza, é uma caminhada de enriquecimento de conhecimentos. E para aqueles que felizmente encontra-se em um degrau mais elevado, espera-se que a obra venha somar algo mais a sua carreira.

Antonio Ilídio Reginaldo da Silva
Diretor – Escola Senai Catalão - GO

Apresentação

Este livro, dedica-se ao aprendizado da programação de microcontroladores utilizando-se da linguagem “C”, de forma prática, conduzindo o leitor a um aprendizado gradativo ao uso dos microcontroladores para executar tarefas específicas. No decorrer do livro o leitor terá a oportunidade de inteirar-se da tecnologia dos microcontroladores da família PIC® MCU da Microchip de forma teórica e prática. Com o auxílio da placa experimental proposta, cujo layout pode ser baixado gratuítamente da internet no endereço http://renato.silva.nom.br e com a realização dos exercícios complementares, o leitor encontrará em condições de desenvolver aplicações de controle de microcontroladores PIC, utilizando-se da linguagem de programação “C”.

A visão da obra consiste em apresentar o microcontrolador com os seus recursos e a medida que forem utilizados, indicar a forma de programação. Inicialmente é apresentado a forma de programação em assembler de uma aplicação simples, e os meios de gravação no microcontrolador. Posteriormente a programação seguirá em linguagem “C”, a medida que for sendo utilizado os recursos do microcontrolador e de alguns periféricos.

As ferramentas aqui utilizadas são livres para uso com execessão do compilador CCS PIC C Compiler da CCS Inc. cujas informações adcionais e sobre aquisição, podem ser adquiridas diretamente do fabricante no endereço http://www.ccsinfo.com/

Introdução:

O desenvolvimento atual da tecnologia nas áreas de automação e robótica deve-se principalmente ao desenvolvimento dos microcontroladores e processadores digitais de sinais (DSP), tendo estes memórias e estrutura que lembra os microcomputadores atuais, executando um software escrito para uma determinada finalidade, sendo extremamente robustos, baratos e confiáveis. Dentre os diversos fabricantes, encontramos os microcontroladores da Microchip®, uma empresa norte americana, fundada em 1989, com sede na cidade de Chandler, Arizona (oeste dos E.U.A.) que fabrica os microcontroladores da família PIC, uma das mais variadas do mercado, tendo eles, uma filosofia de produto em comum, característica que permite a compatibilidade de software e a estruturação das aplicações, pois um código escrito para um modelo de PIC poderá ser migrado para outro modelo sem que sejam necessárias grandes mudanças no código fonte. Isto facilita o trabalho de quem desenvolve e preserva o investimento de quem produz.

Os microcontroladores PIC, reúne em um único chip todos os circuitos necessários para o desenvolvimento de um sistema digital programável, dispondo internamente de uma CPU (Unidade central de processamento) que controla todas as funções realizadas pelo sistema, tendo em seu interior diversos registradores e a Unidade Lógica Aritmética (ALU) onde são executadas todas as funções matemáticas e lógicas, basicamente toda movimentação de dados passa através da ALU. A CPU conta também com memória de programa PROM (Memória programável somente de leitura), memória RAM (memória de acesso randômico) e registrador W (uso geral) dentre outros.

Capítulo

1

História do transistor e do microchip

1.1 O Transistor

Antes de “PICarmos” um pouco, faz-se necessário uma pequena visão do desenvolvimento, iniciando no final dos anos 40, com a construção do primeiro transistor nos laboratórios da BELL em 23 de dezembro de 1947 por John Bardeen, Walter Houser Brattain, e William Bradford Shockley, os quais ganharam o prêmio Nobel de física 1956. O transistor é a contração das palavras transfer resistor, resistência de transferência. É um dispositivo eletrônico semicondutor, componente chave em toda a eletrônica moderna, de onde é amplamente utilizado formando parte de computadores, portas lógicas memórias e uma infinidade de circuitos. Este revolucionário engenho transformou o mundo em pouco tempo.

Cientistas em diversos laboratórios estavam à procura de um componente que substituísse as válvulas e reles antigos. Diversos materiais foram submetidos a testes físico-químicos e classificados em dois grupos, os condutores e os não condutores, isolantes ou dielétricos. Alguns materiais não se enquadravam em nenhum dos dois grupos, ora conduziam ora isolavam, foi então classificado como semicondutor.

Em 1945, por iniciativa de Mervin Kelly, então diretor da Bell Labs, formou-se um grupo de pesquisa para o estudo dos semi-

condutores. Um ano mais tarde o grupo já estava quase formado. William Bradford Shockley, físico do MIT (instituto de pesquisa de Massasshussets), John Bardeen, engenheiro elétrico e Walter Houser Brattain, físico.

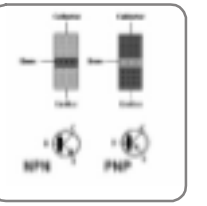
Dois anos mais tarde e depois de muito trabalho, os cientistas conseguiram em 16 de dezembro de 1947 construir o transistor primordial construído com duas folhas de ouro prensados em um cristal de germânio e com uma carga elétrica aplicada, esta fluía entre o cristal, obtendo o efeito de amplificação tão desejado. A Bell Labs. Promoveu uma ampla difusão de informações a fim de incentivar outros a pesquisarem, e fabricarem o transistor, chegando a ponto de vender a patente do transistor por apenas U$ 25.000,00. O objetivo era que outras empresas alavancassem o desenvolvimento de novas tecnologias que pudessem ser usadas em telecomunicações, sua área de atuação.

Anos mais tarde, uma dessas companhias, a Texas Instruments, anunciou o primeiro transistor de silício, material, que apresentava inúmeras vantagens sobre o germânio, uma delas era a abundante matéria prima o que reduziria os custos de fabricação. A Texas tornou-se assim uma poderosa concorrente no mercado.

O transistor é um dispositivo semicondutor de estado sólido, ele foi assim chamado pela propriedade de trocar a resistência pela corrente elétrica entre o emissor e o coletor. É um sanduíche de diferentes materiais semicondutores em quantidade e disposição diferentes intercalados. Pode-se obter assim transistores PNP e NPN. Estas três partes são: Uma que emite elétrons (emissor) uma outra que recebe e os coleta (coletor) e uma terceira (base) que está intercalada entre as duas primeiras, regula a quantidade desses elétrons. Um pequeno sinal elétrico aplicado entre a base e o emissor modula a corrente que circula entre o emissor e coletor. O sinal base emissor por ser muito pequeno em comparação com o emissor base. A corrente emissor coletor é aproximadamente

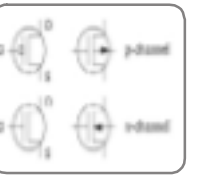
da mesma forma que a da base emissor, mas amplificada por um fator de amplificação, chamado “Beta”. Todo transitor tem um fator beta sendo assim amplificador, pode também se usado para oscilar, para retificar, para comutar, sendo esta a principal função do transistor na eletrônica digital.

1.3 Tipos de transistor



Existem diversos tipos de transistores, mais a classificação mais acertada consiste em dividi-los em transistores bipolares e transistor de efeito de campo FET, sendo está dividida em JFET, MOSFET, MISFET, etc. A diferença básica entre os diversos tipos de transistores está na forma em que controla o fluxo de corrente. Nos transistores bipolares que possuem uma baixa impedância de entrada, onde o controle se exerce injetando uma baixa corrente (corrente de base), ao passo que o transistor de efeito de campo que possui uma alta impedância, tendo o controle através de tensão (tensão de gate).

Os transistores de efeito de campo FET mais conhecidos são os JFET (Junction Field Effect Transistor), MOSFET (Metal-Oxide-Semiconductor FET) e MISFET (Metal-Insulator-Semiconductor FET). Este tem três terminais denominados gate, dreno, supridouro. O gate regula a corrente no dreno, fornecida no supridouro.



Indiscutivelmente com o surgimento do transistor os equipamentos eletrônicos ficaram mais leves, compactos e passaram a consumir menos energia, tendo estas características cada vez mais relevância na indústria.

1.4 O Nascimento do Microchip

Apenas treze anos após a invenção do transistor, houve outro grande salto tecnológico, a invenção do circuito integrado ou microchip, por Jack S. Kilby da Texas Instruments e Robert N. Noyce da Fairchild Semicondutor.

No verão de 1958, Jack S. Kilby, entrou para a equipe da Texas Instruments, em Dallas, E.U.A, onde desenvolveu o primeiro microchip da história, usando componentes ativos (transistores, diodos) e passivos (resistores, capacitores) em uma única peça de material semicondutor do tamanho de um clips de papel. O sucesso da demonstração de laboratório foi o nascimento do primeiro microchip simples da história, em 12 de Dezembro de 1958.

Paralelamente na Fairchild Semicondutor, Robert N. Noyce avançava em suas pesquisas, inspirado nas técnicas de mascaramento que empregavam dióxido de silício para a difusão de impurezas, e utilizando-se de trilhas de ouro ou alumínio aplicado com ajuda de máscara e fotolitográfia, enquanto o processo de Kilby empregava pequenos fios nas ligações internas do circuito.

A tecnologia evoluiu, e também o número de empresas e em 1962 nasceu à lógica TTL, e anos mais tarde a tecnologia MOS (metal-oxide semiconductor), seguida pela CMOS (complementary metal-oxide semiconductor), tecnologia atual hoje em dia. Com a tecnologia CMOS e a crescente miniaturização do microchip, surgiu em 1974 o primeiro microprocessador da história denominado “1802” fabricado pela RCA, seguido pelo microprocessador de 4 bits da Texas Instruments, o TMS1000.

A eletrônica digital baseia-se em tensão (definido como 1) e ausência de tensão (definido como 0) onde o nível alto ou 1 é de 5 volts ou 2/3 da fonte e o nível 0 é zero volt ou um pouco acima, portanto, para a eletrônica digital somente importa o nível alto ou baixo. Com esta ótica as indústria desenvolveram blocos com a união de centenas de transistores para realização de uma tarefa específica denominando-se circuito integrado.

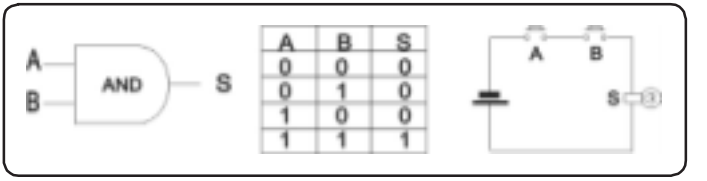
Capítulo 2

Portas Lógicas, Números Binários e Hexadecimais

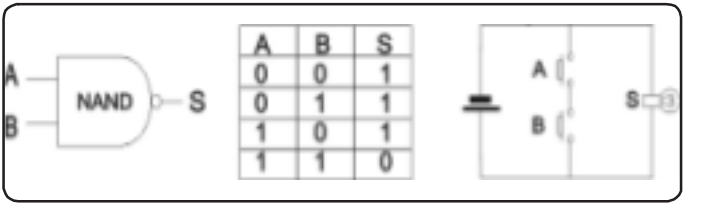
2.0- As Portas Lógicas

As portas lógicas formam a base da eletrônica digital iniciando um novo compêndio na eletrônica e o seu estudo é essencial ao entendimento e aprendizado dos microcontroladores e microprocessadores. Ao agrupar circuitos ativos e passivos em blocos, os cientistas criaram blocos que executavam uma determinada função lógica. Estas funções são AND (E), NAND (NÃO E), OR (OU), XOR (OU EXCLUSIVO), XNOR (NÃO EXCLUSIVO) e NO (NÃO). Para trabalharmos com as portas lógicas faz-se o uso de uma tabela verdade, a qual cada função tem a sua.

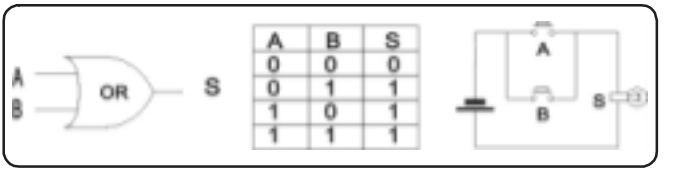
Nas portas AND têm-se nível alto (tensão) em sua saída somente se todas as suas entradas, tiverem nível alto, obtendo assim um "E" em suas entradas, pôr exemplo entrada à "E" entrada b. Dizemos assim, que a saída é o resultado da entrada "A", "E" da entrada "B".



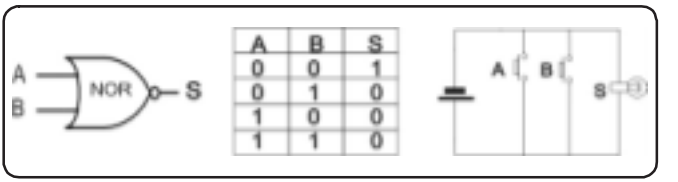
As portas NAND, ao contrário das portas AND somente terá



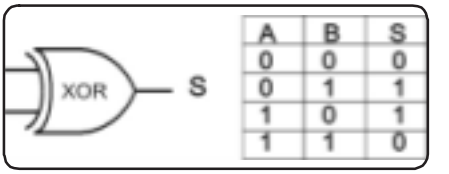
nível alto em sua saída se as suas entradas tiverem nível baixo. As portas lógicas do tipo OR terão nível alto em sua saída se uma "OU" outra entrada tiver nível alto.



As portas lógicas tipo NOR, ao contrário da função OR, somente terá nível alto ou 1 em sua saída se as entradas forem zero. Observe a diferença básica entre a função OR e a NOR.

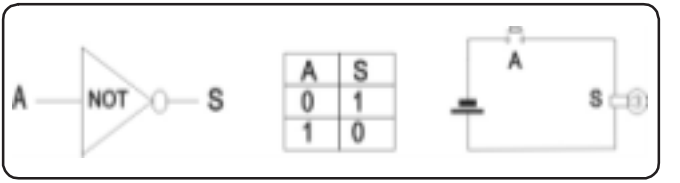


As portas lógicas XOR, garantem o nível alto em sua saída se uma entrada tiver em nível alto e outra em nível baixo. Observe que diferentemente da AND se tivermos 1 e 1 a saída será 0.



As portas XNOR apresentam princípio semelhante à função XOR, apenas com o detalhe de ter saídas invertidas.

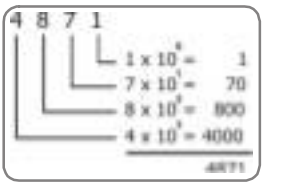
As portas lógicas NO, são as mais fácil de todas, apresenta nível alto em sua saída se sua entrada for nível baixo, invertendo assim os níveis lógicos.



Como já sabemos o nível alto (normalmente 5 volts) é representados pôr 1 e o nível baixo (zero volts) pôr 0, assim a combinação de 1 e 0 em grupos de 8, formão um conjunto denominado "byte". Veremos agora as várias representações de conjuntos numéricos e as suas formas de conversão.

2.1- Números Decimais

Desde cedo aprendemos a raciocinar com números decimais, onde o conjunto matemático contém 10 elementos [0..9], este sistema de numeração baseia-se em potencia de 10 onde cada dígito corresponde ao número 10 (base) elevado a uma potência (expoente) de acordo com sua posição.



2.1.1- Números Binários

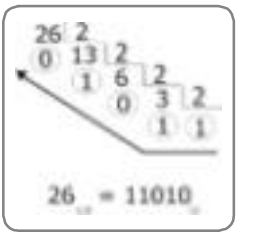
Da mesma forma os números que os números decimais, os números binários, são assim chamados porque o seu conjunto contém apenas 2 elementos [0,1]. Este conjunto numérico representa os estados lógicos 0 e 1. Ao organizarmos os bits 0 e 1 em grupos de 8 temos um byte de oito bits, em grupos de 16 temos um byte de 16 bits e assim sucessivamente.

A escrita do número binário sempre é feita da direita para a esquerda, dizemos que a parta da direita é a mais significativa ou MSB (most significative bit) e a parte da esquerda a menos significativa ou LSB (louse significative bit), dai devemos elevar 2 ao expoente da casa correspondente ao 1 do bit. Observe na tabela de expoentes da figura que o primeiro expoente da direita é 1, portanto $2^0 = 1$ e o quarto expoente da direita para a esquerda é o 3, portanto $2^3 = 8$ agora fazemos a soma de todos os resultados, neste caso 1 + 8 = 9.

Tudo é uma questão de prática, ao praticar um pouco você entenderá melhor esta relação.

Você já deve ter percebido, aqui, que existe uma relação, uma forma de representar o 0 e o 5 volts dos circuitos digitais em números, desta forma, podemos “conversar” facilmente com as máquinas digitais utili-

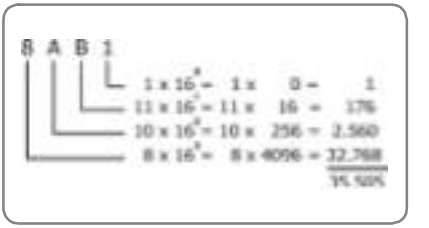
zando números e realizar operações com eles. Para convertemos uma representação decimal em binária, fazemos sucessivas divisões por 2 e anotamos os resultados. Depois ordenamos de forma lógica da direita para a esquerda



2.1.2- Número Hexadecimal

A numeração hexadecimal que como as anteriores tem seu conjunto matemático representado por 16 números, facilita e acelera a decodificação de dados, economizando espaço em armazenamento de dados. Neste conjunto temos 16 números sendo [0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F]. Utilizamos basicamente o conjunto de números decimais (com10 elementos) e lançamos mãos das letras (A,B,C,D,E,F) para complementar o conjunto atribuindo-lhe valor. Desta forma o número A passa a ter o valor 11 e o F o valor 16. O número 17 decimal é igual a 11 hexadecimal e o número 23 decimal é igual ao número 17 hexadecimal. Parece um pouco confuso mas não é, acompanhe no apêndice a tabela de números decimais x binário x hexadecimal. Assim o número 65535 (5 casas) em decimal, ficaria como 11111111.11111111 (16 casas) em binário e FFFF (4 casas) em hexadecimal.

A conversão Hexadecimal para decimal multiplica o valor do digito pela potência de 16, fazendo a somatória. Ex: 8AB1 = 35.505



Para convertermos hexadecimal em binário, a forma mais prática é termos em mente a tabela de números de A até F em binário e depois agrupar os dígitos binários, veja 1A fica com 1 (primeiro dígito) e 1010 da tabela. Outro exemplo 3C convertido dá 11(referente ao 3) e 1100(referente ao C).

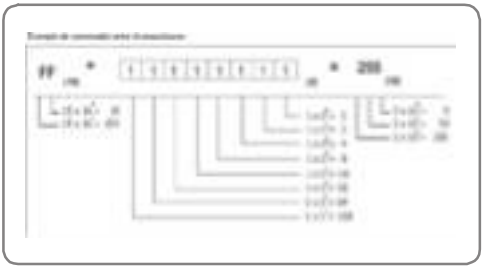
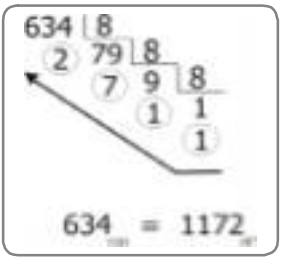


Tabela hexadecimal x binária A - F

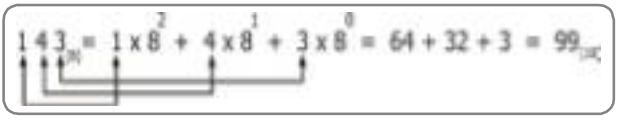
A	B	C	D	E	F
1010	1011	1100	1101	1110	1111

2.1.3- Numeração Octal

O sistema de numeração octal é um conjunto matemático onde temos oito elementos [0,1,2,3,4,5,6,7] que apresenta grande importância principalmente pela facilidade de conversão de binário para octal. A metodologia de conversão é semelhante as anteriores, para conversão de numeração decimal para numeração octal, faz-se divisões sucessivas por 8 até encontrar o menor quociente e posteriormente pegamos o resto da divisão em ordem inversa.



Para conversão de octal para decimal, multiplicamos o numeral por 8 elevado a potencia correspondente.



A Conversão de numeração octal para numeração binária faz-se uso da tabela octal onde temos o correspondente binário de cada número octal, depois agrupamos os bytes correspondentes ao octal da direita para a esquerda.

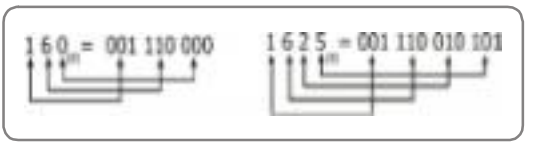


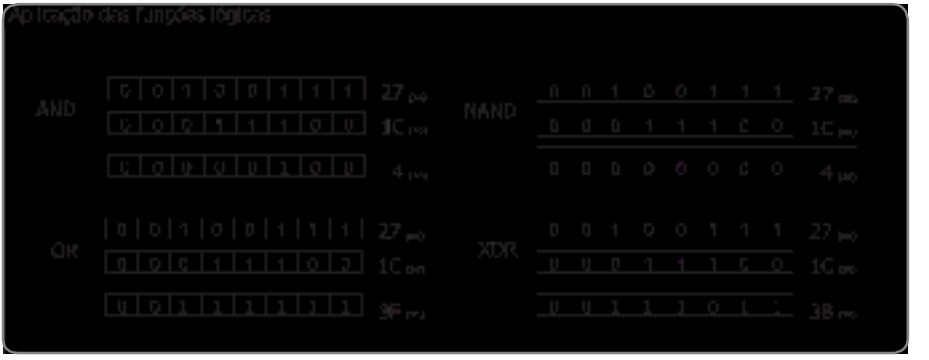
Tabela de conversão entre diversas bases

Decimal	Hexadecimal	Binário	Octal
0	0	0000	000
1	1	0001	001
2	2	0010	010

Decimal	Hexadecimal	Binário	Octal
3	3	0011	011
4	4	0100	100
5	5	0101	101
6	6	0110	110
7	7	0111	111
8	8	1000	
9	9	1001	
10	A	1010	
11	B	1011	
12	C	1100	
13	D	1101	
14	E	1110	
15	F	1111	

As funções lógicas booleanas são utilizadas quase que na totalidade das aplicações com microcontroladores. No capítulo anterior o leitor viu as diversas funções (portas lógicas), veja agora exemplo de aplicação destas funções em numeração binário e hexadecimal.

Neste capítulo, tivemos uma visão geral da relação, conversão e utilização dos diversos conjuntos numéricos e sua indiscutível e relevante importância para os sistemas computacionais. No próximo capítulo veremos os diversos tipos de memórias e a suas utilizações.



Capítulo

3

Memórias e Microcontroladores

3.1 Memórias

A memória é a capacidade de reter, recuperar, armazenar e evocar informações disponíveis, neste caso em forma binária de 0 e 1. Transistores, portas lógicas e flip-flops são utilizados na implementação de memórias eletrônicas, sendo as principais tecnologias definidas como memórias ROM, PROM, EPROM, EEPROM, FLASH e RAM.

Memórias ROM, memória somente de leitura, é uma memória onde os dados gravados não podem ser modificados ou cancelados. Os dados são gravados na fabricação através de uma máscara colocada sobre o chip de modo a registrar nas células disponíveis as informações desejadas.

Memórias PROM, memória somente de leitura programável é aquela onde os dados podem ser inseridos pôr meio de gravadores específicos uma única vez. Cada célula deve armazenar um bit de byte, como um fusível onde o 1 seria o fusível intacto e o 0 fusível rompido, é preciso muito cuidado ao programar este tipo de memória, pois se um bit ou um byte for gravado errado, não há como corrigir. A gravação é feita utilizando uma tensão chamada VPP, cuja amplitude depende da memória, normalmente 25 volts.

Memórias EPROM, memória somente de leitura apagável, esta sem dúvida é bem mais comum dos que a PROM, pode ser apagada se exposta à luz ultravioleta através de uma janelinha de quartzo (transparente a radiações ultravioletas) e programada novamente

pôr meio de gravadores específicos, pôr muitas vezes.

Memórias EEPROM, memória somente de leitura apagável eletricamente, pode ser apagada eletricamente e regravadas milhares de vezes, utilizando gravadores específicos, não precisando expô-la a radiação ultravioleta, sendo muitas vezes menores que as EPROM, já que não tem a janelinha de quartzo.

Memória FLASH, parecida em tudo com as do tipo EEPROM, podendo ser apaga e gravada eletricamente e gravada novamente, podendo ter até 100.000 ciclos de apagamentos

Memória RAM, Constituída de transistores ou flip-flop, podem armazenar dados somente quando tiverem tensão, quando não houver tensão estarão zeradas, sendo voláteis, tem seu uso no processo auxiliar ao processador armazenando informações temporárias. Basicamente são constituídas de dois tipos: estáticas e dinâmicas.

As memórias RAM estáticas, muito utilizada no final dos anos 80, tinham como principal desvantagem o tamanho, pois cada chip podia armazenar pequena quantidade de bytes, algo em torno de 1k, 4k byte. Portanto para criar um banco de memória com certa capacidade, fazia-se necessário projetar uma placa de proporções considerável.

As memórias RAM dinâmicas, ao contrário tinham alta densidade podendo armazenar por chip 1 megabyte facilmente, porém estas memórias necessitam de refresh constante e conseqüentemente circuitos de apoio. São estas empregadas hoje em dia nos computadores, normalmente tem entre 8 e 10 transistores por byte e um tempo de acesso da ordem de 7 nanosegundos !

3.2 Microcontrolador

Microcontrolador é um circuito integrado programável que contém todos os componentes de um computador como CPU (unidade central de processamento), memória para armazenar programas, memória de trabalho, portas de entrada e saídas para comunicar-se com o mundo exterior, sistema de controle de

tempo interno e externo, conversores analógico digital, uart de comunicação e outros.

Pode-se controlar qualquer coisa ou estar incluído em unidades de controle para:

- máquinas pneumáticas, hidráulicas comandadas
- máquinas dispensadoras de produtos
- motores, temporizadores
- sistemas autônomos de controle, incêndio, umidade temperatura
- telefonia, automóveis, medicina, ...etc

Estamos rodeados por máquinas que realizam algum trabalho ajudado por sensores e atuadores que recolhem as informações.

3.3 História dos microcontroladores

Em 1965 a GI Microelectronics, deu seus primeiros passos, fabricando memórias EPROM e EEPROM, desenhou no início dos anos 70 o microprocessador de 16 bits CP1600. Este trabalhava bem, mas de forma ineficaz , no controle de portas de entrada / saída. Para resolver este problema em 1975 desenhou-se um chip destinado a controlar portas de entrada / saída. Nascia assim o PIC (Peripherical Interface Controler), Com estrutura muito mais simples que um processador, este podia manejar as entradas e saídas com muita facilidade, rapidez e eficiência.

Uma das razões do sucesso do PIC é à base de sua utilização, ou seja, quando se aprende a trabalhar com um modelo, fica fácil migrar para outros modelos já que todos têm uma estrutura parecida.

Um dos grandes fabricantes de microcontroladores é a Microchip® que tem como fábrica principal em Chandler, Arizona, onde são fabricados e testados os últimos lançamentos. Em 1993 foi construída outra fábrica em Tempe, Arizona, que também conta com centros de fabricação em Taiwan e Tailândia. Para se Ter uma idéia de sua produção, só na família 16CSX, é de aproximadamente 1 milhão de unidades semanais.

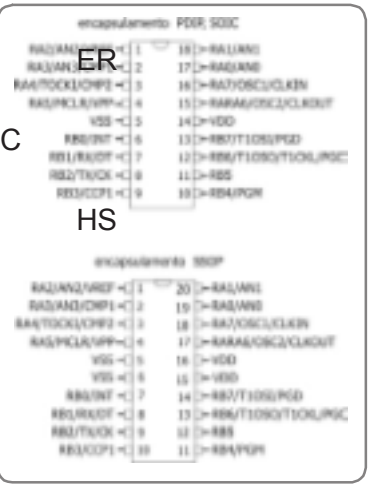
Cada tipo de microcontrolador serve para um propósito e cabe ao projetista selecionar o melhor microcontrolador para o seu trabalho. Dentre os mais populares, encontramos o 16F84, ainda fácil de se obter, mas está sendo substituído pelo modelo 16F627 ou 16F628 por ter mais recursos e preço aproximado. A diferença entre o PIC 16F627 e o PIC 16F628 está na quantidade de memória.

3.4 Apresentando o PIC 16F62x

O microcontrolador PIC 16F62x, reúne em uma pastilha todos os elementos de uma CPU RISC de alta performance, sendo fabricado em encapsulamento DIP (18 Pinos), SOIC (18 pinos) ou SSOP (20 pinos). Onde encontramos:

- Set (conjunto) de instruções com 35 funções
- 200 nanosegundos por instrução @ 20 Mhz.
- Clock de até 20 Mhz.
- 1024 bytes de memória de programa (16F627)
- 2048 bytes de memória de programa (16F628)
- Stack com 8 níveis
- 16 registradores especiais de funções
- Capacidade de interrupção
- 16 portas de entrada / saídas independente
- Alta corrente de saída, suficiente para acender um LED
- Comparador analógico.
- Timer0 de 8 bits com prescaler, postscaler
- Timer1 de 16 bits com possibilidade de uso de cristal externo
- Timer2 de 8 bits com registro de período, prescaler e postscaler
- Captura com 16 bits e resolução máxima de 12,5 nS.
- Comparação com 16 bits e resolução máxima de 200 nS.
- PWM com resolução máxima de 10 bits.
- USART para comunicação serial
- 16 bytes de memória RAM comum

- Power On Reset (POR)
- Power Up Timer (PWRT)
- Oscillator start-up (OST)
- Brow-out Detect (BOD)
- Watchdog Timer (WDT)
- MCLR multiplexado
- Resistor pull-up programáveis no PORTB
- Proteção de código programável
- Programação em baixa voltagem
- Power save SLEEP
- Oscilador
- Resistor externo – ER
- Resistor interno - INTRC
- Clock externo – EC
- Cristal alta velocidade - XT
- Cristal baixa velocidade – HS
- Cristal – LP
- Programação “in-circuit”
- Baixo consumo
- < 2.0 mA @ 5.0V, 4 Mhz.
- 15 uA @ 3.0V, 32 Khz.
- < 1 uA em repouso @ 3.0V.
- Quatro localizações para ID de usuário



3.5 Alimentação

Normalmente o PIC é alimentado com uma tensão 5,0 volts provenientes de uma fonte DC com regulação positiva, um regulador 7805 ou 78L05 podem ser utilizados para tal função, lembrando aqui que o 78L05 tem capacidade de suprir até 100ma, não devendo exceder 80% deste consumo pois aumenta muito o aquecimento do regulador. O consumo de corrente do microcontrolador é mínimo, consumindo menos de 2,0 mA com 5 volts, trabalhando a 4Mhz ou 15,0 micro amp com 3 volts, trabalhando

a 32 Khz. Quando em modo StandBy, consome menos de 1,0 micro amp. com 3.0V, porém devemos ver o consumos dos outros componentes do circuito.

- PIC 16F62x - 3.0V a 5.5V
- PIC 16LF62x - 2.0V a 5.5V

Em suas portas de saídas temos uma corrente de aproximadamente 10 mA, o suficiente para iluminar um led diretamente, no entanto, sempre devemos utilizar um resistor em série na porta normalmente de 10k. Respeitando suas características o PIC trabalhará sem problemas com grande estabilidade durante muito tempo sem necessidade de manutenção.

3.6 Definição da CPU

As CPUs dependendo do tipo de instruções que utilizam podem ser classificadas em:

- CISC: (Complex Instruction Set Computer) processadores

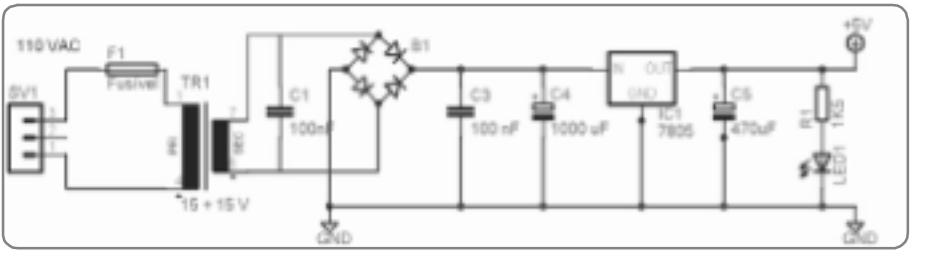
com conjunto de instruções complexas, dispõe de um conjunto com elevado número de instruções algumas sofisticadas e potentes. Em contrapartida requerem muitos ciclos de máquina para executar as instruções complexas.

- RISC: (Reduced Instruction Set Computer) processadores

com conjunto de instruções reduzidos, em nosso caso são 35 instruções simples que executam em 1 ou 2 ciclos de máquina com estrutura pipeline onde todas as instruções executam na mesma velocidade.

- SISC.(Specific Instruction Set Computer) processadores

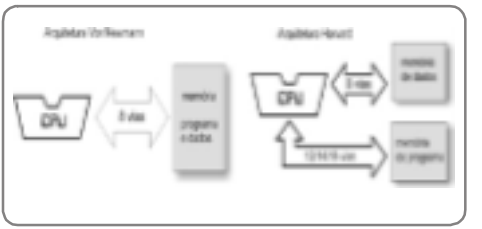
com conjunto de instruções específicas.



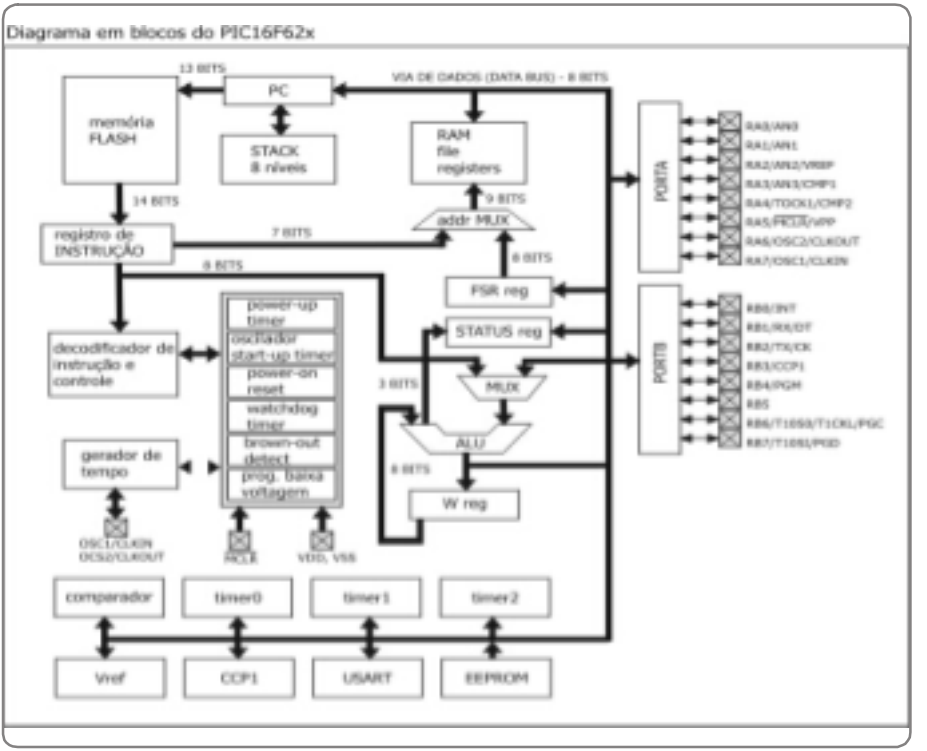
3.7 Arquitetura Interna

Entende-se por arquitetura interna a forma como o circuito é construído, representada por blocos, isto é como suas partes internas se interligam, podemos definir os PICs como sendo “Arquitetura Harvard”, onde a CPU é interligada à memória de dados (RAM) e a memória de programa (EPROM) por um barramento específico.

Tradicionalmente os microprocessadores têm como base à estrutura de Von Neumann, que se caracteriza por dispor de uma única memória principal em que se armazenam dados e instruções. O acesso à memória é feito através de um sistema de uma única via (bus de dados, instruções e de controle).



A arquitetura interna do PIC é do modelo Harvard,



onde dispõe de memórias de dados e de programas. Cada memória dispõe de seu respectivo bus, o que permite, que a CPU possa acessar de forma independente a memória de dados e a de instruções. Como as vias (bus) são independentes estes podem ter conteúdos distintos na mesma direção. A separação da memória de dados da memória de programa faz com que as instruções possam ser representadas por palavras de mais que 8 bits, assim o PIC, usa 14 bits para cada instrução, o que permite que todas as instruções ocupem uma só palavra de instrução, sua arquitetura ortogonal, onde qualquer instrução pode utilizar qualquer elemento da arquitetura como fonte ou destino.

Todo o processo baseia-se em banco de registros onde todos os elementos do sistema como, temporizadores, portas de entrada/saída, posições de memórias, etc, estão implementados fisicamente como registros. O manejo do banco de registros, que participam ativamente na execução das instruções, é muito interessante ao ser ortogonal.

3.8 Características do PIC 16F62x

Conforme vemos no diagrama em blocos do PIC 16F627 e 16F628, podemos ressaltar as seguintes características:

memória de programa EEPROM de 1Kb x 14 bits no 16F627.

memória de programa EEPROM de 2Kb x 14 bits no 16F628.

Memória de dados EEPROM de 64 bytes.

memória de dados RAM com 224 bytes dividida em 4 bancos.

Registro de propósito específico (SFR) com 32 posições.

Registro de propósito geral (GPR) com 224 posições.

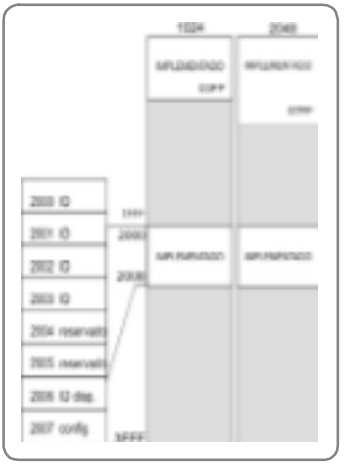
ALU de 8 bits e registro de trabalho W que normalmente recebe um operando que pode ser qualquer registro, memória, porta de entrada/saída ou o próprio código de instrução.

Pilha (Stack) de 8 níveis.

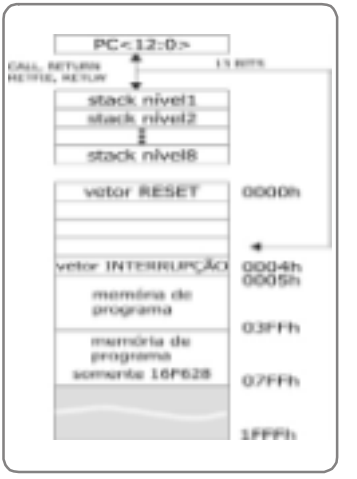
Contador de programa (PC) de 13 bits (o que permite endereçar até 8 KB de memória).

Recursos conectados al bus de dados:
PortA de 8 bits <RA0:RA7>
PortB de 8 bits <RB0:RB7>
Temporizadores / contadores TMR0, TMR1, TMR2
Comparadores
Captura , Comparação e PWM
Voltagem de referencia
USART para comunicação serial
Memória EEPROM

3.9 Organização da Memória



O PIC contém um registrador denominado PC (Program Counter) que é implementado com 13 bits, capaz de endereçar até 8K de programa, mas que somente 1k é implementado fisicamente no 16F627 (0000h à 03FF) e 2K são implementados no 16F628 (0000H à 07FFh). Este registrador ligado diretamente a "Pilha" (stack) armazena o endereço contém o endereço da instrução que vai ser executada. Ao incrementar ou alterar o conteúdo do PC, o microcontrolador tem um mapa seguro de onde esta e para onde ir.



3.10 A Pilha ou Stack

A pilha é uma memória independente da memória de programa e da memória de dados, com estrutura LIFO (Last In First Out) último dado a entrar

será o primeiro dado a sair com oito níveis de profundidade ou armazenamento com 13 bits cada um, sua função é guardar o valor do PC quando ocorre um salto do programa principal para o endereço de um subprograma a ser executado, fazendo assim com que o microcontrolador tenha total controle as chamadas de rotinas. Seu funcionamento é como um buffer circular onde o endereço da última chamada é o primeiro a retornar em uma chamada RETUR, RETLW ou RETIE, como não há nenhuma flag indicando o transbordamento da pilha se houver uma chamada de rotina após outra coma mais de oito níveis a primeira será sobrescrita, impossibilitando o retorno correto do programa causando um erro fantástico.

3.11 Organização da memória de dados

A memória de dados divide-se em quatro bancos, contendo os registros de propósitos gerais (GPR), registros de funções especiais (FSR). A seleção do banco de memória é feita através dos bits RP1 (STATUS <6>) e RP0 (STATUS <5>) conforme a seguinte tabela:

RP1, RP0	Banco	Endereço
0 0	0	000h - 07Fh
0 1	1	080h - 0FFh
1 0	2	100h - 17Fh
1 1	3	180h - 1FFh

Em cada banco temos 7F posições de memórias (128 bytes) o que nos dá 512 bytes, no entanto existe uma lacuna nos bancos onde temos endereços não implementados, assim para memória RAM, ou melhor, para os registro de propósito geral (GPR) temos 224 bytes. Os Registros especiais (FSR) ocupam os primeiros 32 bytes de cada banco e são utilizados pela CPU e pelos módulos periféricos para controlar o funcionamento do dispositivo, onde alguns destes registros especiais estão duplicados nos 4 bancos

para reduzir o código e tornar o acesso mais rápido.

Os registros que afetam a CPU são: STATUS, OPTION, INTCON, PIE1, PIR e PCON. Veremos agora descrição destes e outros registros.

3.12 Registradores de Funções Especiais

Registro de STATUS endereços 03h, 83h,103h e 183h, contém o estado da Unidade Lógica Aritmética ALU (C, DC, Z), estado de RESET (TO, PD) e os bits para seleção do banco de memória (IRP, RP1, RP0).

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
IRP	RP1	RP0	#TO	#PD	Z	DC	C

bit	descrição
IRP	Registrador de seleção de banco 1 = banco 2 e 3 (100h-1FFh) 0 = banco 0 e 1 (00h - FFh)
RP1,RP0	Registrador de seleção de banco RAM

RP1,RP0	banco	localização
00	banco 0	00h à 7Fh
01	banco 1	80h à FFh
10	banco 2	100h à 17Fh
11	banco 3	180h à 1FFFh

TO	TimeOut - bit somente leitura 1 = depois de power-up, instrução CLRWDT ou SLEEP 0 = ocorreu estouro de WDT
PD	Power Down - bit somente para leitura 1 = depois de power-up ou CLRWDT 0 = execução da instrução SLEEP
Z	Zero 1 = resultado da operação aritmética ou lógica é zero 0 = resultado da operação aritmética ou lógica não é zero
DC	Digit Carry - Transporte de dígito 1 = um valor menor é subtraído de um valor maior 0 = um valor maior é subtraído de um menor

bit	descrição
IRP	Registrador de seleção de banco 1 = banco 2 e 3 (100h-1FFh) 0 = banco 0 e 1 (00h - FFh)
C	Carry - Transporte 1 = um valor mais pequeno é subtraído de um valor maior 0 = um valor maior é subtraído de um menor

Registro OPTION ou OPTION_REG endereço 81h e 181h

É um registro de leitura e escrita que contém vários bits de controle para configurar o funcionamento do prescaler ao timer0 e ao WDT, interrupção externa ao timer0 e os resistores de pull-up do PORTB.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RBPU	INTED	TOCS	TOSE	PSA	PS2	PS1	PS0

bit	descrição
RBPU	habilitação dos resistores pull-up no portB 1 = resistências de pull-up desligadas 0 = resistências de pull-up ligadas
INTDEG	Interrupção no pino RB0/INT 1 = interrupção ligada, sensível à descida do pulso 0 = interrupção desligada, sensível à subida do pulso
TOCS	seleção da fonte de clock para o timer 0 TMR0 1 = tmr0 atua como contador por transição de RA4/TOCKL
TOSE	fonte de pulso para o timer 0 TMR0 1 = incrementa pulso alto para baixo (descendente) 0 = incrementa pulso de baixo para alto (ascendente)
PS2, PS1, PS0	divisor de freqüência (prescaler)

BIT 2, 1, 0	TMR0	WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
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

Registro INTCON - endereço 0Bh, 8Bh, l0Bh e 18Bh

É um registro de leitura e escrita que contém os bits de habilitação das várias interrupções (exceto do comparador), inclusive por mudança de estado no PORTB.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

bit	descrição
GIE	interrupção global 1 = habilita todas as interrupções 0 = desabilita todas as interrupções
PEIE	interrupções de periféricos 1 = habilita todas as interrupções de periféricos 0 = desabilita todas as interrupções de periféricos
TOIE	interrupção por transbordamento (over_ow) de timer0 TMR0 1 = habilita interrupção para TMR0 0 = desabilita interrupção para TMR0
INTE	interrupção externa 1 = habilita interrupção externa de RB0/INT 0 = desabilita interrupção externa de RB0/INT
RBIE	interrupção por mudança de estado no portB 1 = habilita interrupção para o portB 0 = desabilita interrupção para o portB
TOIF	_ag de over_ow para o TMR0 1 = ocorreu estou em TMR0 - deve ser zerado via soft 0 = não ocorreu estouro
INTF	ocorrência de interrupção externa em RB0/INT 1 = ocorreu interrupção 0 = não ocorreu interrupção
RBIF	ocorrência de estado no portB 1 = ocorreu mudança de estado em RB7:RB4 0 = não ocorreu mudança de estado em RB7:RB4

Registro PIE1 - endereço 8Ch.

Este registro contém os bits individuais de interrupções dos periféricos, sendo que para seu funcionamento o bit PEIE (INTCON <6>) deve estar habilitado.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
EEIE	CMIE	RCIE	TXIE	-	CCP1IE	TMR2IE	TMR1IE

bit	descrição
EEIE	_ag de interrupção de escrita na EEPROM completada 1 = habilita interrupção de término de escrita 0 = desabilita interrupção de término de escrita
CMIE	interrupção do comparador habilitada 1 = habilita interrupção do comparador 0 = desabilita interrupção do comparador
RCIE	interrupção de recebimento de caracter no USART 1 = habilita interrupção de recebimento do USART 0 = desabilita interrupção de recebimento do USART
TXIE	interrupção de envio de caractere no buffer do USART 1 = habilita a interrupção de transmissão do USART 0 = desabilita a interrupção de transmissão do USART
CCP1IE	interrupção do CCP1 para captura, comparação e PWM 1 = interrupção habilitada 0 = interrupção desabilitada
TMR2IE	interrupção 1 = habilita interrupção 0 = desabilita interrupção
TMR1IE	_ag de over_ow 1 = habilita interrupção para estouro de TMR1 0 = desabilita interrupção de estouro de TMR1

Registro PIRI – endereço 0Ch

Este registro contém as flags individuais que indicam as interrupções provocadas por periféricos.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
EEIF	CMIF	RCIF	TXIF	-	CCP1IF	TMR2IF	TMR1IF

bit	descrição
EEIF	_ag de interrupção de escrita na EEPROM 1 = operação de escrita na eeprom terminou. limpar via soft 0 = operação de escrita na eeprom não terminou
CMIF	_ag de interrupção do comparador 1 = houve mudança na saída do comparador 0 = não houve mudança na saída do comparador
RCIF	_ag de interrupção de recebimento no USART 1 = buffer de recebimento está cheio 0 = buffer de recebimento está vazio
TXIF	_ag de interrupção de transmissão do USART 1 = buffer de transmissão está vazio 0 = buffer de transmissão está cheio

bit	descrição
CCPIF	_ag de captura, comparação e PWM (não aplicado) modo captura: 1 = ocorreu captura; 0 = não ocorreu captura modo comparação: 1 = ocorreu comparação; 0 = não ocorreu comp.
TMR2IF	_ag de interrupção do TMR2 1 = houve ocorrência entre TMR2 e PR2 0 = não houve ocorrência
TMR1IF	_ag de interrupção do TMR1 1 = ocorreu estou em TMR1 - deve ser limpo via soft 0 = não ocorreu estouro no TMR1

Registro PCON - endereço 0Ch

Este registro contém as flags que permitem diferenciar entre um Power-on Reset (POP), um Brown-out Reset (BOD), um Reset por Watchdog (WDT) e um Reset externo por MCLR.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
-	-	-	-	OSCF	-	POR	BOD

bit	descrição
OSCF	freqüência do oscilador INTRC/ER 1 = 4 Mhz. 0 = 32 Khz.
POR	status Power-on-reset 1 = não ocorreu 0 = ocorreu (após ocorrência deve ser setado)
BOD	status do Brown-out 1 = não ocorreu reset brown-out 0 = ocorreu reset (após ocorrência deve ser setado)

Os demais registros especiais relacionados com o funcionamento dos periféricos serão utilizados junto dos periféricos.

3.13 Palavra de configuração e identificação:

É um registrador com 14 bits que se escreve durante o processo de gravação do dispositivo que deve estar de acordo com o esquema / sistema em que será utilizado, pois determina a forma do oscilador, se usará reset entre outros. Ocupa a posição reservada de memória 2007h.

13	12	11	10	9	8	7
CP1	CP0	CP1	CP0	-	CPD	LVP

6	5	4	3	2	1	0
BODEN	MCLRE	FOSC2	PWRTE	WDTE	FOSC1	FOSC0

<table>
<tr><td>bit</td><td>descrição</td></tr>
<tr><td>CP1, CP0</td><td>
<table>
<tr><td>bit 13,12,11,10</td><td>para 2K de mem.</td><td>para 1k de mem.</td></tr>
<tr><td>11</td><td>desativado</td><td>desativado</td></tr>
<tr><td>10</td><td>0400h - 07FFh</td><td>desativado</td></tr>
<tr><td>01</td><td>0200h - 07FFh</td><td>0200h - 03FFh</td></tr>
<tr><td>00</td><td>0000h - 07FFh</td><td>0000h - 03FFh</td></tr>
</table>
</td></tr>
<tr><td>CPD</td><td>bit de proteção de código
1 = desabilitada proteção da memória de código
0 = proteção habilitada</td></tr>
<tr><td>LVP</td><td>habilitação do modo de programação em baixa voltagem
1 = habilitado em Rb.4
0 = desabilitado e Rb.4 usado como entrada/saída</td></tr>
<tr><td>BODEN</td><td>habilitação do reset por detecção Brown-out
1 = reset habilitado
0 = reset desabilitado</td></tr>
<tr><td>MCLRE</td><td>seleção do Master Clear Reset
1 = reset habilitado em Ra.4
0 = reset desabilitado e Ra.4 como entrada</td></tr>
<tr><td>PWRTEN</td><td>termpo de estabilização - POWER-UP
1 = desabilitado
0 = habilitado</td></tr>
<tr><td>WDTE</td><td>habilitação do WDT - WatchDog Timer
1 = habilitado
0 = desabilitado</td></tr>
</table>

bit	descrição
FOSC2 FOSC1 FOSC0	

bits 4, 1, 0	descrição
000	LP cristal com baixa freqüência
001	XT cristal/ressonador <= 4Mhz.
010	HS cristal alta freqüência > 4 Mhz
011	EC clock em Ra.7 e I/O em Ra.6
100	INTRC resistor/capacitor interno com I/O em Ra.7 e Ra.6
101	INTRC resistor/capacitor com I/O em Ra.7
110	ER resistor externo em Ra.7 - I/O em Ra.6
111	ER resistor externo em Ra.7 clkout em Ra.6

3.14 Portas de Entrada / Saída

As linhas de comunicação de entrada ou saída de dados (io) no microcontrolador são chamadas de portas, conforme vemos na representação esquemática do microcontrolador, encontramos 16 portas de entrada ou saída (I/O). Estas portas estão divididas em dois grupos, o grupo A com 8 portas e o grupo B com 8 portas. O grupo A é chamado de PORTA e o grupo B de PORTB. Cada porta para ser identificada é chamada de R seguido do grupo (A, B) e do número da porta, sempre iniciando em zero. Ex. RA0, RA1, RA2, RB0, RB1, RB2.

Ao contrario de alguns modelos, o 16F62x pode utilizar oscilador interno, ou externo. Quando usar o oscilador interno, sobrarão duas portas para utilizar como entrada ou saída.

Porta Bidirecional A (PORTA)

Consta de 8 linhas de entrada/saída multifuncionais com as seguintes funções:

Pino	Função	Tipo	Saída	Descrição
Ra0	Ra0 AN0	ST NA	CMOS -	Entrada / saída bidirecional Entrada do comparador analógico

Pino	Função	Tipo	Saída	Descrição
Ra1	Ra1 AN1	ST NA	CMOS -	Entrada / saída bidirecional Entrada do comparador analógico
Ra2	Ra2 AN2 Vref	ST NA -	CMOS - AN	Entrada / saída bidirecional Entrada do comparador analógico Saída de voltagem de referência
Ra3	Ra3 AN3 CMP1	ST NA -	CMOS - CMOS	Entrada / saída bidirecional Entrada do comparador analógico Saída do comparador 1
Ra4	Ra4 TOCKL CMP2	ST ST -	OD - OD	Entrada / saída bidirecional Entrada de sinal para o timer0 Saída do comparador 2
Ra5	Ra5 MCLR VPP	ST ST -	- - -	Entrada / saída bidirecional Master Clear – Reset Voltagem de programação
Ra6	Ra6 OSC2 CLKOUT	ST XTAL -	CMOS - CMOS	Entrada / saída bidirecional Entrada do oscilador a cristal Saída de clock
Ra7	Ra7 OSC1 CLKIN	ST ST XTAL	CMOS - -	Entrada / saída bidirecional Entrada do oscilador a cristal Entrada de clock externo

ST = Schmitt trigger, NA = Analógico, OD = Coletor Aberto

Porta Bidirecional (PORTB)

Consta de 8 linhas de entrada/saída multifuncionais com as seguintes funções:

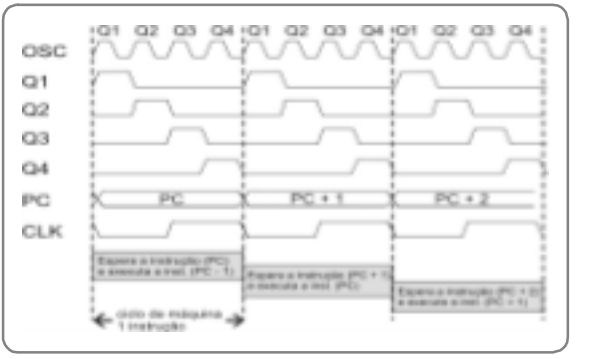
Pino	Função	Tipo	Saída	Descrição
Rb0	Rb0 INT	TTL ST	CMOS	Entrada / saída bidirecional Interrupção externa
Rb1	Rb1 RX DT	TTL ST ST	CMOS CMOS	Entrada / saída bidirecional Recebimento comunicação serial USART Entrada / saída de dados sincromono
Rb2	Rb2 TX CK	TTL ST	CMOS CMOS CMOS	Entrada / saída bidirecional Transmissão comunicação serial USART Entrada / saída de clock sincromono
Rb3	Rb3 CCP1	TTL ST	CMOS CMOS	Entrada / saída bidirecional Captura / comparação / PWM
Rb4	Rb4 PGM	TTL ST	CMOS CMOS	Entrada / saída bidirecional Programação em baixa voltagem
Rb5	Rb5	TTL	CMOS	Entrada / saída bidirecional
Rb6	Rb6 T10S0 T1CKL PGC	TTL - ST ST	CMOS XTAL - -	Entrada / saída bidirecional Saída do oscilador do timer1 Entrada do oscilador do timer1 Clock para programação

Pino	Função	Tipo	Saída	Descrição
Rb7	Rb7 T10SI PGD	TTL XTAL ST	CMOS CMOS	Entrada / saída bidirecional Entrada do oscilador do timer1 de Sleep Entrada / saída de dados para programação

ST = Schmitt trigger, NA = Analógico, OD = Coletor Aberto

3.15 Oscilador

Todo microcontrolador ou microprocessador para que funcione, necessita de um sinal de relógio (clock) para fazê-lo oscilar já que todas as operações internas ocorrem em perfeito sincronismo. No PIC o clock tem quatro fases, chamadas de Q1,Q2,Q3 e Q4. Estas quatro pulsações perfazem um ciclo máquina (instrução), durante o qual uma instrução é executada. Internamente o contador de programa (PC) é incrementado a cada ciclo Q1 onde a instrução é requisitada da memória de programa e armada na instrução registrada em Q4. A instrução é decodificada e executada no período Q1 a Q4.

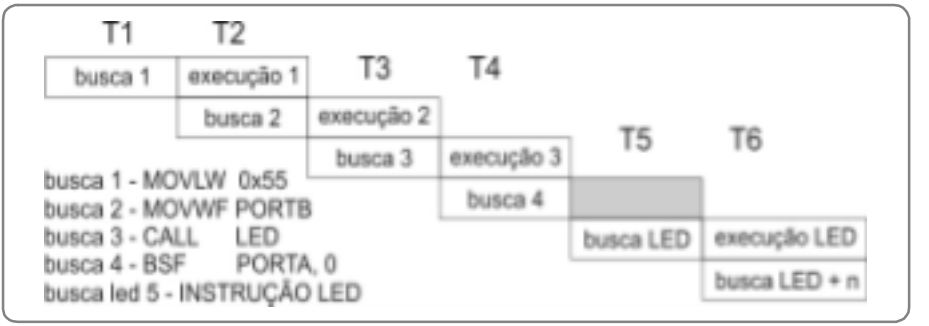


No 16F62x temos 8 possibilidades para o clock, selecionado através da palavra de configuração como vimos no item 4.13.

3.16 Pipeline

O processo “pipelining” é uma técnica de segmentação que permite ao microcontrolador realizar simultaneamente a busca de código de uma instrução da memória de programa, num ciclo de instrução, enquanto que a sua decodificação e execução, são feitos

no ciclo de instrução seguinte. Consideramos que cada instrução é armada e executada em um ciclo de máquina, contudo se uma instrução provocar uma mudança no conteúdo do contador de programa (PC), caso ele não aponte para o endereço seguinte na memória de programa, mas sim para outro (como acontece em saltos ou chamadas de sub-rotinas), então deverá considerar-se que a execução desta instrução demora dois ciclos de máquina, porque a instrução deverá ser processada de novo mas desta vez a partir do endereço correto. veja:



T1 é feita a busca da instrução MOVLW 0x05

T2 é executada a instrução MOVLW 0x05 e feita a busca da instrução MOVWF PORTB

T3 é executada a instrução MOVWF PORTB e feita busca da instrução CALL LED

T4 executado o salto (CALL) e feita a busca da instrução seguinte BSF PORTA,0 e como BSF PORTA,0 não é a primeira instrução após entrar na sub-rotina LED, faz-se necessário uma nova leitura do salto, gastando um ciclo.

T5 busca da instrução da subrotina

T6 execução da primeira instrução da sub-rotina e busca da próxima instrução.

3.16.1 Oscilador com cristal modo XT, LP ou HS

O cristal ou ressonador cerâmico é conectado em, paralelo nos pinos RA6(OSC1) e RA7(OSC2) com dois capacitores para massa, conforme a tabela

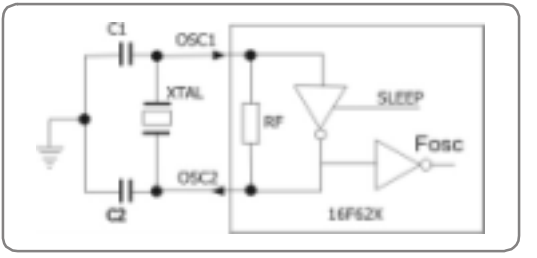
seleção do capacitor para uso com cristal.

Modo	Freqüência	OSC 1 - C1	OSC 2 - C2
LP	32 Khz. 200 Khz.	68 - 100 pF. 15 - 30 pF.	68 - 100 pF. 15 - 30 pF.
XT	100 Khz. 2 Mhz. 4 Mhz.	68 - 150 pF. 15 - 30 pF 15 - 30 pF.	150 - 200 pF 15 - 30 pF 15 - 30 pF.
HS	8 Mhz. 10 Mhz. 20 Mhz.	15 - 30 pF 15 - 30 pF. 15 - 30 pF.	15 - 30 pF 15 - 30 pF. 15 - 30 pF.

seleção do capacitor para uso com ressonador cerâmico

Modo	Freqüência	OSC 1 - C1	OSC 2 - C2
XT	455 Khz. 2 Mhz. 4 Mhz.	22 - 100 pF. 15 - 68 pF 15 - 68 pF.	22 - 100 pF 15 - 68 pF 15 - 68 pF.
HS	8 Mhz. 20 Mhz.	10 - 68 pF 10 - 22 pF	10 - 68 pF 10 - 22 pF

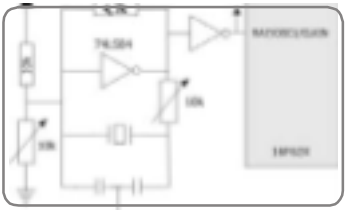
Modo de ligar o cristal



3.16.2 oscilador com cristal paralelo

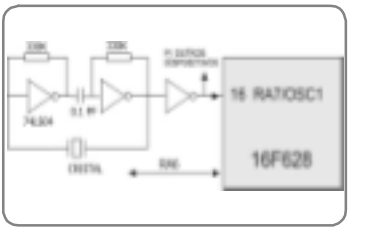
Preparado com portas TTL, este circuito simples, apresenta boa estabilidade e performance, fazendo uso da fundamental do cristal. Necessita de um resistor de 4,7k para uma realimentação

negativa para estabilidade do circuito e os potenciômetros de 10k faz ajuste (bias) do 74AS04 em uma região linear.



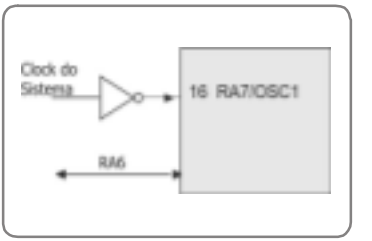
3.16.3 Oscilador com cristal em série

Também desenhado para usar a fundamental do cristal, necessita de um resistor de 330k para prover realimentação negativa



3.16.4 Clock externo

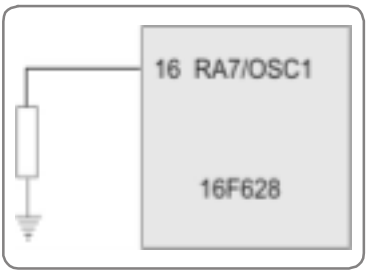
Quando em uma determinada aplicação, já existe um sinal de clock (mínimo de 200Khz.), pode ser usado para o microcontrolador através do pino OSC1, conforme a imagem abaixo:



3.16.5 Oscilador com resistor externo

Sem dúvida neste modo, temos uma economia no custo do projeto, com apenas um resistor para massa podemos controlar a freqüência do microcontrolador. O resistor drena a corrente “bias” DC de controle do oscilador e em adição a ao valor da resistência a freqüência do oscilador irá variar de unidade em unidade, em função da voltagem DC e da temperatura. O parâmetro de controle é corrente DC e não capacitância.

Resistor	Freqüência
0	10.4 Mhz.
1k.	10 Mhz.
10k.	7.4 Mhz.
20k.	5.3 Mhz
47k.	3 Mhz.
100k.	1.6 Mhz.
220k.	800 Khz.
470k.	300 Khz.
1M.	200 Khz.



3.16.6 Oscilador interno 4 Mhz.

Este modo provê 4Mhz fixo @5,0 volts e 25ºC podendo sofrer variações no mínimo de 3,65 Mhz E de no máximo 4,28 Mhz conforme alimentação e temperatura.

3.16.7 Oscillator Start-Up timer (OST)

O OST faz uma pausa de 1024 ciclos de clock através da entrada do OSC1, para que haja uma estabilização de tensão e periféricos. Ocorre somente na utilização de oscilador no modo XT, LP e HS.

3.17 Reset

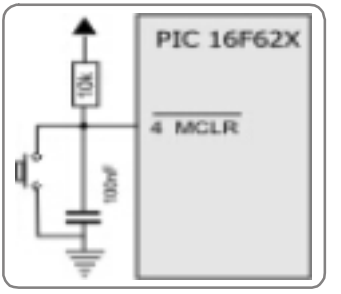
O vetor RESET, localização 0x00h, ativo em nível baixo, leva o microcontrolador a reiniciar seus registradores para valores iniciais pré-definidos na fabricação e um dos mais importantes efeitos de um reset, é zerar o contador de programa (PC), o que faz com que o programa comece a ser executado a partir da pri-

meira instrução do software programado. Em suma Este reinício de atividade pode ocorrer de causa manual, por deficiência na alimentação ou mesmo erro no software. Para o microcontrolador funcionar corretamente deve ter seu pino MCLR em nível alto. Caso ocorra uma variação para zero, ocorre a condição de RESET. Este evento pode ocorrer de seis formas:

1	Nível baixo em MCLR durante a operação normal.
2	Power-on reset (POR).
3	Brown-out detect (BOD).
4	Reset durante o repouso (SLEEP).
5	Estouro de WDT durante um repouso (SLEEP).
6	Estouro de WDT em operação normal

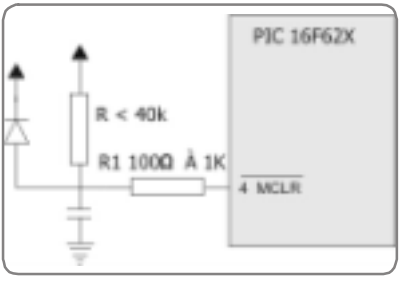
3.17.1- Reset Normal

A condição de reset normal ocorre colocamos o pino MCLR em nível baixo. Normalmente isto é feito através de uma chave Push-Bottom, como mostrado no esquema ao lado.



3.17.2- Reset Power-on (POR)

Este reset segura o chip em condição de reset tempo até que VDD esteje em nível suficiente para operação. É selecionado pelo bit 1 do registro PCOM endereço 0Ch.



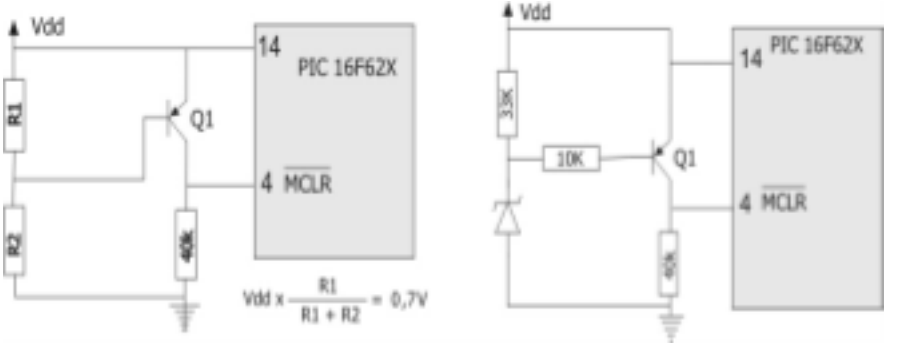
3.17.3- Power-up Timer (PWRT)

Este provê um tempo fixo nominal de 72ms (28ms a 132ms) para uso em power-on reset e brown-out.

3.17.4- Brown-out detect (BOD)

Se VDD cair abaixo de VBOD por mais tempo do que 100us (TBOD), então ocorrerá a situação do brown-out resetando o microcontrolador. Se VDD cair abaixo de VBOD, por um período inferior a TBOD, então a condição não terá garantias de ocorrer. Em qualquer reset (Power-on, brown-out, WDT, etc..) o chip continuará em reset até que VDD suba acima de VBOD, invocando o power-on timer dando um atraso adcional de 72ms.

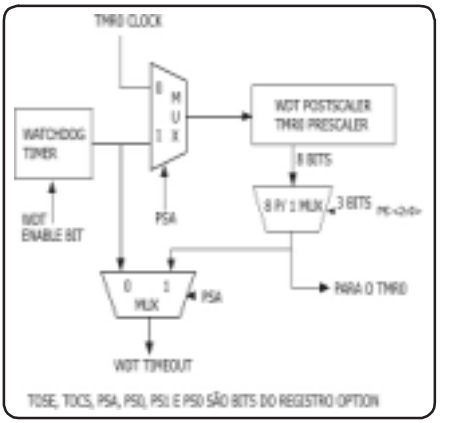
Pode ser habilitado ou desabilitado pelo bit BODEN da palavra de configuração no endereço 2007h e sua ocorrência pode ser monitorada pelo bit BOD do registro PCON endereço 8Eh



3.17.5- Reset por transbordamento de WDT

Quando ocorre transbordamento do WDT, normalmente em 18ms, este, leva o microcontrolador a condição de reset.

3.18- WatchDog Timer (WDT)



A tradução deste termo não faz sentido, mas pode ser entendido como “cão de guarda”, é um contador de 8 bits, preciso, que atua como temporizador e tem o objetivo de vigiar o microcontrolador impedindo que este entre em alguma rotina ou alguma instabilidade, que o faria trabalhar em um loop infinito ou parasse de responder.

O WDT tem um controle de tempo independente do oscilador principal com base em uma rede RC, causando normalmente o estouro, normalmente em de 18 ms. (podendo ser aplicado postscaler), causando um reset no microcontrolador.

Para ativá-lo devemos colocar em 1 o bit WDTE da palavra de configuração, e estando ativo, caso seu software deixe de responder por algum motivo, o WDT irá estourar causando um reset no microcontrolador carregando novamente o software. Devido a esta característica talvez a melhor definição para este sinal é “cão de guarda”.

Havendo necessidade pode-se aplicar o divisor de freqüência (postscaler) no WDT o qual poderia alcançar tempos de até 2 segundos.

Para que o seu software não seje resetado a cada estouro de WDT, deve-se limpa-lo em períodos de tempo inferior ao calculado para o estouro, utilizando-se a instrução CLRWDT ou SLEEP, sendo que esta última colocaria o microcontrolador em repouso.

O projetista / programador deve colocar a instrução CLRWDT em pontos estratégicos de seu software de acordo com o fluxo lógico de maneira que o software vigie o WDT evitando o seu estouro. Por exemplo, se no software há uma rotina ou temporização que gaste um determinado tempo várias vezes maiores que

o WDT, nesta rotina deve haver uma instrução CLRWDT.

A instrução CLRWDT simplesmente zera o conteúdo do registrador WDT, reiniciando a contagem. Já a instrução SLEEP além de zerar o WDT, também detém todo o sistema entrando em repouso com baixo consumo de energia.

3.19- Set de instruções

Chamamos de Set de instruções o conjunto de instruções que comandam o microcontrolador. Estes comandos já vêm gravados de fábrica e ficam na memória ROM. A CPU RISC contém um número reduzido de instruções isto o torna mais fácil de aprender, pois temos menos instruções. No entanto temos que reutiliza-las mais vezes para realizar operações diferentes, Ao contrário outras CPU como o x86 base de todos os computadores INTEL tem um set de instruções gigantesco. Desta forma temos instruções separadas para cada operação, aumentando também o grau de aprendizado de seu assembler.

No PIC o set de instruções divide-se em três grupos:

- Operações orientadas para Byte
- Operações orientadas para bit
- Operações orientadas para literais.

Conjunto de instruções para operações orientadas por byte

Mnemônico	Ciclos	Afeta Status	Descrição
ADDWF f,d	1	C,DC,Z	soma W e f, resultado em W
ANDWF f,d	1	Z	AND W com F
CLRF f	1	Z	CLear File f – limpa o valor da variável
CLRW	1	Z	CLeaR Work – limpa o valor de W
COMF f,d	1	Z	COMplementa F
DECF f,d	1	Z	DECrementa F
DECFSZ f,d	1 ou 2	-	DECrementa F e Salta se o result. for Zero

Mnemônico	Ciclos	Afeta Status	Descrição
INCF f,d	1	Z	INCrementa F
INCFSZ f,d	1 ou 2	-	INCrementa F e Salta se o result. For Zero
IORWF f,d	1	Z	Inclusive OR W com F
MOVF f,d	1	Z	MOVe para F
MOVWF f	1	-	MOVe W para F
NOP	1	-	Não faça nada
RLF f,d	1	Z	Rotaciona F a esquerda (Left)
RRF f,d	1	Z	Rotaciona F a direita (Right)
SUBWF f,d	1	C,DC,Z	SUBtrai W de F
SWAPF f,d	1	-	Troca bits de posição
XORWF f,d	1	Z	Exclusive OR W com F

Conjunto de instruções para operações orientadas para bit

BCF f,d	1	-	Bit Clear File – limpa (0) um bit do byte
BSF f,d	1	-	Bit Set File – seta (1) um bit do byte
BTFSC f,d	1 ou 2	-	Bit Testa File Salta se for zero - Clear
BTFSS f,d	1 ou 2	-	Bit Testa File Salta se for um - Setado

Operações orientadas para literais e controle de operações.

ADDLW k	1	C,DC,Z	Soma (ADD) Literal com W
ANDLW k	1	Z	AND F com W
CALL k	2	-	Chama uma sub-rotina
CLRWDT	1	TO,PD	Limpa WatchDog Timer
GOTO k	2	-	Go To Adress – vai para um endereço
IORLW k	1	Z	Inclusive OR Literal com W
MOVLW k	1	-	MOVa Literal para W
RETFIE	2	-	Retorne de Interrupção
RETLW k	2	-	RETorne com Literal em W
RETURN	2	-	Retorne de uma sub-rotina
SLEEP	1	TO, PD	Vai para o repouso
SUBLW k	1	C,DC,Z	SUBtraia Literal de W
XORLW k	1	Z	EXclusive OR Literal com W

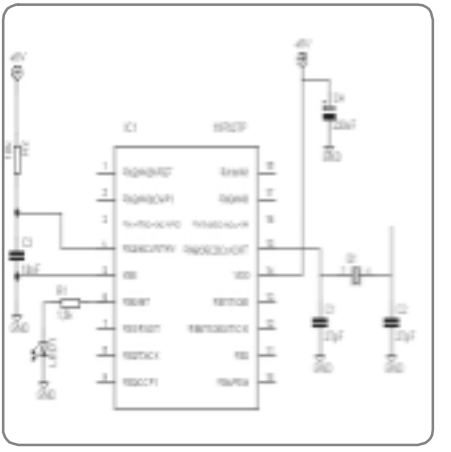
Capítulo 4

Construindo o Primeiro projeto:

4.1 Pisca Led

Nosso primeiro projeto será fazer um led piscar (oscilar) a uma certa freqüência. Neste primeiro exemplo, embora simples, serve perfeitamente para mostrar o uso das intruções vistas até aqui. Para tal usaremos o seguinte esquema ao lado:

Neste caso a forma de controle do tempo em que o led permanece aceso ou apagado através do artifício de fazer com que o microcontrolador perca tempo, este artifício chama-se Delay Time.



A rotina abaixo faz um loop, determinado pelo valor no registrador w (work), assim o numero de ciclos de máquina deve ser colocado em W antes de chamar a rotina. Para um período de tempo maior devemos utilizar variável (registrador) declarada no início do software, com valor antes de entrar na rotina de tempo. Veja:

```
milisegundos
movlw   miliseg
delay                           ;       total   1000    ciclos
movlw           0xF8            ;       248
call                    microsegundos   ;       (       248     x       4       )
+       2       =       994     microsseg.
                nop                             ;       995
```

```
decfsz  miliseg,        F       ;       subtrai 1,      salta   se      for     0
goto                    delay           ;       ainda   não     é       zero,
então   continua
retlw           0
                        ;       sai     desta   rotina  e       volta   a
rotina
principal

                                                                        ;       call
+       movlw   +       loops   +       retlw


;       2       +       1       +       995     +       2       =       1000
microsegundos
microsegundos:
addlw           0xFF
        ;       subtrai 1       de      w       ou      w       =       w       -
1
btfss           STATUS, Z                                       ;       salta   se
w       =       0
goto                    microsegundos   ;       ainda   não     é       zero
então   repete
retlw           0
                                ;       retorna a       rotina  principal
```

Este processo se repete quantas vezes for o valor da variável “miliseg” colocada antes de chamar o procedimento “milisegundos”. Assim para um delay de 10 mSeg fazemos:

```
Movlw                   0x10
Call                            milisegundos
```

Trabalhando com um cristal de 4Mhz temos 1 microsegundos por ciclo, então no caso esta rotina no máximo, nos daria 255 milisegundos, o que seria suficiente para ver um flash rápido do led, porém, podemo ampliar esta faixa adcionando ajustando a rotina milisegundos para 250 e chamá-la 4 vezes, assim teremos um segundo, mais precisamente 1.000035 segundos.

```
umsegundo
                                        ;       esta    rotina  irá     executar
um      loop    quantas vezes   for
                                        ;       o       valor   de      nsegundos.
Neste   caso    (       4       x       250     )       =       1.000035
seg.

movlw           0xFA
        ;       carrega W       com     250
call                    milisegundos            ;       executa loop    por     250
milisegundos
decfsz  nsegundos,      F               ;       número  de      segundos        =
zero    ?
goto                    umsegundo                                       ;       não,
```

```
então   repete  a       rotina
retlw           0
                        ;       sim,    então   sai
```

Veja o código fonte de nosso exemplo.

```
;-------------------------------------------------
;Projeto........:      Pisca_Led
;Cliente........:      Programando     Microcontroladores      PIC
;Desenvolvimento:      Renato
;Versão:........:      1.00
;Modelo.........:      16F627  -       16F628
;Descrição......:      Faz     um      led     lig.    na      porta   RB0
piscar  a       1       Hz.
```

```
;-------------------------------------------------

PROCESSOR       16F627
#include        <P16F627A.INC>

__CONFIG                        _CP_OFF &       _WDT_OFF        &       _PWRTE_ON
&       _XT_OSC

ORG                                     0x0000
;-------------------------------------------------
;variáveis      de      memórias
miliseg                 equ     0x20
nsegundos       equ     0x21
;-------------------------------------------------
inicio:
nop
                                                        ;       fca     um
ciclo   sem     fazer   nada
movlw                   0x00
        ;       zera    o       registrador     W
movwf                   STATUS                                          ;
coloca  o       valor   de      W       em      Status
bsf                                     STATUS, RP0             ;       Vai     para
o       banco   1       da      RAM
movlw                   b'00000000'             ;       coloca  0       no
registrador     W
movwf                   TRISA
;       determina       o       modo    do      PORTA
movlw                   b'00000000'             ;       coloca  0000    0001    em
W
movwf                   TRISB
;       Bit     0       de      PORTB   como    saída
bcf                                     STATUS, RP0             ;       retorna ao
banco   0
crlf                            PORTB
        ;       limpa   o       buffer  do      PORTB
clrf                            PORTA
        ;       limpa   o       buffer  do      PORTA
loop:
bsf                                     PORTB,  0
;       led     acendeu na      porta   RB0
movlw                   0x04
        ;       carrega W       com     4
movwf                   nsegundos                               ;       carrega o
valor   para    variável
call                            umsegundo                               ;
chama   rotina  de      um      segundo
bcf                                     PORTB,  0
;       led     apagou  na      porta   RB0
movlw                   0x04
        ;       carrega W       com     4
movwf                   nsegundos                               ;       carrega o
valor   para    variável
call                            umsegundo                               ;
chama   rotina  de      um      segundo
goto                            loop
                ;       repete  o       processo        infnitamente

;-------------------------------------------------
;       esta    rotina  irá     executar        um      loop    quantas vezes   for
;       o       valor   de      nsegundos.      (       4       x       250     )
```

```
=       umsegundo       1.000035        seg.
movlw           0xFA
                ;       carrega W       com     250
call                    milisegundos                    ;       executa loop    por
250     milisegundos
        decfsz  nsegundos,      F                       ;       número  de
segundos        =       zero    ?
goto                    umsegundo                                               ;
não,    então   repete  a       rotina
retlw           0
                                        ;       sim,    então   retorna com     0
em      w

;-------------------------------------------------
milisegundos
movwf           miliSeg
Delay                           ;       total   1000    ciclos
movlw           0xF8            ;       248
call                    microsegundos   ;(      248     x       4       )       +
2       =       994     microssegundos
nop                             ;       995
decfsz  miliSeg,        F       ;       subtrai 1       e       salta   se      for
0
goto                    Delay
        ;       ainda   não     é       zero,   então   continua
retlw           0
                                ;       sai     desta   rotina  e       volta   a
rotina
principal
```

```
;--------------------------------------------------
microsegundos:
addlw           0xFF
        ;       subtrai 1       de      w       ou      w       =       w       -
1
btfss           STATUS, Z                                       ;       salta   se
w       =       0
goto                    microsegundos   ;       ainda   não     é       zero
então   repete
retlw           0
                                ;       retorna a       rotina  principal
;--------------------------------------------------
END;
;--------------------------------------------------
```

Este exemplo exemplifica o modo de controle de portas e funções internas do PIC. Ao invés do led, poderíamos ter um mecanismo hidráulico fazendo uma determinada tarefa em períodos de tempo iguais ou com variações. Por exemplo à máquina que coloca a tampinha em refrigerantes, cerveja e muitos outros produtos. Basicamente é a mesma coisa do led onde você aciona um dispositivo ou vários, aguarda um determinado tempo ou aguarda sinal de outros sensores e depois de acordo com o seu software aciona a(s) saída(s). Poderia também ser uma esteira, onde a saída do PIC acionaria um rele, acionando o motor da esteira em períodos de tempos pré-determinados, enfim praticamente não há limites para o número de coisas que podem ser feitas, baseado ou derivado deste pequeno exemplo.

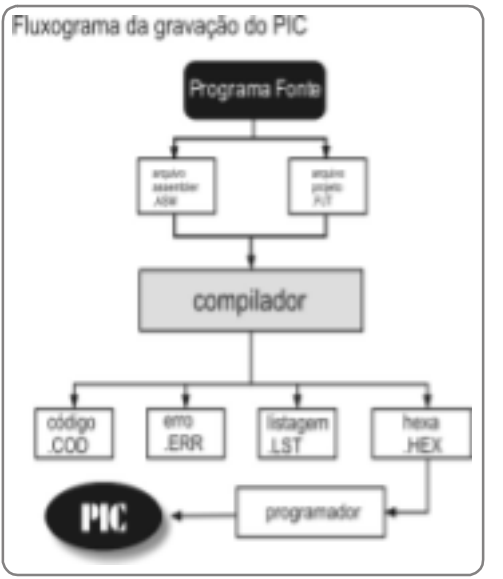
Mas para você que pensa que acabou, esta redondamente enganado, agora temos que converter este texto (programa) em números hexadecimais e coloca-lo dentro da memória do PIC. Para tal usaremos o MPLAB.

4.1 MPLAB versão 7.0

MPLAB é uma ferramenta da MicroChip para edição do software, simulação e até gravação do microcontrolador. Este tem a função de auxiliar no desenvolvimento de projetos, facilitando assim a vida do projetista. Esta ferramenta é distribuída gratuitamente pela MicroChip, podendo ser baixado da internet diretamente do site do fabricante no endereço www.microchip.

com ou em nosso site.

O MPLAB integra em uma única ferramenta, um editor de programa fonte, compilador, simulador e quando conectado às ferramentas da Microchip também integra o gravador do PIC, o emulador etc.



O programa fonte digitado, será convertido pelo MPLAB em códigos de máquina (veremos logo mais) que será gravado e executado pelo microcontrolador. Normalmente todo software que converte uma seqüência de comandos em linguagem de máquina é chamado de compilador. O compilador é composto por diversos níveis desde analisador léxico até linkeditor.

O ponto alto do MPLAB é o simulador, que permite que você rode (execute) seu programa sem a necessidade de grava-lo no microcontrolador, permitindo assim fazer diversas correções enquanto se desenvolve o software. Não desanime se o primeiro software que você fizer apresentar algum problema, lembre-se que nada acontece do dia para a noite, mas com dedicação e perseverança, fará com que você, caro leitor, supere as dificuldades que aparecerem.

O conceito do MPLAB é que você leitor trabalhe com pastas de projetos, ao fazer um novo projeto, primeiro crie uma pasta em seu computador, depois inicie o MPLAB iniciando um novo projeto ou abrindo se o projeto já existir. Na pasta do projeto que o MPLAB mantém está armazenado todas as informações do PIC utilizado, do clock utilizado da linguagem de programação, das posições das janelas, enfim o projeto de forma global. Para encerrar o projeto salve e feche apenas o projeto.

Crie uma pasta em seu computador chamada “c:\projeto\pisca_led”, depois inicie o MPLAB, quando estiver no ambiente de

trabalho do MPLAB você verá o seu menu principal. Selecione a opção “Project Wizard”; esta janela permite criar um projeto em quatro passos:

Passo 1)- Seleção do tipo do microcontrolador utilizado, selecione PIC16F628A

Passo 2)- Seleção da linguagem a ser utilizada. Posteriormente faremos alteração nesta janela, por ora deixe como está. “Microchip MPASM Toolsuite”

Passo 3)- Especifique o nome do projeto “pisca_led” e a pasta de armazenamento dos arquivos “C:\projeto\pisca_led”.

Passo 4)- Seleção de arquivo fonte “.ASM”, é a seleção de arquivos fontes que farão parte do projeto. Se já tiver o arquivo com a extensão .ASM poderá adicioná-los aqui, caso contrário apenas avance e conclua esta fase.

Depois no menu você encontrará a opção “Configure” e a sub opção “Configuration Bits”, nesta janela ajuste os bits de configuração do projeto.

- Oscilador = XT
- Watchdog Timer = off
- Power Up Timer = enabled
- Brown Out Detect = disabled
- Master Clear Enabled = enabled
- Low Voltage Program = off
- Data EE Read Protect = off
- Code Protect = off

Feche a janela “Configuration Bits” e Clique no Menu “File” e depois em “New”. Abrirá então a janela “Code Editor” onde você ira escrever o programa fonte. Esta janela é um editor de texto comum como o bloco de Notas do Windows, com a diferença que ele diferencia os comandos, literais

e os labels com cores diferentes, tudo isso para ajuda-lo na hora da escrita do software.

Depois de editar o soft, salve-o na pasta do projeto e clique com o botão esquerdo do mouse sobre “Source Files” na tela da esquerda e depois em “Add File”. localize na pasta o arquio digitado e adicione-o no projeto.

Ajuste a freqüência de clock em “Debugger” e depois em “Setting”, na janela que se abre, na aba “Clock”, digite a freqüência do clock que estamos trabalhando. 4 Mhz.

Digite o programa fonte com calma para evitar erros de digitação. A maneira de digita é livre, particularmente acho que as letras minúsculas deixam o texto mais legível a medida que sobra mais espaço entre elas, outros preferem tudo em maiúscula,. O importante é que você mantenha uma forma constante, pois o MPLAB faz diferenciação entre minúscula e maiúscula. Se o leitor nomear uma rotina como “UmSegundo” e em alguma parte fizer a chamada “call Umsegundo” irá gerar um erro, já que o caractere “S” é diferente. O mesmo vale para os comandos como “movwf Status, rp0”, aqui será gerado dois erros um referente ao “status” e outro referente a “RP0” que devem ser em maiúsculo. Para os exemplos vou utilizar letras minúsculas, o importante é o leitor ter ciência das diferenças para evitar estes erros e depois gastar o tempo procurando-os.

Após digitar o programa fonte, e se tudo estiver correto ,clique no menu “Project” e depois em “Make” ou use a tecla de atalho “F10” para compilar seu programa fonte e gerar o tão esperado arquivo hexa.

Durante o processo de compilação o leitor verá a janela de progresso, e após o término da compilação será exibida a janela “OutPut” com indicação visual dos passos tomados pelo compilador.

Caso o arquivo de código fonte contenha algum erro, na compilação este será detectado e o processo falhara com término do compilador, sendo o resultado exibido na janela “OutPut” para leitura do usuário e correções no arquivo de código fonte.

Neste exemplo eu coloquei intencionalmente uma virgula

em uma linha aleatória e mandei compilar o projeto. A saída da janela “OutPut” foi a seguinte:

Erro do arquivo de código fonte, na linha 28:
, CALL milisegundos ; led aceso por um tempo

Erro reportado pelo MPLAB:
Error[108] C:\PROJETO\PISCA_LED\LED.ASM 28 : Illegal character (,)
Halting build on first failure as requested.
BUILD FAILED: Sun Ago 07 01:39:56 2006

Como se vê o MPLAB reporta a linha onde encontrou o erro e ainda lhe diz porque está errado, neste caso, foi encontrado um caractere inválido na linha 28. Fazendo a correção (apagando a virgula) e compilando novamente o projeto obtivemos a mensagem:

BUILD SUCCEEDED: Sun Ago 07 01:40:14 2006
Neste caso a compilação foi um sucesso.

Algumas vezes o software é tão pequeno e simples que é quase impossível haver erro, podendo ser gravado em seguida, mas na grande maioria das vezes é uma boa idéia fazer simulação do funcionamento do software antes de gravar o dispositivo e colocar no PCB. Esta simulação é feita no próprio MPLAB, da seguinte forma.

No menu principal, clique no item ‘Debugger’ depois no subitem “Select Tool” e finalmente em “MPLAB Sim”.

No menu “VIEW” selecione as opções, “File Register (Symbolic)” e “Special Function Register”, procure organizar as janelas de modo que todas estejam visíveis na tela.

Para a simulação do programa é conveniente, por uma questão de comodidade e velocidade, utilizar as teclas de atalho do MPLAB, as quais são aqui descritas com os seus respectivos usos, lembrando que estas opções estão dentro do menu “Debugger”.

- F9 (RUN) – faz com que o programa seje executado em ve-

locidade normal. Você nem ao menos o verá na tela. Normalmente esta opção é usada em conjunto com a tecla F2 (Breakpoint).

- ANIMATE – esta função não possui tecla de atalho; faz

com que o programa siga passo a passo com indicação na tela da linha que está sendo executada, e nas outras janelas abertas as indicações das respectivas funções.

- F5 (HALT)– faz uma parada na execução do programa.
- F7 (STEP INTO) – esta função faz com o programa seje

executado passo-a-passo como no ANIMATE com a diferença que a cada vez que pressionado esta tecla uma linha é executada, causando sua parada na linha seguinte da próxima instrução. Observe que a linha onde está parado o cursor ainda não foi executada.

- F8 (STEP OVER)– muito parecida com a função STEP INTO,

com a diferença~que executa a linha onde está o cursor, ou seja, a linha onde está parado o cursor já foi executada. A cada vez que esta tecla é pressionada a próxima linha é executada.

- STEP OUT – executa uma sub rotina, sendo muito útil no

caso de rotinas demoradas como é o caso do nosso exemplo. A sub rotina milisegundos é chamada 4 vezes e a cada vez chama a sub rotina microeec que executa 255 subtrações. Já pensou Ter que ficar teclando F7 ou F8 até sair desta sub rotina !

- F6 (RESET/PROCESSOR RESET) – esta função causa um

reset geral no programa posicionando o cursor na primeira linha de comando, zerando as memórias e os registradores.

- F2 (BREAKPOINT) – sem dúvida alguma umas das mais

úteis; esta função causa pontos de paradas no programa. Este procedimento pode ser feito de duas formas. A forma fácil é dar dois cliques bem no canto da linha onde deve ficar o ponto de parada, será colocado um ícone vermelho com um “B”, indicando “BreakPoint”. A forma mais difícil é teclar F2 e na janela digitar o

número da linha em hexadecimal, onde ficará o ponto de parada. Na janela “BreakPoint” é possível desabilitar ou remover o breakpoint existente

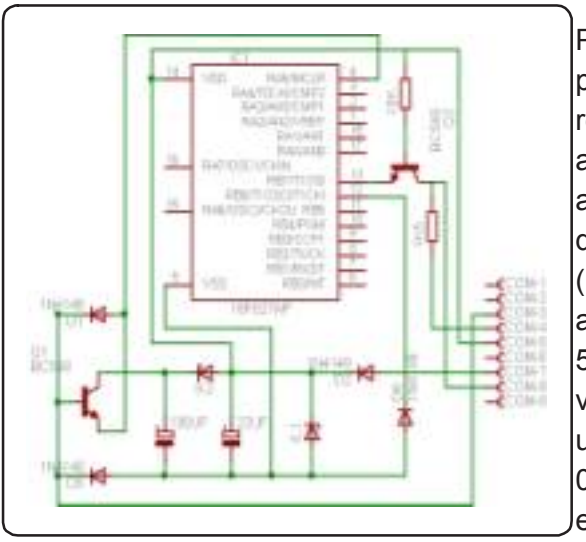
Algumas outras funções do MPLAB como veremos logo mais à frente na medida que forem necessárias para o nosso estudo, por

hora, após simular o funcionamento do soft, chegou o momento de gravá-lo no microcontrolador. Para isso é necessário ter em mão um gravador de microcontrolador para o modelo que estamos trabalhando.

4.2- O Gravador

O gravador é um dispositivo projetado em concordância com as características do microcontrolador. Existem muitos tipos de gravadores e softwares de aplicação. Alguns são muito simples e econômicos, outros são complexos e caros.

Dentre os diversos tipos o que nos tem apresentado melhor resultado é o gravado “JDM” (http://www.jdm.homepage.dk), cujo esquema original é apresentado na figura abaixo e esquema , layout de circuito impresso melhorado pode ser encontrado em nosso site em www.renato.silva.nom.br ou a critério do leitor também em forma de kit montado.

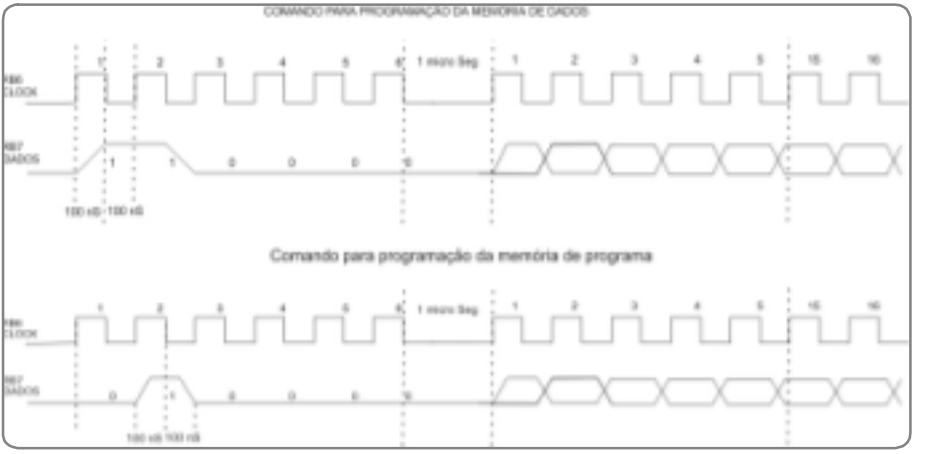


A programação do Pic é serial, feita por uns pinos específicos, requerendo uma tensão de alimentação VDD (4,5V a 5,0V) e uma tensão de programação VPP (12,0V a 14,0V) no modo alta voltagem e 4,5V a 5,5V no modo de baixa voltagem, ambos com uma variação mínima de 0,25V. A programação escreve na memória de programa, memória de dados, localização especial para o ID e o byte de configuração.

A memória de usuário vai de 0x0000 até 0x1FFF e no modo programação o espaço se estende de 0x0000 até 0x3FFF, com a

primeira metade (0x0000 a 0x1FFF) usada para o programa de usuário e a Segunda metade (0x2000 a 0x3FFF) inicia a memória de configuração.

O espaço da memória para configuração (0x2000 a 0x200F) é fisicamente implementado mas somente estará disponível o espaço de 0x2000 a 0x2007, as outras posições estão reservadas. A posição 0x2007 poderá ser fisicamente acessada pela memória do usuário. O usuário pode armazenar informação de identificação ID em quatro localizações, de 0x2000 até 0x2003. Estas posições podem ser lidas normalmente mesmo após a proteção de código ativada.



A programação opera com um simples comando, inserido bit a bit, na cadencia do pulso de clock. Os seis primeiros bits são de comando, seguido pelos 16 bits de dados do usuário.

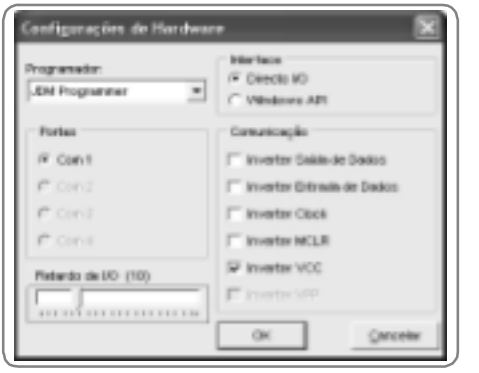
No início da operação levanta-se o VPP em 13V e após um período mínimo de 5uS, eleva-se VDD de 2,2V para 5,5V e após 5uS o clock começa a oscilar, fazendo com que o PIC aceite os dados a uma freqüência máxima de 10 Mhz. É importante manter o pino 10(RB4) em nível baixo, pois uma flutuação poderia fazer com que ele entrasse inadvertidamente em modo de programação em baixa voltagem. No modo de baixa voltagem temos a mesma lógica acima com a diferença que não precisamos elevar MCLR para 13V bastando elevá-lo para o nível alto e levar também o

PGM (RB4) no mesmo instante.

4.3.1 IC-Prog

Para efetuar a gravação, faremos uso do aplicativo ic-prog encontrado em http://www.ic-prog.com/. Este aplicativo é de fácil operação e grava uma quantidade considerável de chips, dependendo é claro do hardware de gravação.

Antes de utilizá-lo, deve-se configura-lo e caso se utilize o windows 2000™ ou windows XP™ , deve-se instalar o drive “ic-prog.sys”, iniciando o ic-prog no menu “Configurações/Opções” na aba “Diversos” encontramos “ativar drive NT/2000/XP”. Marque esta opção e reinicie o ic-prog.



Posteriormente faça a configuração do hardware JDM em “configurações/hardware”, conforme a figura ao lado.

4.3.2- Gravando o programa

Selecione o modelo do PIC em “Configuração/dispositivo/ Microchip Pic”, Depois abra o arquivo “pisca_led.hex”, selecione o modo do clock, e os fusíveis de configuração. E finalmente clique no ícone de gravação. Após o processo de gravação, o leitor será informado sobre o sucesso da gravação.

4.3.3- Erro de gravação.

Se após a verificação você obtiver e mensagem de erro, referenciando ao endereço 0x000, é porque a comunicação falhou ou não há alimentação no circuito. No entanto a mensagem pode se referenciar a outro endereço, por exemplo ao endereço 0x0007. Um dos meus PICs passou a apresentar esta mensagem na verificação. Eram alguns dados truncados na memória, bastou limpar o dispositivo e tudo voltou ao normal.

4.3.4- Arquivo Hexa

Alguns leitores podem estar se perguntando, o que são aqueles números que vimos na tela principal do ic-prog, ou como funcionam ?

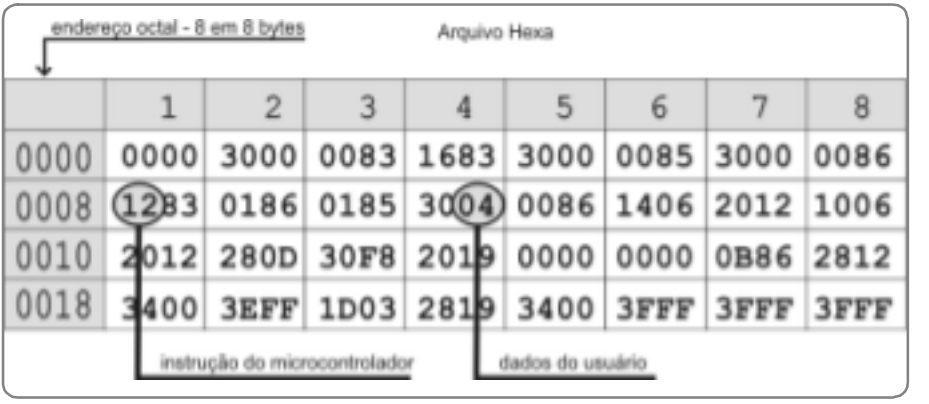
Bom, primeiramente aqueles números são do arquivo hexa que acabamos de compilar com o MPLAB. Estão organizados de dois em dois bytes, sendo a primeira instrução do microcontrolador, gravado em sua rom durante o processo de fabricação e o segundo byte são dados do usuário e posições de memória. Vejamos:

A organização é de oito em oito bytes, como o leitor pode ver na figura acima, portanto a primeira coluna representa números octais de oito em oito, assim nosso programa tem 29 bytes, indo da linha 0018(8), coluna 5(8). O primeiro byte 0000 é a nossa instrução NOP, nossa Segunda instrução é MOVLW 0x00 (mova literal para registrador W) e valor movido para w é 0 então fica 3000. 30 é a instrução MOVLW e 00 o valor movido.

Outro exemplo; A instrução GOTO LOOP: da linha 31 foi alocada pelo compilador como GOTO 0xD, onde 0xD é a localização do label LOOP na memória do microcontrolador.

A função do compilador é facilitar a escrita do programa e transforma-lo em números hexadecimais em conformidade com a tabela rom do microcontrolador utilizado, mais precisamente em com o set de instruções.

Desde o início deste o leitor foi conduzido por uma breve história do desenvolvimento dos microcontroladores, passando por funções lógica booleana, bases numéricas, descrição do hardware do microcontrolador chegando a programação assembler e gravação do dispositivo. No entanto nos próximos capítulos estudaremos formas de fazer tarefas mais complexas com menos esforço, fazendo uso da linguagem de programação em “C” e do compilador “CCS PIC C Compiler™ ”, tendo este demonstrado melhor resultado entre os diversos compiladores disponíveis.



	1	2	3	4	5	6	7	8
0000	0000	3000	0083	1683	3000	0085	3000	0086
0008	1283	0186	0185	3004	0086	1406	2012	1006
0010	2012	280D	30F8	2019	0000	0000	0B86	2812
0018	3400	3EFF	1D03	2819	3400	3FFF	3FFF	3FFF

Capítulo 5

Linguagem “C”

5.1- Linguagem de programação

A linguagem “C” nasceu na Bell Labs, divisão da AT&T, a famosa companhia americana de telecomunicações desenvolveu em seus laboratórios o sistema operacional Unix, posteriormente desenvolveu também a linguagem “C” em 1969 por Dennis Ritchie, que também é um dos principais criadores do Unix, à partir da linguagem “B” de Ken Thompson. Seu propósito era gerar uma linguagem de alto nível, em oposição à linguagem de máquina (Assembly), conhecida como de baixo nível. O “C” é uma linguagem para uso geral, ou seja, desenvolvimento dos mais diversos tipos de aplicação. Tem como características a modularidade, portabilidade, recursos de “baixo” e “alto” nível, geração de código eficiente, confiabilidade, regularidade, além de conter um número pequeno de comandos.

A linguagem “C” ou simplesmente “C” uma linguagem pequena composta de poucos comandos, e operadores, que pode apresentar variações de um fabricante para outro no modo de escrita e nas funções internas que por sua vez são grupos de comandos e operadores idealizados pelo construtor da linguagem para realizar uma determinada tarefa.

O pré-processador do compilador, reconhece os comandos e todos os símbolos definidos pelo comando #define, sendo pouco inteligentes fazem basicamente substituição. O verificador léxico ou “reconhecedor de palavras” passam pelo programa trocando

palavras-chaves e símbolos orientados para humanos por valores numéricos mais compactos orientados para máquina. Na verdade é uma grande tabela de pesquisa onde a palavra-chave é trocada por um código ou token, a grande maioria das mensagens de erro reportadas acontecem neste estágio.

Depois vem o analisador o grande perito em “C”. Ele sabe a sintaxe e quase toda a semântica da linguagem. Este possui a responsabilidade principal pela análise contextual, um erro se seja reconhecido apenas depois de dois ou três comandos terem sido analisados será descoberto. Unicamente responsável pela interpretação semântica, ele pode detectar um comando pedindo um ato impossível. Por exemplo uma divisão por zero.

Depois o código é otimizado, linkeditado com as funções de bibliotecas internas, regenerado e finalmente assemblado.

É justamente esta inteligência ao trabalhar com o código que faz um compilador eficaz, gerando menos código e utilizando menos memória do microcontrolador.

Após utilizar diversos compiladores existentes atualmente, optei pelo uso do CCS por apresentar uma gama de vantagens sobre os demais. O compilador CCS pode ser adquirido diretamente no site do fabricante no endereço www.ccsinfo.com onde se encontra também vários exemplos de utilização e outras informações referentes ao compilador.

A instalação do compilador é bem simples, basta executar o aplicativo e seguir as instruções das telas.

5.2- Comentários

Os comentários no programa fonte não têm função nenhuma para o compilador e serve apenas para aumentar a legibilidade e clareza do programa, podem ser inseridos com “//” que valem de onde começam até o fim da linha ou com “/*” e “*/”, sendo considerado comentário tudo entre “/*” e “*/”.

5.3- Identificadores

Um identificador é um símbolo definido pelo usuário que pode ser um rótulo (label), uma constante, um tipo, uma variável, um nome de programa ou subprograma (procedimento ou função). Normalmente devem começar com um caractere alfabético e não podem conter espaços em branco, podendo ter no máximo 32 caracteres, não havendo distinção entre maiúsculas e minúsculas.

5.4- Endentação

A endentação também não tem nenhuma função para o compilador e serve para tornar a listagem do programa mais claro dando hierarquia e estrutura ao programa.

5.5- Constantes

Constantes são valores declarados no início do programa e que não se alteram na execução do programa. Podem ser expressas em qualquer base,

5.6- Variáveis

Uma declaração de variável consiste do nome do tipo de dado seguido do nome da variável. Todas as variáveis devem ser declaradas antes de serem usadas. As variáveis devem ser declaradas no início de cada função, procedimento ou início do programa.

5.7- Elementos definidos pela linguagem C:

Letras (alfanuméricas) Aa até Zz;

Dígitos (numéricos) - 0 até 9;
Operadores;
Tipos de dados.

5.8- Operadores e Símbolos Especiais

*	multiplicação	a = b * c
/	divisão	a = b / c
%	resto	a = 13 % 3 a = 1
+	adição	a = b + c
-	subtração	a = b - c
=	atribuição	a = b
==	comparação	compara dois operandos ex. if(a == 10)
<	menor que	a < b
<=	menor ou igual	a <= b
>	maior que	a > b
>=	maior ou igual	a >= b
!=	diferente de	a = 10 Se(a != 3) então verdadeiro
<<	deslocamento a esquerda	3 (00000011) << 1 = 6 (00000110)
>>	deslocamento a direita	4 (00000100) >> 1 = 2 (00000010)
&	E	00000011 & 00000110 = 00000010
&&	lógica E (AND)	a=2 Se(a>1 && a<3) então verdadeiro
^	OU exclusivo	00001100 ^ 00000110 = 00001010
|	OU inclusivo	00001101 | 00001101 = 0001101
||	lógica OU (OR)	a=4 Se(a>3 || a<5) então verdadeiro
!	lógica Não (NO)	Se(! a) equivalente a Se(a == 0)
~	complemento	~00010101 = 11101010
“	delimitador de string	“pic”

5.9- Tipos de dados

Um Tipo de Dado define o conjunto de valores que uma variável pode assumir e as operações que podem ser feitas sobre ela.

Toda variável em um programa deve ser associada a um tipo de dado, conforme a tabela abaixo.

Tipo	Descrição do tipo
Static	Variável global inicializada com 0
int1	De_ne um n_mero com 1 bit faixa 0 ou 1
int8 ou Int	De_ne um n_mero com 8 bits faixa de 0 a 127
int16 ou Long	De_ne um n_mero com 16 bits faixa de 0 a 32.767
int32	De_ne um n_mero com 32 bits faixa de 0 à 3.4 E38
Char	De_ne um caractere com 8 bits faixa Aa à Zz
Float	De_ne um n_mero com 32 bits faixa de 3.4 E-38 à 3.4 E38
Short	Por padrão é o mesmo que int1

O conceito do “C” é que existe uma função principal chamada “main” que controla o bloco principal do programa, logicamente ficaria assim:

```
#include      <16F628A.h>
void   main() {
...declarações
};
```

A chave ({) abre uma estrutura de bloco de declarações, onde temos comandos, chamadas de funções ou blocos em assembler . Para cada chave aberta deve-se ter uma outra chave (}), indicando o fechamento do bloco de declarações.

```
void   main{
             while( true   ){
                                                  //repetição  infnita
                                    output_high(  PIN_B0 );
//coloca     o      bit    led     em     1
                                    delay_ms(     1000   );
                                    //aguarda     1      segundo
                                    output_low(   PIN_B0 );
       //coloca     o      bit     led    em     0
                                    delay_ms(     1000   );
                                    //aguarda     um     segundo
             };
};
```

Normalmente o compiladore dá uma ajudinha na hora de

escrever o código fonte alertando para essas particularidades. O modo de escrever é livre, você pode utilizar letras maiúsculas,

minúsculas ou combinadas, somente não pode haver espaço entre o label ou a variável, assim a seguinte declaração de variável “Int variável 1 ;” apresenta dois erros graves. Um deles é o uso de acentuação no “á” o outro é a presença de espaço. O correto para esta declaração seria o seguinte “int variável_1 ;”.

Os caracteres acentuados fazem uso do oitavo bit do byte de caracteres o que não é suportado pelos compiladores uma vez que na língua inglesa não existe acentuação fazendo uso portanto dos caracteres de até 7 bits ou até o caractere 127. No apêndice encontra-se uma tabela de caracteres ASCII com os caracteres básicos de 0 a 127 e uma tabela de caracteres estendido de 127 a 255.

Os comandos são:

IF, WHILE, DO, FOR, SWITCH, CASE, RETURN, GOTO, LABEL, BREAK, CONTINUE.

5.10- Comando “IF”

Este comando de estrutura condicional pode estar sozinho ou em conjunto com “else” ou “else if”, na construção de blocos de estruturas condicionais muito eficazes. Sempre recebe um ou mais parâmetros pois sempre comparamos alguma coisa com alguma coisa, e conforme o resultado da comparação fazemos algo diferente. Vejamos:

```
if(    modo   ==     0x01   ){
       //variável   modo   =      1?
              buffer =      0x03   &      0xFF;              //sim,
então  buffer =      2      AND    256
}
else{

//senão
              buffer =      0x03   ^      0xFF;              //buffer
=      2      XOR    256
};
```

No exemplo acima vemos o operador de igualdade/comparação (==) que indica se o operando da direita é igual ao da esquerda , e logo abaixo tempo o operando de atribuição (=), onde

o operando da direita atribui-se ao da esquerda.

Neste caso, houve a necessidade de se verificar a outra fase

da condição, e atribuir-lhe valor.

No caso acima se fez uma comparação e tomou-se uma decisão, se após a decisão houvesse mais de uma declaração, abriríamos um bloco de controle assim:

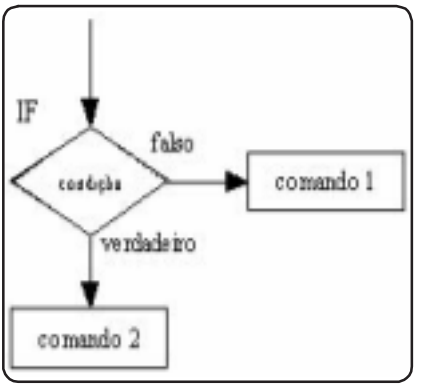
```
if(    TEMP   ==     0      ){
              output_high(  PIN_B0 )      ;
}
else   if(    TEMP   ==     1      )      {
              output_high(  PIN_B1 )      ;
}
else   if(    TEMP   ==     2      )      {
              output_high(  PIN_B2 )      ;
}
else{
              output_low(   PIN_B0 )      ;
              output_low(   PIN_B1 )      ;
              output_low(   PIN_B2 )      ;
};
```

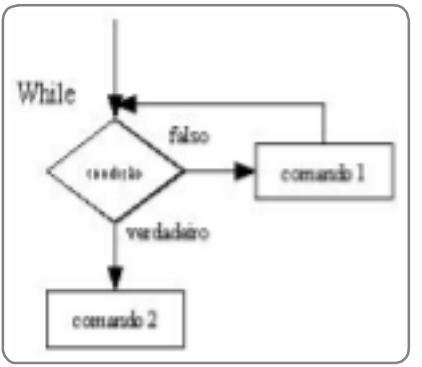
5.11- Comando “WHILE”

Este comando pode ser utilizado sozinho ou em conjunto com o comando “DO” construindo blocos de controle extremamente eficazes. “while” ou “enquanto” sempre avalia uma expressão enquanto ela é verdadeira podendo ou não executar uma declaração em conjunto.

```
while  (      input( PIN_B0 )      );                          //aguarda
o      pino   Rb0    descer
output_high(  PIN_B1 );
//coloca      o      pino   Rb1    em     1
while  (      !      input( PIN_B0 )      );           //aguarda    o
pino   Rb0    subir
output_low(   PIN_B1 );
       //coloca      Rb1    em     0
```

depois de compilado, foi gerado o seguinte código:





```
00BF:         BTFSC         PORTB.0
00C0:         GOTO                0BF
00C1:         BSF                       PORTB.1
00C2:         BTFSS         PORTB.0
00C3:         GOTO                0C2
00C4:         BCF                       PORTB.1
```

Neste caso, o comando While irá aguardar até que o pino Rb0 mude de 1 para 0, veja, caro leitor que o compilador colocou na linha 00BF a instrução para testar a flag Rb0 e saltar se estiver limpa, caso contrário na linha 00CD têm um salto para a linha 00BF, criando um loop até termos Rb0 igual a 0.

Quando a condição a ser avaliada não muda de estado temos um loop infinito como normalmente acontece dentro da função "main". No exemplo abaixo criamos nosso pisca_led.

```
void   main(){
//função     principal
             while( TRUE   ){                  //execute    sempre,
loop   infnito
                                 output_high(  PIN_B1 );
//pino B1     =      1,     led    aceso
                                 delay_ms(     1000   );
                                 //aguarda     1      segundo
                                 output_low(   PIN_B1 );
//pino B1     =      0,     led    apagado
             };

                                 //fecha       o      bloco  while
};

                                       //fecha       a      função
principal
```

5.12- Comando "DO"

Este comando "execute" sempre é executado em conjunto com o comando "while" formando um bloco "execute ... enquanto(verdadeiro)". com a diferença que a condição é avaliada posteriormente a execução,

ou seja, primeiro executa o bloco depois avalia se continua executando, ao contrário do “while”.

```
void   main(){
              do{
                                    output_high( PIN_B1 );
                                    delay_ms(    1000   );
                                    output_low(  PIN_B1 );
              while( !      input( PIN_B0 );
};
```

5.13- Comando FOR

Este comando é muito usado em expressões gerar loop controlado. Apresenta 3 parâmetros, sendo:

Primeiro parâmetro: é o valor inicial da variável a ser utilizada como comparativa para execução do loop controlado.

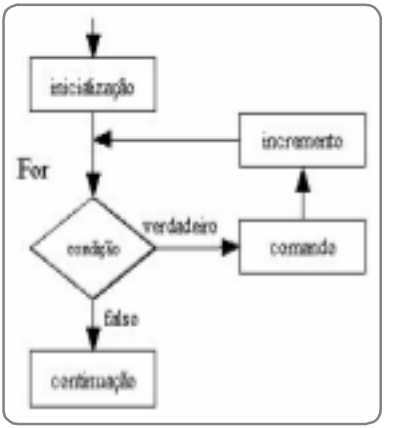
Segundo parâmetro: é a condição a ser avaliada para continuidade do loop.

Terceiro parâmetro: é a condição de incremento ou decremento, que o primeiro parâmetro terá até atingir o segundo parâmetro. Ex:.

```
//recebe      um     byte   no     pino   Rb1,   e      armazena      na
variável      "c"
for(   i      =      0;     i      <      8;     i++    ){
              while( !output_low( PIN_B0));                    //aguarda
sinal  de     clock
                                    shift_right(&c,     1,     PIN_B1 );
//recebe      um     bit
              };
};
```

Este loop começará com x=0 e será executado enquanto x for menor que 8, sendo incrementado de um em um. Em cada avaliação será executado o bloco de comandos onde um comando While aguardará um pulso no pino Rb.0 para capturar o bit no pino RB.1 e coloca-lo na variável "c" através de deslocamento de um bit a direita (shift). A avaliação de cima para baixo é mais eficiente

na geração do código, no exemplo acima foi gerado 12 linhas de



instruções e neste abaixo, 8 linhas de instruções, apenas mudando a forma de contagem e conseguimos economizar memória.

```
for(   i      =      8;     i      >      0;     i--    ){
               while( !output_low(  PIN_B0));                  //aguarda
sinal  de      clock
                                    shift_right(&c,     1,     PIN_B1 );
//recebe       um     bit
               };
};
```

5.14- Comando SWITCH

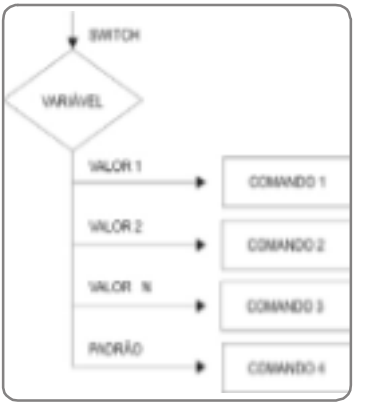
Este comando facilita bastante na tomada de múltiplas decisões, onde avaliamos uma condição echaveamos entre várias possibilidades. No entanto o mesmo utiliza mais memória do que utilizar grupos de if. Seu uso é feito em conjunto com o comando “case”. Vejamos:

```
if(    crc_check(    pc_buffer,    8      )      )
               switch(       pc_buffer[    3      ]      ){
                                    case   0x01   :      send_version()
;break;
                                    case   0x02   :      relay_on()
;break;
                                    case   0x03   :
reset_cpu();break;
               }
}
```

basicamente tomamos decisão, baseado em várias condições, podendo ter também, uma condição padrão caso nenhuma venha a ser avaliada como correta. veja.

```
switch (       temp   )      {
               case   0      :      output_high(  PIN_B0 );
                                    break;
               case   1      :      output_high(  PIN_B1 );
                                    break;
               case   2      :      {
```

```
                                                output_high(
PIN_B0 );
                                                output_high(
PIN_B1 );
                                    }
```



```
                                    break;
                default       :     output_high( PIN_B3 );
                                    break;
};
```

Neste caso faz-se a avaliação da variável “temp” comparando-a com valores pré-determinados, executando o comando ou blocos de comandos que satisfaça a condição, tendo após cada comando ou bloco de comandos, o comando “break” que faz um salto para o final do comando “switch” evitando assim novas comparações com as opções que vem abaixo. Caso nenhuma condição seje satisfeita e se houve a declaração “default” esta será executada como padrão.

Também é possível executar vários comandos após o chaveamento abrindo-se um bloco

5.15- Comando RETURN

Este comando retorna de uma sub-rotina ou função escrita pelo usuário, retornando um determinado valor ou resultado de operações.

```
//declaração da função que lê um byte na memória
EEPROM
//e retorna o valor da posição 02 da
memória.
int recupera_dado{
        return ( read_eeprom( 0x02 ) );
}
```

5.16- Comando GOTO

Este comando, dificilmente utilizado, faz um desvio incondicional altera a seqüência normal de execução em um bloco de comandos, transferindo o processamento para um ponto no programa fonte marcado com o rótulo especificado no comando GOTO.

```
Ex:   goto  loop:
```

5.17- Comando BREAK

Este comando pode ser usado para sair de um loop for, while ou do-while, switch passando a execução para o comando imediatamente seguinte ao loop.

```
while  (len-- >      0){
              i      ++;
              if(    i      ==     12     )      break;
};
```

5.18- Comando CONTINUE

Este comando causa o fim de um dos laços de uma repetição e o retorno imediato.

```
for(bit_counter=0;  bit_counter  <      16;    bit_counter++){
             if(!bit_test(crc_Dbyte,15)){
                                  crc_Dbyte     <<=    1;
                                  continue;
             }
             crc_Dbyte     <<=    1;
             crc_Dbyte     ^=     0xFFFF;
}
```

5.19- Estrutura de um Programa em C

Normalmente um programa em C possui três partes distintas: Bloco de Diretivas de Compilação, Bloco de declarações e Bloco de Implementação

No bloco de diretivas de Compilação, incluímos as diretivas para o compilador que são todas as palavras iniciadas pelo caractere “#”, inclusive definições feitas pelo usuário. Essas palavras são comandos para o compilador que não fazem parte do arquivo compilado, mas são necessários para sua geração, substituindo o código fonte ou incluindo novos blocos.

Os arquivos com a terminação .h são denominados arquivos de inclusão, ou simplesmente includes, e contêm informações que devem ser tratadas pelo compilador para que seje possível a geração do programa.

No bloco das declarações são declaradas todas as variáveis definidas pelo usuário.

No bloco de implementações são implementadas todas as funções que compõem o programa, a função “main” e todas as demais funções.

Na programação em “C”, as funções têm papel de destaque, tornando o programa mais modular e claro na escrita. Toda função sempre retorna um valor, ou retorna um valor vazio (void), recebendo um valor como parâmetro ou parâmetro vazio..

A função sempre deve ser declarada acima da função “main” para que o compilador não se desoriente e deve possuir indicação de que tipo de parâmetro retornar ou receber.

```
void   acende_led(   void   ){
                          output_high(  PIN_B0 );
                          delay_ms(     200    );
};
```

Estes parâmetros são também chamados de argumentos, que podem ser misturados com diversos tipos de dados, podem retornar um único valor através do comando “return”.

5.20- compilador “CCS C Compiler”

As diversas combinações de comandos, operandos e argumentos e o uso de ponteiros, faz com que o “C” seje altamente eficiente. Porém o uso difere um pouquinho da programação de microcomputadores, onde temos farta memória ram e alta velocidade de processamento. Para microcontroladores tenho experimentado diversos compiladores e avalio que o CCS Pic C Compiler™ da CCS INFO (www.ccsinfo.com) apresenta melhores resultados, por gerar um código mais enxuto e disponibilizar farta biblioteca de

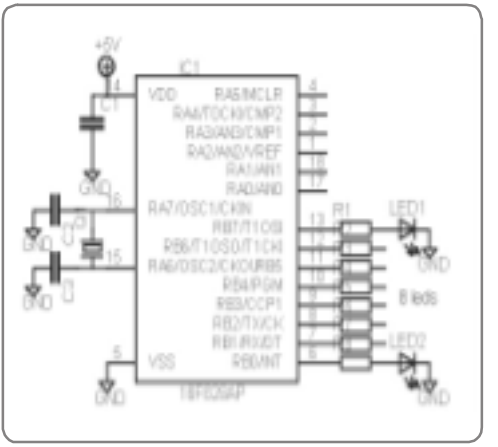
funções pré-definidas.

Após esta introdução ao “C”, penso que o correto é fazer alguns experimentos para reforçar o aprendizado e adquirir experiência com o compilador.

A instalação do compilador não apresenta complicações, basta executar o aplicativo e seguir as intruções das telas. Depois de instalado, crie em seu computador uma pasta para projetos, execute o atalho “Pic C Compiler” e no menu do aplicativo clique em “Project” depois “New” e depois em “Pic Wizard”.

Especifique o nome do arquivo como “byte_para_bit” na sua pasta de projetos e após salvar, termos uma tela, onde podemos ajustar o funcionamento do hardware do PIC através das opões e das orelhinhas na parte inferior da tela. Cada orelhinha refere-se a uma implementação do microcontrolador PIC, que por hora usaremos apenas esta primeira orelhinha ou como dizem alguns “abinha”.

Marque as opções como a figura e clique em “OK”, assim a janela de programa será preenchida com parte do código. Note aqui, na primeira linha, a diretiva, #include “C:\projeto\byte_para_bit.h”, onde por padrão, são colocadas as definições do projeto. Embora esta técnica esteje correta, eu particularmente não gosto deste procedimento e sempre copio o conteúdo do arquivo .h e transcrevo-o para o programa fonte. Desta



forma, acredito ter maior clareza sobre a escrita, o que farei aqui, exemplificando neste primeiro exemplo a exibição de bytes de 0 à 256 em binário.

```
#include     <16F628A.h>
#FUSES NOWDT  //Watch      Dog    Timer  desabilitado
#FUSES XT
//oscilador   cristal      <=     4mhz
#FUSES PUT           //Power      Up     Timer
#FUSES NOPROTECT     //sem  proteção     para   leitura      da
eprom
#FUSES BROWNOUT      //Resetar    quando detectar     brownout
#FUSES MCLR   //Reset      habilitado
#FUSES NOLVP  //     prog.  baixa  voltagem     desabilitado
#FUSES NOCPD  //
#use   delay(clock=4000000)

void   main() {
              int    i;
              setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
              setup_timer_1(T1_DISABLED);
              setup_timer_2(T2_DISABLED,0,1);
              setup_comparator(NC_NC_NC_NC);
              setup_vref(FALSE);

              for(   i      =      0;     i      <      256;   i      ++
){
                            output_b(     i      );
                            delay_ms(     300    );
                            if(    i      ==     255    )      i      =
0;
              };
}
```

O que acrescentamos foi a variável “i” inteira de 8 bits, no início da função main, onde deve ficar as declarações de variáveis locais e o laço for, onde contará de 0 até 255 (0xFF). Observe aqui que este laço funcionará indefinidamente porque a condição de finalização (i<256) nunca será atingida porque quando chegar a 255 a declaração if(i==255) fará com que a variável “i” seje zerada reiniciando o laço e assim indefinidamente.

Também poderíamos ter escrito:

```
while( i      <      256    ){
                     //estabelece a      condição      de     loop
                     output_b(     i      );
                            //coloca     o      byte   no     port   B
```

```
                delay_ms(    300   );
            //aguarda    300    milésimos    de    seg.
                i     ++;

//incrementa i      em     uma    unidade
                if(   i      ==     255    )      i      =      0;
       //zera i      se     i      =      255
```

```
};
                    //fnal do     bloco  while
```

No laço “for” gastamos 19 linhas e no laço “while” 17 linhas, poderíamos ter economizado mais de utilizássemos o comando trisb para setar a direção do port B. Para tal devemos colocar no início do arquivo, logo abaixo da definição do clock para delay as seguintes definições:

```
#use   fast_io(a)
#use   fast_io(b)
#defne trisa         0b00000000          //     0      output,       1
input
#defne trisb         0b00000000          //     ou     #defne trisb  0x00
```

Assim, teríamos para o primeiro laço, 16 linhas e para o segundo 14 linhas. Esta economia de memória pode ser significativa se o soft for maior, com várias movimentações nas portas do PIC com mudanças freqüentes de direções.

A diretiva #use fast_io(x), afeta como o compilador gerará o código para as instruções da entrada e de saída. Esta diretiva orientadora faz efeito até que outra diretiva de I/O seja encontrada. O método rápido (fast_io) faz com que as intruções I/O possam ser realizadas sem programar a direção dos registradores, o usuário então deverá ajustar a direção dos registradores através da função set_tris_X( ).

Já a diretiva #use fixed_io(b_outputs=PIN_B2, PIN_Bx), fará com que o compilador gere o código para fazer um pino de I/O input ou output cada vez que é usada. Os pinos são programados de acordo com a informação nesta diretiva, mantendo um byte na RAM para definição I/O padrão. E a diretiva #use standard_io(B) é o método padrão, e faz com que o compilador gere o código para fazer o pino entrada ou saída cada vez que for utilizado. este s dois últimos métodos dispensa a utilização da função set_tris_x().

Para referenciar a um pino no port A ou B o leitor poderá utilizar o método padrão que é PIN_Ax ou PIN_Bx, onde x representa o bit do port. Assim para colocar o pino B2 em nível alto pode-se escrever output_high(PIN_B2) ou output_low(PIN_B2) para nível baixo e para ler o status do pino pode-se utilizar x =

input(PIN_B2).

Porém este método embora fácil, podemos melhorar a escrita e diminuir o código utilizando a referência direta ao bit, definindo no início do programa um byte com o endereço da porta e depois nomeando o bit do byte endereçado. veja:

```
#byte  Ra     =      0x05          //defne       o      byte   Ra     com
o      endereço      do     portA
#byte  Rb     =      0x06          //defne       o      byte   Ra     com
o      endereço      do     portB
                     //
#bit   sensor =      Ra.1          //
#bit   rele                =      Ra.2          //
#bit   led                         =      Rb.2          //
```

Agora nas funções de entrada e saída, faz-se referência direta ao bit. Para acender o led faz-se led = 1 e para apagá-lo faz-se led = 0. de forma idêntica para ler o status do pino (bit) fazemos avaliação direta no bit como if(sensor) rele = 1;

```
while( true   ){
//loop permanente
              if(    sensor )      {                                    //if
Ra.1   =      1
                                          rele   =      1;
                     //liga o      relê   em     Ra.2
                                          delay_ms(400);       //aguarda
400    milisegundos
                                          rele=  0;
                     //desliga     o      relê   em     Ra.2
              };
              led    =      1
                                          //acende      o      led    em     Rb.2
              delay_ms(     250    );                    //aguarda     250
milisegundos
              led    =      0
                                          //apaga       o      led    em     Rb.2
              delay_ms(     250    );                    //aguarda     250
milisegundos
};
```

Com referência ao exemplo anterior onde fizemos output_b((i), agora escrevemos Rb = i;

Este modo de escrita, acredito, torna o programa mais claro na sua forma de raciocínio e de escrita.

Capítulo

6

Temporizadores - timers

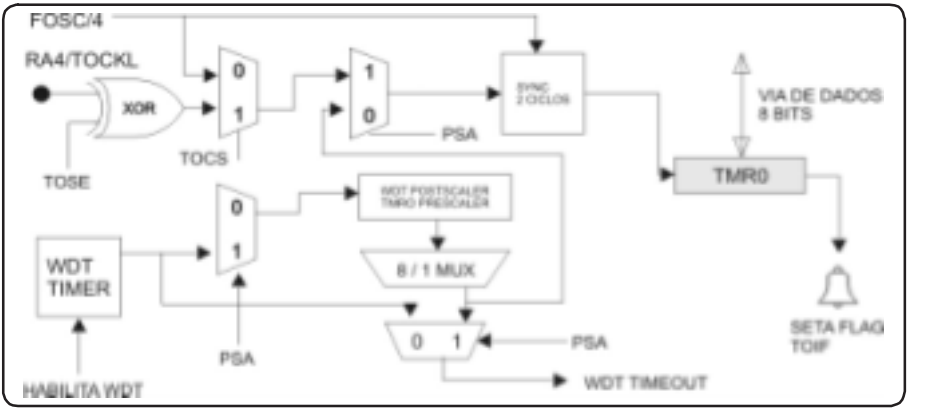
6.1- Temporizador TMR0

O timer0 é um contador/temporizador de 8 bits de leitura ou escrita, com seleção de sinal de clock interno ou externo, com possibilidade de divisão do sinal (prescaler), e geração de interrupção de transbordamento, na passagem de FFh para 00h.

O funcionamento do timer0 é controlado pelos bits TOSE, TOCS e PSA, do registro OPTION no endereço 81h e 181h.

A principal função deste registrador é controlar o comportamento do temporizador principal. O bit PSA tem a função de ligar o divisor de freqüência ao TMR0 ou ao WDT sendo que os bits PSA0, PSA1 e PSA2 selecionam a faixa em que o divisor de freqüência (prescaler) irá trabalhar. Se o prescaler estiver ligado ao timer0 não poderá ser utilizado no WDT e vice-versa.

O bit TOCS direciona a um multiplexador a procedência



dos sinais de clock que podem ser externos no pino RA4/TOCL atuando como contador ou interno sincronizado com o clock do microcontrolador (Fosc/4) atuando como temporizador. Como contador o bit TOSE seleciona se o pulso será sensível a subida ou a descida.

Este timer comporta-se como um registro de propósito especial ocupando a posição 01h e duplicado em 101h, o qual pode ser lido ou escrito a qualquer momento, por estar diretamente conectado a via de dados (Bus). Quando se escreve um novo valor sobre o TMR0 para uma nova temporização, há um abraso de dois ciclos de clock para sincronismo com o clock.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

bit	descrição
7 e 6	sem efeito
TOCS	Fonte de clock para o timer 0 1 = timer0 atua como contador por transição em Ra4/Tockl 0 = timer0 atua como temporizador
TOSE	Fonte de pulso para o tmer 0 1 = incrementa pulso ascendente para descendente 0 = incrementa pulso descendente para ascendente
PSA	prescaler 1 = prescaler atua em WDT 0 = prescaler atua em TMR0
PS2, PS1, PS0	seleção do prescaler

bit 2, 1, 0	TMR0	WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
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

Uma interrupção para o timer0 é gerada quando o contador

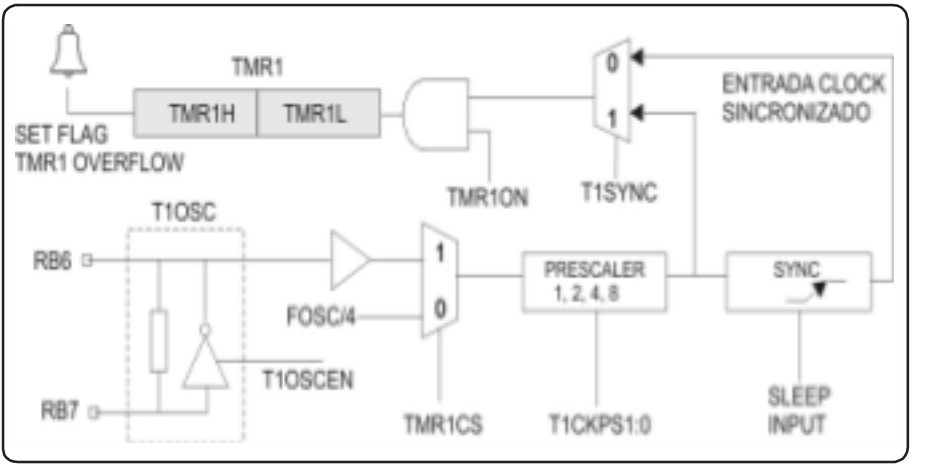
ou temporizador estoura pela passagem do FFh para 00h, levando o bit 2 do registro INTCON (endereço 0Bh, 8Bh, 10Bh e 18Bh) chamado TOIF para 1 sinalizando estouro de contagem. Este bit deve ser zerado via software para uma nova contagem. O bit 5 do registro INTCON chamado TOIE sinaliza uma interrupção por estouro de contagem, passagem de FF para 0. Se o tmr0 for acionado durante o SLEEP a interrupção não irá tirar o microcontrolador do SLEEP .

6.1- Temporizador TMR1

O timer1 ou TMR1 é um temporizador/Contador crescente de 16 bits, que faz a contagem de 0000h até FFFFh ou 65.535 decimal e para tanto utiliza dois registradores de 8 bits. Um armazena a parte mais significativa do byte (MSB) e chama-se TMR1H, o outro armazena a parte menos significativa do byte (LSB) sendo chamado de TMR1L.

O timer 1 pode ser utilizado como timer ou como contador, contando impulsos externos aplicados em RC0/T1OSO/T1CKI. sendo utilizado desta forma como captura e comparação.

A fonte de clock pode-se ser aplicada sobre o prescaler conforme os bits TICKPS0 e TICKPS1. A Fonte de impulsos externa pode pode estar ou não em sincronismo com o clock interno conforme



o bit T1SYNC, já a fonte interna sempre está em sincronismo com o clock interno, com duração mínima de Fosc/4.

Os registradores que controlam o funcionamento do timer1 são:

endereço	Registro	Bit	descrição
0Ch	PIR1	0 - TMR1IF	_ag de interrupção do timer 1
8Ch	PIE1	0 - TMR1IE	_ag de habilitação do timer1
10h	T1CON		

Descrição do registro T1COM

bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

bit	descrição
T1CKPS1, T1CKPS0	T1CKPS1, T1CKPS1 | prescaler 11 | 1:8 10 | 1:4 01 | 1:2 00 | 1:1
T1OSCEN	habilitação do oscilador 1 = habilitado 0 = desabilitado
T1SYNC	Sincronização com clock externo 1 = não sinconizado 0 = sincronizado
TMR1CS	Fonte de clock para o timer1 1 = clock esterno em Rb.6 sensível a subida 0 = clock interno (Fosc/4)
TMR1ON	Liga ou desliga o timer1 1 = ligado 0 = desligado

O timer1 pode operar de modo síncrono e asíncromo

No modo síncrono o timer1 desconecta-se do circuito de sincronismo quando o microcontrolador entra em SLEEP, Já no modo asíncromo o timer1 segue contando mesmo com a entrada em SLEEP, sendo o modo mais utilizado principalmente para gerar base de tempo para PWM, comparação e captura

Como base de tempo, este pode ser utilizado como oscilador

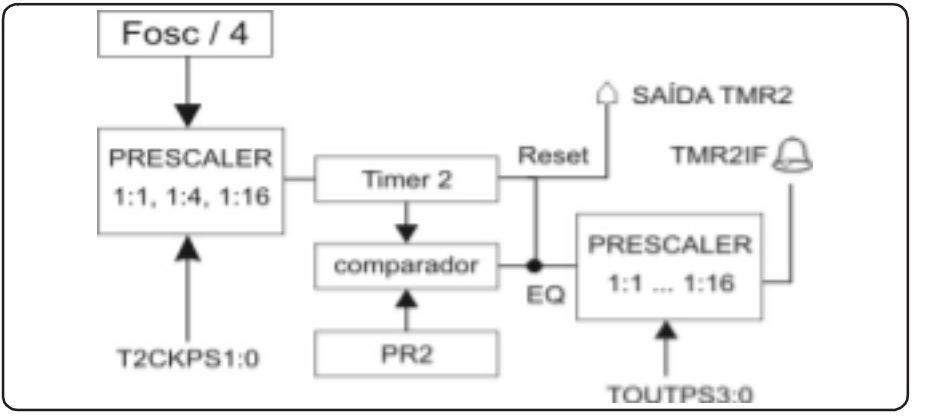
com cristal entre os pinos T1OSI e T1OSO, devendo ser habilitado pelo bit T1OSCEN. Pode operar em 32Khz com capacitor de 33pF, em 100Khz com capacitor de 15pF e em 200Khz com capacitor de 15pF.

6.2- Temporizador Timer2

O timer 2 ou TMR2 é um contador ascendente de 8 bits com prescaler e postscaler para usar como base de tempo de PWM.

O timer2 tem um registro de período de 8 bits chamado PR2 que produz uma saída EQ toda vez que a contagem de TMR2 coincide com o valor de PR2. Os impulsos de EQ são aplicados ao postscaler que podem dividi-lo em até 1:16, sendo depois aplicados no registro TMR2IF, gerando uma interrupção.

Uma outra alternativa ao uso do sinal EQ é a utilização em interface serial SSP



Os registradores que controlam o funcionamento do timer2 são:

endereço	Registro	Bit	descrição
0Ch	PIR1	1 - TMR2IF	_ag de interrupção do timer 2
8Ch	PIE1	1 - TMR2IE	_ag de habilitação do timer 2
12h	T2CON		

Descrição do registro T2COM

bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
ToutPS3	ToutPS2	ToutPS1	ToutPS0	TMR2ON	T2CKPS1	T2CKPS0

bit	descrição
TOUTPS3, TOUTPS2, TOUTPS1 e TOUTPS0	TOUTPS3..TOUTPS0 / postscaler 0000 / 1:1 0001 / 1:2 0010 / 1:3 0011 / 1:4 0100 / 1:5 0101 / 1:6 0110 / 1:7 0111 / 1:8 1000 / 1:9 1001 / 1:10 1010 / 1:11 1011 / 1:12 1100 / 1:13 1101 / 1:14 1110 / 1:15 1111 / 1:16
TMR2ON	habilitação do timer 2 1 = habilitado 0 = desabilitado
T2CKPS1, T2CKPS0	Prescaler do sinal de clock T2CKPS1, T2CKPS0 / prescaler 00 / 1:1 01 / 1:6 10 / 1:16

6.3- Configuração do Timer

A forma de ajustar os timers no compilador ”CCS” é utilizando a função setup_timer_x(modo) O x representa um dos timer e o parâmetro modo as constantes internas do compilador.

Como os timers são posições de RAM, podemos ler e escrever nestes à vontade. No timer0 utilizamos as funções GET_TIMER0(), SET_TIMER0(x) e configuramos como temporizador com a função Setup_Timer_0(modo), onde modo pode ser definido como uma das constantes da tabela:

constante	descrição.
RTCC_INTERNAL	fonte de clock interno
RTCC_EXT_L_TO_H	fonte de clock externo sensível a subida
RTCC_EXT_H_TO_L	fonte de clock externo sensível a descida
RTCC_DIV_2,	prescaler 1: 2
RTCC_DIV_4	prescaler 1:4
RTCC_DIV_8	prescaler 1:8
RTCC_DIV_16	prescaler 1:16
RTCC_DIV_32	prescaler 1:32
RTCC_DIV_64	prescaler 1:64
RTCC_DIV_128	prescaler 1:128
RTCC_DIV_256	prescaler 1:256

ex. setup_timer_0 (RTCC_DIV_8 | RTCC_EXT_L_TO_H);

Para o timer 0 como contador temos que utilizar a função setup_counters (rtcc_state, ps_state), onde rtcc_state pode ser definido como RTCC_INTERNAL, RTCC_EXT_L_TO_H ou RTCC_EXT_H_TO_L e ps_state como uma das constantes da tabela acima. Também é possível especificar aqui, as constantes de configuração do WDT, porém acho mais prático utilizá-las com a função Setup_WDT, que veremos mais a frente.

ex. setup_counters (RTCC_INTERNAL, WDT_18MS);

Tipicamente você necessitará ajustar o timer a fim de fazer o temporizador gerar uma freqüência ou medir um período de tempo. Vamos supor que você precise marcar 5 milisegundos, utilizando um clock de 4Mhz, temos inicialmente que encontrar a resolução do timer, isto é o tempo que ele leva para transbordar (passar de 256 para 0).

resolução = 1/clock * (256 * RTCC_DIV)

0,000.001 * 256 * 4 = 0,001.024 segundos, é pouco

0,000.001 * 256 * 16 = 0,004.096 segundos, é pouco

0,000.001 * 256 * 32 = 0,008.192 segundos, passou !

Com prescaler em 16 faltou um pouco e com prescaler em 32 passou um pouco, então calculamos um valor de inicialização do timer para o tempo desejado.

255-TMR0 = Tempo desejado / (RTCC_DIV)

255-TMR0 = 5000 us / 32 = 156,25

TMR0 = 255 - 156 = 100

Iniciamos o TMR0 com 100, este contará até 256 em 32 vezes de 156 microsegundo, totalizando 4.992 microsegundos que está muito próximo do valor desejado.

De forma idêntica podemos ler no timer 1, com a função GET_TIMER1() , escrever com a função SET_TIMER1(x) e configurá-lo com a função setup_timer_1 (modo), onde “modo” pode ser uma ou mais constantes definidas como:

constante	descrição.
T1_DISABLED	timer 1 desabilitado
T1_INTERNAL	fonte de clock interno
T1_EXTERNAL	fonte de clock externo
T1_EXTERNAL_SYNC	prescaler 1: 2
T1_CLK_OUT	prescaler 1:4
T1_DIV_BY_1	prescaler 1:1
T1_DIV_BY_2	prescaler 1:2
T1_DIV_BY_4	prescaler 1:4
T1_DIV_BY_8	prescaler 1:8

ex.

timer1 como termporizador interno e prescaler 1:8

```
setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8 );
```

timer1 como contador externo e prescaler 1:8

```
setup_timer_1( T1_EXTERNAL | T1_DIV_BY_8);
```

As equações do timer0 aplicam-se também ao timer 1, apenas lembrando que o timer1 é um registrador de 16 bits (65.536), então resolução do tempo = 1/(Fosc/4)*65536*T1_DIV

Como os anteriores, o timer 2, também pode ser lido ou escrito utilizando GET_TIMER2() , escrito com SET_TIMER2(x) e configurado com a função, setup_timer_2 (modo, periodo, postscaler), que apresenta três parâmetros.

modo: especifica o divisor de clock interno, sendo T2_DISABLED, T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_16

período: de 0 a 255 determina quando o valor do clock é rezetado.

postscaler: é um número de 1 à 16 que determina quando a interrupção ocorrerá

ex. setup_timer_2(T2_DIV_BY_1, 3, 2);

//a 4 mhz, a resolução do timer2 será de 1us com overflow a cada 4us, e interrupção a cada 8us.

```
#include     <16F628A.h>
#FUSES NOWDT                                    //No  Watch  Dog
Timer
#FUSES RC
//Resistor/Capacitor       Osc    with   CLKOUT
#FUSES PUT                                                 //Power
Up     Timer
#FUSES NOPROTECT           //Code not    protected    from   reading
#FUSES BROWNOUT                  //Reset       when   brownout
detected
#FUSES MCLR                                             //Master
Clear  pin    enabled
#FUSES NOLVP                                    //No   Low    Voltage
Programming
#use   delay(clock=4000000)

#bit   TOIF   =      0x0B.2       //Flag de     sinalização   de
estouros     TMR0
#byte  Ra     =      0x05                              //endereço    do
portA
#byte  Rb     =      0x06                              //endereço    do
```

```
PortB
#bit   saida  =      Rb.0               //defnição   do     bit
#use   fast_io(a)
#use   fast_io(b)
#defne trisb         0b00000000         //     todos  os     bits   como
saída
```

```
void   main() {
               //resolução   =      1/clock       *      (256   *
RTCC_DIV)
               //0,000.001*(256*2) =      0,000.512     (512uS)
               setup_timer_0(RTCC_INTERNAL       |      RTCC_DIV_2);
               setup_timer_1(T1_INTERNAL  |      T1_DIV_BY_1);
               setup_timer_2(T2_DISABLED,0,1);
               setup_comparator(NC_NC_NC_NC);
               setup_vref(FALSE);
               //
               set_tris_b(   trisb  );           //ajusta      direção
dos    bits    do     portB
               //
               set_timer0(   0      );                          //zera o
timer0
               while( true   ){
//loop contínuo
                                    while( TOIF   );
//aguarda     transbordamento       do     timer0
                                    TOIF          =      0;
                              //zera o      indicador     de
transbordamento
                                    saida  =      1;
                        //nível     alto   no     pino
                                    while( TOIF   );
//aguarda     transbordamento       do     timer0
                                    TOIF          =      0;
                              //zera o      indicador     de
transbordamento
                                    saida  =      0;
                        //nível     baixo  na     saída
               };
}
```

Neste simples exemplo temos um gerador de onda quadrada a 488Khz, onde o loop while irá aguardar o timer0 transbordar o que ocorre a cada 0,000.512 segundos. Veja que ficamos aguardando o indicativo da flag de transbordamento, uma outra forma de fazer a mesma coisa seria monitorar a que momento o timer passa pelo ponto zero.

```
set_timer0(   0      );

//zera o      timer0
while( true   ){

//loop contínuo
                                    while( get_timer0() ==     0      );
//aguarda     transb do     tmr0
                                    saida  =      1;
```

```
                        //nível       alto   no     pino
                                      while( get_timer0() ==     0      );
  //aguarda     transb do     tmr0
                                      saida  =      0;

                        //nível       baixo  na     saída
                    };
  };
```

Quanto tempo você leva para apertar um botão?, vamos ver?

```
  #include        <16F628A.h>

  #FUSES  NOWDT,  RC,     PUT,    NOPROTECT,      BROWNOUT,       NOMCLR, NOLVP,
  NOCPD
  #use    rs232(baud=9600,parity=N,xmit=PIN_B2,rcv=PIN_B1,bits=8)
  #use    delay(clock=4000000)
```

```
#byte   Ra      =       0x05
#byte   Rb      =       0x06
#bit    continua        =       Rb.1
#bit    botao                   =       Rb.1
#use    fast_io(a)
#use    fast_io(b)
#defne  trisa           0b00000000              //
#defne  trisb           0b00000011              //
void    main()  {
                //      ((      clock/4 )       /       RTCC_DIV        )       =
(4.000.000/4)/256)      3.906
                //      1       /       3.906   =       0,000.256       us      de
resolução
                //      (       256us   *       255     )       =       65.5ms. para
overfow
                Setup_Counters(RTCC_INTERNAL    |       RTCC_DIV_256);
                Setup_timer_1(T1_DISABLED);
                Setup_timer_2(T2_DISABLED,0,1);
                Ssetup_comparator(NC_NC_NC_NC);
                Setup_vref(FALSE);
                //
                set_tris_b(     trisb   );              //ajusta        direção dos
bits    do      portB
                //
                while(  !       continua        ){
                                        while(  !       botao   );
                        //aguarda       o       botão   descer
                                                                set_rtcc(       0
);                                              //zera  o       timer   0
                                        while(  botao   );
                                //aguarda       o       botão   subir
                                                                time    =
get_rtcc();             //pega  o       tempo   do      botão

                                        printf("Tempo   do      Botão   =       %u
ms.",   time);
                };
};
```

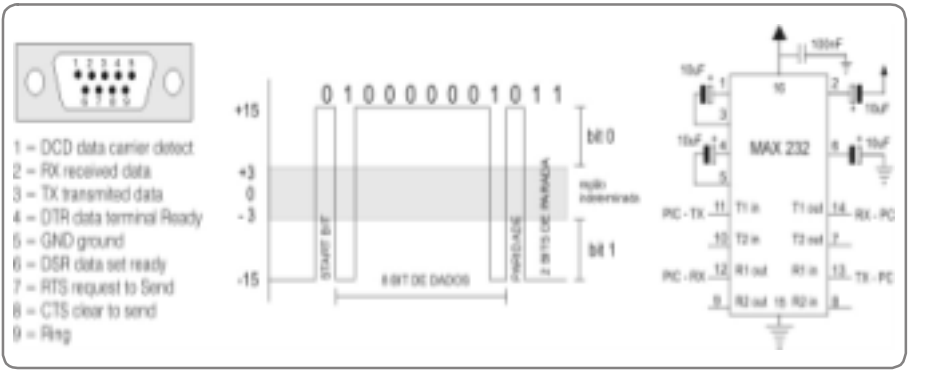
Capítulo

7

Comunicação

7.1 Comunicação Serial RS232

No exemplo anterior, utilizamos a função “printf” para comunicarmos com um microcomputador. Na verdade esta função é a principal função para direcionamento de saída de dados, onde comunicamos com dispositivos externos. A comunicação segue utilizando as regras de comunicação conhecida como RS232, onde um pacote de dados, inicia-se com um bit de start, seguido do byte a ser enviado ou recebido, um bit de paridade (par ou impar) e dois bits de parada. A lógica neste caso é invertida, ou seja, o bit 1 é alguma coisa entre -3 e -15 volts e o bit 0 é algo em torno de +3 a +15volts. Entre +3volts e -3volts há uma região indeterminada onde o hardware de comunicação serial conhecido como USART não atua. Os parâmetros extremos (+15 e -15) são limites máximos, assim sendo, não há um valor fixo para este parâmetro e sim um valor máximo. É comum encontrar portas seriais com valores entre

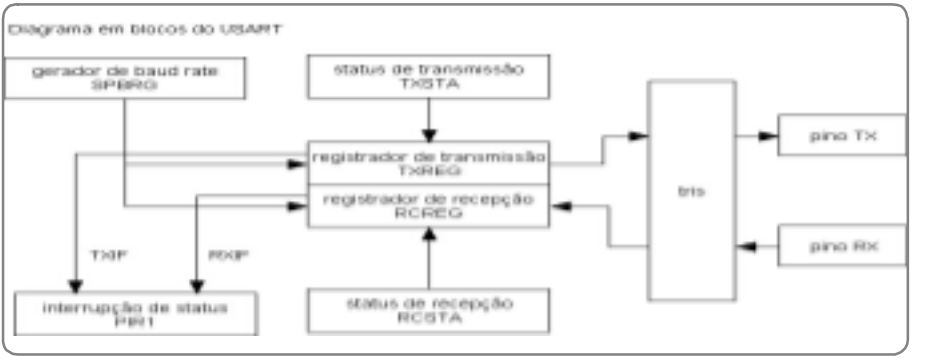


+8/-8 e +12/-12 e em notebooks é comum o valor de +5/-5.

Para interfacear, e ajustar os níveis lógicos além de fazer sua inversão para a lógica correta, utiliza-se o “MAX232” ou semelhante, que se encarrega de ajustar estes parâmetros.

A comunicação é feita em “pacotes”, sendo formado por:

Start bits	sempre bit 1
bits de dados	normalmente 8 bits
bit de paridade	“none” se não é requerida detecção de erro “odd” ou “even” se usar detecção de erro
bit de parada	1 ou 2 bits



Paridade de dados pode se impar “odd” ou par “even” ambas apresentando o mesmo nível de detecção de erro, mas não a correção destes, quando se utiliza o pacote de 8 bits adiciona-se um bit extra no pacote de dados de modo que a quantidade de bits 1 no byte seja par, ou ímpar.

O BAUD é um bit por segundo, assim cada bit de dado tem o tempo de 1/(baud rate) trabalhando com 9600 bps temos 104uS por bit (1/9600).

A configuração de como o pic tratará a lógica RS232 é feita com a definição #use RS232( ), que informa ao compilador para ajustar internamente o USART do pic com os parâmetros especificados na definição. São eles:

Strean ID	cria um canal de comunicação de_nido em _id_
BAUD=x	especi_ca a velocidade de transmissão de dados
XMIT=pin	de_ne o pino (bit do port) para transmissão
RCV=pin	de_ne o pino (bit do port) para recepção
PARITY=X	de_ne o bit de paridade, N (nenhum), E(par), O(impar)
BITS =X	byte com x bits, pode ser 5, 6, 7, 8, 9
INVERT	inverte a lógica de comunicação, caso não use max232
restart_wdt	restart WDT durante as rotinas de comunicação
disable_ints	desabilita interrupções durante as rotinas de comunic
long_data	getc()/putc() aceita um inteiro de 16 bits. somente para byte de 9 bits
enable=pino	especi_ca um pino para habilitar transmissão, util em rs485
debugger	indica este stream será usado para enviar/receber dados a uma unidade CCS ICD
errors	faz com que o compilador guarde os erros na variável RS232_ERRORS
_oat_high	Usada para saídas com coletor aberto
return=pin	use com FLOAT_HIGH e MULTI_MASTER este pino é usado para ler o retorno do sinal. O padrão para FLOAT_HIGH é XMIT e para MULTI_MASTER é RCV.
multi_master	use pino RETURN para determinar if outro stream trams. ao mesmo tempo, Não pode ser usado com o UART.
sample_early	getc(), normalmente faz a captura no meio da amostra. esta opção faz com que seje no _nal. Não pode ser usado com o UART.
force_sw	usa software no lugar do hardware USART, ou quando se especi_ca os do hardaware.
BRGH1OK	Permite baixas velocidades de transmisão em chips que apresentão prob de transmissão

A maioria destes parâmetros, dificilmente serão utilizados, sendo que a definição comum é:

```
#use   rs232(baud=9600,parity=N,xmit=PIN_B2,rcv=PIN_B1,bits=8)
```

Antes de utilizar a diretiva #USE RS232, devemos utilizar a diretiva #USE DELAY(clock) e se não utilizar a definição padrão do USART do PIC, deve-se utilizar as diretivas FIXED_IO ou FAST_IO antes de usar a diretiva #USE RS232.

```
#use   fxed_io(b_outputs=pin_b2,  b_inputs=pin_b1)
#use   delay(Clock=4000000)
#use   rs232(baud=9600,    xmit=PIN_B2,  rcv=PIN_B1,   bits=8)
```

A comunicação pode ser sincronizada ou desincronizada, esta última a mais utilizada nas aplicações. A velocidade de transmissão da interface é controlada pelo gerador de baud rate SPBR que mediante um relógio interno, pode utilizar 16 (alta velocidade) ou 64 (baixa velocidade) pulsos por bit transmitido. SPBRG é um registro de 8 bits, que permite valores entre 0 e 255. O baud rate pode ser calculado pela fórmula:

BR (baud rate) = FOSC/(S*(SPBRG+1)) onde S = 16 ou 64

Modificando a fórmula anterior podemos calcular o valor de SPBRG com:

SPBRG = [FOSC/ (S * BR)] – 1

A taxa de erro pode ser calculada pela razão entre o baud rate calculado e o baud rate desejado.

taxa_erro = (baud rate calculado - baud rate desejado) / baud rate desejado.

Não especifique um baud rate maior que a capacidade de clock do circuito, tenho utilizado com sucesso um clock de 13 mhz. e obtenho ótimos resultados com uma taxa de 38k bps.

Havendo necessidade pode-se mudar a taxa de baud rate durante a execução do programa com a função SET_UART_SPEED(baud) .

```
//-------------------------------------------------------
//     confguração  inicial      do     microcontrolador
//-------------------------------------------------------
void   confg_pic()  {
             baud_rate           =      read_EEPROM(  0x05   );
             switch(baud_rate){
                                 case   0x01   :      set_uart_speed(
       1200   );     break;
                                 case   0x02   :      set_uart_speed(
       4800   );     break;
                                 case   0x03   :      set_uart_speed(
       9600   );     break;
                                 case   0x04   :      set_uart_speed(
```

```
14400 );     break;
                                    case   0x05   :      set_uart_speed(
28800 );     break;
                                    case   0x06   :      set_uart_speed(
38400 );     break;
```

```
                                        case   0x07   :      set_uart_speed(
57600  );     break;
               };
};
```

As funções utilizadas para comunicação serial estão descritas nesta tabela e logo após o tópico sobre correção de erros, veremos um exemplo prático.

função	descrição
getch(), getchar()	aguarda um caracter no buffer de recebimento
gets(char *string)	lê uma seqüência de caracteres (string) até 0x0D
putc(), putchar()	envia um caractere ao buffer de transmissão.
PUTS(string)	envia uma seqüência de caracteres ao buffer de transmissão e avança linha (13) 0x0D e avanço de linha (10) 0x0A
printf(string, valor)	envia uma seqüência de caracteres formatados.

formato	descrição
C	Caracter
U	inteiro sem sinal
x	inteiro em Hexadecimal min_sculas
X	inteiro em Hexadecimal mai_sculas
D	inteiro com sinal
%e	real no formato exponencial
%f	real (Float)
Lx	inteiro longo em Hex min_sculo
LX	inteiro longo em Hex mai_sculo
Lu	decimal longo sem sinal
Ld	decimal longo com sinal

KBHIT()	retorna TRUE se houver um caractere no buffer de recepção

Protocolo de correção de erro é o nome dado a rotina de detecção de erros de comunicação que pode ocorrer por n motivos, como interferência eletromagnética, erro no hardware, aquecimento, defeitos em componentes, enfim o que não falta é probabilidades de erros. Assim fórmulas de detecção de erros

são importantes na comunicação. Alguns métodos verificam um bloco de dados com polimônimos apropriados gerando byte de verificação.

CRC= verificação cíclica de redundância

LRC = checagem de redundância longitudinal

BCC = caracter de checagem de bloco.

Verificação de Redundância Cíclica ou CRC

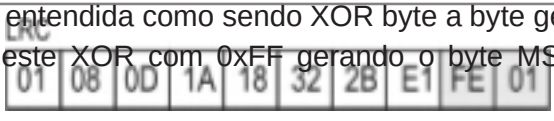
É um método de detecção polinomial que permite a detecção de praticamente toda ocorrência de erros, utilizando polimôminos padronizados como:

CRC 16 = X16 + X15 + X2 +1

CRC CCITT = X16 + X12 + X5 +1

```
//------------------------------------------------------
//     CRC    type......:   ISO/IEC      13239
//     Polynomial....:      X16    +      X12    +      X5     +      1
=      '8408'
//     Direction.....:      Backward
//     Preset........:      'FFFF'       Residue.......:'F0B8'
//------------------------------------------------------
static long   crc16_compute(      int    *      buffer,       int
buffer_size){
       int16  crc16  =      0xFFFF;
       int    i;
       for    (i     =      0;     i      <      buffer_size;  i++)   {
                     int    j;
                     crc16  =      crc16  ^      ((unsigned
int)buffer[i]);
                     for    (j     =      0;     j      <      8;     j++)
{
                                   if     (crc16 &      0x0001)
crc16  =      (crc16 >>     1)     ^      0x8408;
                                   else   crc16  =      (crc16 >>     1);
                     }
       }
       return crc16;
}
```

Sempre utilizo a verificação longitudinal de redundância LRC, que pode ser entendida como sendo XOR byte a byte gerando o byte LSB, e este XOR com 0xFF gerando o byte MSB da verificação



```
//-------------------------------------------------------
//LRC  longitudinal redundance   ciclical
//parâmetros: ponteiro     do     buffer de     dados
//
      quantidade   de     bytes  do     buffer de     dados
//
      modo   de     checagem     1      adicionar     0      checar
//-------------------------------------------------------
static short  lrc(   int    *buffer,      int    size,  int    modo   ){
              int    x;
              int    lsb    =      0;
              int    msb    =      0;
              restart_wdt();
              //para cálculo      desconsidera os     dois   últimos
bytes
              for(   x      =      0;     x      <=     (size  -      2)
;      x++    ){
                                  lsb    ^=     buffer[       x      ];
                     //     byte   lsb    XOR    buffer[x]
              };

                            //
              msb    =      lsb    ^      0xFF;
                            //     byte   msb    XOR    lsb
              if(    modo   ==     1){
                                  //     adiciona
                                  buffer[size   -1]    =      lsb;   //
byte   lsb
                                  buffer[size                 ]      =
msb;   //     byte   msb
              }
              else

                     //checagem
                                  if(    buffer[size   -1]    ==     lsb
&&

buffer[size                 ]      ==     msb    )      return TRUE;
                                  else   return FALSE;
}
```

Um protocolo de comunicação é uma forma pré-definida para comunicação entre dispositivo, que a utiliza em ambas as direções. e que visa estabelecer regras para se comunicar. Definiremos aqui um protocolo de comunicação, derivado do protocolo utilizado em RFID, o qual diferencia-se apenas pela quantidade de bytes transmitidos/recebidos. Na Identificação por rádio freqüência RFID pode-se enviar até 1000 bytes para o transponder. Este protocolo inicia-se com o
cabeçalho de dados (01h),

seguido pela quantidade
de bytes que serão envia-
dos, de flags de controle e
ou dados e finalmente pela checagem LRC.

Neste exemplo vamos implementar diversas rotinas para criar com o microcontrolador um canal de comunicação de perguntas e respostas utilizando diversas funções, ainda não exemplificadas que serão comentadas logo mais.

SOF	LEN	CMD	DADO	LRC	
01	06	5C	00	5B	A4

Abra o compilador e Inicie um novo projeto; utilize o "project/new/pic wizard", faça os ajustes do modelo, clock 4Mhz como mostra a figura ao lado. Baixe o soft do nosso site e cole na janela do editor, compile e posteriormente grave o microcontrolador com o IC-Prog. baixe também nosso utilitário desenvolvido para este exemplo para controlar o pic que pode ser interfaceado com a nossa placa experimental ou construída a partir do esquema abaixo.

```
/*
---------------------------------------------------------------
              Projeto........:    programação  de    microcontrolador
pic
              Versão:........:    1.00
              Modelo.........:    16F628A
              Descrição......:    comunicação
---------------------------------------------------------------
*/
       //-------------------------------------------------------
       //     hard   defnitions
       //-------------------------------------------------------
#include      <16F628A.h>
       //
#FUSES WDT                                          //Watch      Dog
Timer  bilitado
#FUSES XT
//oscilador   cristal      <=     4mhz
#FUSES PUT                                          //Power      Up
Timer
#FUSES NOPROTECT    //sem  proteção     para   leitura      da
eprom
#FUSES BROWNOUT             //Resetar     quando detectar      brownout
#FUSES NOMCLR                      //Reset      desabilitado
#FUSES NOLVP                              //prog.      baixa  voltagem
desabilitado
#FUSES NOCPD                              //sem  travar o      chip

#use   fast_io(a)
#use   fast_io(b)
#use   delay(clock=4000000,RESTART_WDT)
```

```
#use rs232(baud=9600,parity=N,xmit=PIN_B2,rcv=PIN_
B1,bits=8,RESTART_WDT)
      //
      //-------------------------------------------------------
```

```
       //     defnição     das    portas I/O    ports
       //-------------------------------------------------------
#byte  Ra     =      0x05
#byte  Rb     =      0x06
       //
#bit   led_r               =      Rb.4          //     led    do
sistema       vermelho
#bit   led_g               =      Rb.5          //     led    do
sistema       verde
       //
#defne trisa         0b00000000          //     0      output
#defne trisb         0b00000010          //     1      input
       //
       //-------------------------------------------------------
       //defnições  internas
       //-------------------------------------------------------
#defne FIRWARE_VERSION            0x01   //versão      do     software
#defne FIRWARE_RELEASE            0x05   //release
#defne PACKET_SOF                                                0x01
//cabeçalho   do     pacote
#defne VERIFICA
0X00   //
#defne ADICIONA
0X01   //
#defne CMD_OK
       0x01   //comando     com    sucesso
#defne tamanho_buffer                    30                    //30
bytes
#defne ERRO_LRC
0x08   //reporta     erro   de     lrc
       //-------------------------------------------------------
       //defnição    de     endereços     da     eeprom
       //-------------------------------------------------------
#defne ENDERECO_FLAG                     0x01   //byte de     opção  para
uso    geral
#defne ENDERECO_COR_LED    0x02   //cor  do     led    utilizado     no

sistema
#defne ENDERECO_TEMPO                    0x03   //quant       de     inc.
de     tmr1
       //
       //-------------------------------------------------------
       //defnição    dos    comandos      enviados      pelo   computador
       //-------------------------------------------------------
#defne CMD_VERSAO                                                0x3A
//envia       a      versão so     soft   do     pic
#defne CMD_LED_TIME                                 0x3B   //ajusta      o
tempo  do     led
#defne CMD_COR_LED                                        0x3C
//especifca   a      cor    do     led
#defne CMD_RAM
       0x4A   //leitura     da     memória       ram
#defne CMD_LE_EEPROM                     0x4B   //leitura     da
```

```
memória      eeprom
#defne CMD_GRAVA_EEPROM    0x4C   //grava      um     byte   na
eeprom do
pic
#defne CMD_RESET_CPU                    0x5A   //reseta     a      cpu
#defne CMD_GRAVA_FLAG                   0x5B   //grava      fag
#defne CMD_STATUS                                          0x5C
//status      da     fag
#defne CMD_LIGA_LED                             0x5D   //ufa  tô     vivo
       //
       //-------------------------------------------------------
       //     variáveis
       //-------------------------------------------------------
enum   COR{RED,      GREEN, AMBER};      //     seqüência    0,     1,
2
       //
```

```
byte   buffer[tamanho_buffer];//buffer  de     recepção      e
transm.
int    conta_interrupcao;                                     //conta
nr     de     interrup.     do     tmr1
int    tempo_led;
                             //tempo       do     led    de     0
a      255    x
interrupção
int    cor_led;
                                          //cor  do     led
       //
int    fag;
                                                             //fag  de
sinalização  interna
       //estes       bits   do     byte   fag    sinalizam     eventos
no     soft
       //valores     válidos            0      =      não    1      =
sim
#bit   fag_use_led         =      fag.0  //usar led    ?
#bit   fag_led_on          =      fag.1  //led  está   aceso?
       //
       //-------------------------------------------------------
       //valor       padrão das    opções armaz. na     eeprom
       //-------------------------------------------------------
#ROM          0x2100 =      {0x00, 0x01,  0x01,  0x20,  0xFF}
       //
       //-------------------------------------------------------
       //     interrupção   do     timer  1
       //-------------------------------------------------------
#int_timer1
void   timer1_isr(void){
              restart_wdt();
              if(    fag_use_led   ){
                             if(    fag_led_on    ){
                                          //incrementa  o
contador     de     interrupções
                                                 conta_interrupcao
++;
                                                 //
                                                 if(
conta_interrupcao   >=     tempo_led     ){                 //

switch(      (COR)cor_led  ){

                     case   RED                  :      led_r  =      0;
break;

                     case   GREEN  :      led_g  =      0;     break;

                     case   AMBER  :      led_r  =      0;     led_g  =
0;     break;

};
```

```
conta_interrupcao   =     0;     //zera o      contador

fag_led_on    =     0;           //sinaliza    que    o      led


                                        //está apagado
                                                    };
                                    };
                };
}
        //
        //-------------------------------------------------------
        //     LRC    longitudinal redundance    ciclical
        //-------------------------------------------------------
static short  lrc(   int    *buffer,      int    size,  int    modo   ){
              int    x;
              int    lsb    =      0;
              int    msb    =      0;
              restart_wdt();
```

```
                //para cálculo      desconsidera os     dois   últimos
bytes
                for(   x      =     0;     x      <=     (size  -      2)
;      x++     ){
                                   lsb    ^=     buffer[      x      ];
                               //    byte   lsb    XOR    buffer[x]
                };

                                   //
                msb    =      lsb  ^      0xFF;
                                   //     byte   msb    XOR    lsb
                if(    modo   ==    1){
                                   //     adiciona
                                   buffer[size   -1]    =      lsb;
//     byte   lsb
                                   buffer[size                 ]      =
msb;            //     byte   msb
                }
                else

                       //checagem
                                   if(    buffer[size   -1]    ==     lsb
&&

buffer[size                ]       ==     msb    )      return TRUE;
                                   else   return FALSE;


}
        //
        //-------------------------------------------------------
        //     retorna       o      pacote para   o      PC
        //-------------------------------------------------------
void    responde(     int    len    ){
                int    j;
                restart_wdt();
                lrc(   buffer,       len,   ADICIONA      );
                for(   j      =      0;     j      <      len;   j++    ){
                                            printf(      "%2X", buffer[
j       ]       );
                };
}
        //-------------------------------------------------------
        //     acende o      led
        //-------------------------------------------------------
static void    liga_led(){
                restart_wdt();
                if(    fag_use_led   ){
//sistema       usa    led    ?
                                   fag_led_on    =      1;
                       //sinaliza   que    o      led    está    aceso
                                   conta_interrupcao=   0;//zera      0
contador        de     interrupções
                                   switch(       (COR)cor_led ){
                                                        case   RED
```

```
            :     led_r  =     1;    break;
                                                case
GREEN  :      led_g  =     1;    break;
                                                case
AMBER  :      led_r  =     1;    led_g  =     1;    break;
                                    };
                                    set_timer1(  50000  );
                                    //
                                    buffer[      0      ]      =
PACKET_SOF;                 //cabeçalho
                                    buffer[      1      ]      =      7;

//tamanho    do    pacote
                                    buffer[      2      ]      =
CMD_LIGA_LED;//informa    o     comando      exec
                                    buffer[      3      ]      =
cor_led;                                      //informa    a     cor
setada
                                    buffer[      4      ]      =
CMD_OK;                                             //informa
comando      ok    !
                                    responde(    7      );
            };
}
```

```
        //-------------------------------------------------------
        //salva      um     dado   no     endereco     especifcado
        //-------------------------------------------------------
static void   salva(int    endereco,     int    dado){
              write_eeprom( endereco,     dado   );
              //operação    de     escrita      pode   demorar
alguns milisegundos
              //a    fag    de     interrupção  de     escrita
completada    é      testada,
              //mas  resolví      dar    um     tempo  a      mais
              delay_ms(10);
}
        //
        //-------------------------------------------------------
        //     retorna      um     dado   gravado      na     eeprom
        //-------------------------------------------------------
int    recupera(     c      ){
              restart_wdt();
              return(      read_eeprom(  c      )      );

}
        //-------------------------------------------------------
        //     envia  a      versão so     soft   para   o      PC
        //-------------------------------------------------------
void   frware(){
              restart_wdt();
                                                   //wdt=0
              buffer[      0      ]      =      PACKET_SOF;
//cabeçalho   de     dados
              buffer[      1      ]      =      7;
                                                          //tamanho    do
pacote de     dados
              buffer[      2      ]      =      CMD_VERSAO;
//informa     o      comando      executado
              buffer[      3      ]      =      FIRWARE_VERSION;
//informa     a      versão
              buffer[      4      ]      =      FIRWARE_RELEASE;
//informa     o      release
              responde(    buffer[      1      ]      );
                           //informa    ao     PC
}
        //-------------------------------------------------------
        //     ajusta o      tempo  de     inc.   tmr1   para   led
        //-------------------------------------------------------
void   tempo( int    t      ){
              restart_wdt();
                                                   //wdt=0
              tempo_led    =      t;
                                                          //ajusta
variável      interna
              buffer[      0      ]      =      PACKET_SOF;
//cabeçalho   de     dados
              buffer[      1      ]      =      6;
```

```
                                                        //tamanho    do
pacote de     dados
              buffer[       2       ]       =       CMD_LED_TIME;//informa
o       comando       executado
              buffer[       3       ]       =       CMD_OK;
                            //diz   que     foi     executado,    ok!
              responde(     buffer[       1       ]       );
       //responde    ao      PC
              salva(ENDERECO_TEMPO,       t);                 //salva
na      EEPROM
}
       //-------------------------------------------------------
       //     ajusta a      cor     do      led
       //-------------------------------------------------------
void   AjustaCor(    int     t       ){
              restart_wdt();
                                                        //wdt=0
              cor_led       =       t;

//ajusta      a       variável      interna
```

```
              buffer[       0       ]       =       PACKET_SOF;
//cabeçalho   de      dados
              buffer[       1       ]       =       7;
                                                    //tamanho     do
pacote de     dados
              buffer[       2       ]       =       CMD_COR_LED;
//informa     o       comando       executado
              buffer[       3       ]       =       CMD_OK;
                            //diz   que     foi     executado
              buffer[       4       ]       =       t;
                                                    //informa     a
nova    cor
              responde(     buffer[       1       ]       );
       //responde     ao      PC
              salva(ENDERECO_COR_LED,     t);     //salva       na
EEPROM
}
       //-------------------------------------------------------
       //     retorna       o       conteúdo      da      RAM
       //-------------------------------------------------------
void   ram(   int     banco, int     inicio,       int     fm      ){
              int     nr_byte,      c,      j;

//variáveis
              restart_wdt();

//wdt=0
              nr_byte       =       fm      -       inicio;
                                                    //quantidade  de
bytes
              if(nr_byte    >=      30)     nr_byte       =       28;
//buffer      máx.
              buffer[       0       ]       =       PACKET_SOF;
                            //cabeçalho
              buffer[       1       ]       =       nr_byte       +       2;
                            //+     2       bytes   p/      lrc
              buffer[       2       ]       =       CMD_RAM;
                                                    //inf. comando
executado
              //preenche    o       buffer com      a       posição       da
memória       ram
              for(    j       =       3;      j       <       nr_byte;
j++){
                                    if(banco==0x01)     c       =
read_bank(1,  inicio++);
                                    if(banco==0x02)     c       =
read_bank(2,  inicio++);
                                    if(banco==0x03)     c       =
read_bank(3,  inicio++);
                                    buffer[       j       ]       =       c;
              };
              responde(     nr_byte       );      //responde    ao      PC
}
       //-------------------------------------------------------
       //     ajusta fag    interna       interna       de
```

```
sinalização
      //-------------------------------------------------------
void  gravaFlag(  int   c     ){
            restart_wdt();
                                                  //wdt=0
            fag   =     c;

                  //ajusta     a     variável     interna
            buffer[     0     ]     =     PACKET_SOF;
      //cabeçalho  de    dados
            buffer[     1     ]     =     6;
                                                        //tamanho
do    pacote de    dados
            buffer[     2     ]     =     CMD_GRAVA_FLAG;
//informa    comando     executado
            buffer[     3     ]     =     CMD_OK;
                              //informa    comando      ok!
            responde(   buffer[     1     ]     );
            //envia     ao    PC
            salva( ENDERECO_FLAG,     c     );
//salva      na    EEPROM
}
      //-------------------------------------------------------
      //    retorna     o     valor  da    fag   interna      de
sinalização
      //-------------------------------------------------------
void  retornaFlag(  void  ){
            restart_wdt();
                              //wdt=0
            buffer[     0     ]     =     PACKET_SOF;
//cabeçaho   de    dados
            buffer[     1     ]     =     6;
                                    //tamanho    do
pacote
            buffer[     2     ]     =     CMD_STATUS;
//informa    o     comando     executado
            buffer[     3     ]     =     fag;
                              //dado a     enviar (fag   de
sinal)
```

```
                responde(     buffer[       1       ]       );
//envia         ao      PC
}
        //---------------------------------------------------------
        //      responde        informando      ocorrência      de      erro
        //---------------------------------------------------------
void    erroLRC(void){
                restart_wdt();
                buffer[       0       ]       =       PACKET_SOF;
//cabeçalho     de      dados
                buffer[       1       ]       =       6;
                                                      //tamanho     do
pacote de       dados
                buffer[       2       ]       =       ERRO_LRC;
        //informa       ocorrencia      de      erro
                buffer[       3       ]       =       0;
                                                      //fag
                responde(     buffer[       1       ]       );
}
        //---------------------------------------------------------
        //      lê      um      byte    na      eeprom do       PIC     e
transmite       ao      PC
        //---------------------------------------------------------
void    ler_eeprom(   int     endereco      ){
                restart_wdt();
                buffer[       0       ]       =       PACKET_SOF;
        //cabeçalho     de      dados
                buffer[       1       ]       =       6;
                                                                    //tamanho
do      pacote de       dados
                buffer[       2       ]       =       CMD_LE_EEPROM;
//informa       comando         executado
                buffer[       3       ]       =       recupera(     endereco
);      //ret.  byte    da      EEPROM
                responde(     buffer[       1       ]       );
                //envia         ao      PC
}
        //---------------------------------------------------------
        //      Recebe um       byte    do      PC      e       grava   no
endereço        especifcado
        //---------------------------------------------------------
void    grava_eeprom( int     endereco,       int     dado    ){
                restart_wdt();
                                                      //zera WDT
                buffer[       0       ]       =       PACKET_SOF;
//cabeçalho     de      dados
                buffer[       1       ]       =       7;
                                                      //tamanho     do
pacote de       dados
                buffer[       2       ]       =       CMD_GRAVA_EEPROM;
//informa       cmd     executado
                buffer[       3       ]       =       endereco;
                                              //informa     enderdeço     EEPROM
                buffer[       4       ]       =       dado;
```

```
                                                        //informa    dado
da     EEPROM
              responde(     buffer[       1       ]       );
                            //envia       ao      PC
              salva( endereco,      dado);
              //grava       na      EEPROM

}
       //-------------------------------------------------------
       //limpa       o       buffer de       dados
       //-------------------------------------------------------
void   limpa_buffer(){
              int     j;
              j       =       0;
              //executa     enquanto      não     atingir       o       fnal
do     buffer
              while( ++j     <=      tamanho_buffer       ){     buffer[j]
=      0;     };
}
       //-------------------------------------------------------
       //     main   module
       //-------------------------------------------------------
```

```
void   main() {
              int    i;
              setup_timer_0(RTCC_INTERNAL);
              //
              //     tempo  =      1/(clock/4)   *      (65536-
set_timer1)   *      T1_DIV_BY
              //     t      =      0,000.001     *      (65536-50000)*1
=
              //     t      =      0,000.001     *      15536  =
0,0155
              //     tempo  máximo =      255    *      0,015  =      3,8
segundos
              setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
              //
              setup_timer_2(T2_DISABLED,0,1);  //timer      2
desabilitado
              setup_ccp1(CCP_OFF);
                                          //cap,comp,pwm      desab.
              setup_comparator(NC_NC_NC_NC);          //comparador
desab.
              setup_vref(FALSE);
                                                //volt.       referência
desab.
              //
              enable_interrupts(int_timer1);   //habilita   inter.
timer  1
              enable_interrupts(GLOBAL);
              setup_wdt(WDT_18MS);       //especifca  tempo  do     WDT
(padrão       18)
              //
              set_tris_a(  trisa  );           //direção    dos    bits
do     portA
              set_tris_b(  trisb  );           //direção    dos    bits
do     portB
              //
              Ra     =      0;           //limpa      todos  os     bits
do     portA
              Rb     =      0;           //limpa      todos  os     bits
do     portB
              //
              //     valores      iniciais     para   as     variáveis
              conta_interrupcao   =      0;
              fag                                     =      recupera(
ENDERECO_FLAG);
              cor_led                    =      recupera(
ENDERECO_COR_LED    );
              tempo_led    =      recupera(    ENDERECO_TEMPO      );
              //
              limpa_buffer();
              //
              while( true   ){
                                    buffer[0]     =      0;
                                    buffer[0]     =      getc();
                                    if(    buffer[0]     ==     PACKET_SOF
```

```
){
                                                                    //
       formato      do     pacote de     dados
                                                                    //     +---------
---------------------+
                                                                    //     |      SOF
|      len    |      cmd    |      DATA   |      LRC    |
                                                                    //     +-----+---
--+-----+-----+------+
                                                                    //     |
1            |            1            |            1            |
       n                   |            2            |      número de
bytes
                                                                    //     +---------
---------------------+
                                                                    //     |
0            |            1            |                         2            |
3..          |                         |      pos.   no
buffer
                                                                    //     +---------
---------------------+
                                                                    buffer[1]    =
getc();             //quant      de     bytes
                                                                    //-1   é      a
correção     buffer que    começa em     0
                                                                    for(   i     =
2;     i     <=     buffer[1]-1; i++    ){

buffer[      i     ]      =      getc();
             //resto      do     pacote

                                                                    };
```

```
                                                        //
                                                        if(    lrc(
buffer,       buffer[1]-1,  VERIFICA)){   //lrc  ok?

switch(       buffer[       2       ]      ){

              case   CMD_VERSAO                         :      frware()
                                                ;break;

              case   CMD_LED_TIME         :       tempo(buffer[3]);break;

              case   CMD_LIGA_LED         :       liga_led()
                     ;break;

              case   CMD_COR_LED   :      AjustaCor(buffer[3]);break;

              case   CMD_RESET_CPU :      reset_cpu()
       ;break;

              case   CMD_STATUS                         :
retornaFlag()               ;break;

              case   CMD_GRAVA_FLAG:      gravafag(     buffer[3]     )


                                ;break;

              case   CMD_LE_EEPROM :      ler_eeprom(   buffer[3]     )

                                ;break;

              case   CMD_RAM:

                                ram(   buffer[3],    buffer[4],
buffer[5]     )

                                ;break;

              case   CMD_GRAVA_EEPROM:

                                grava_eeprom(buffer[3],    buffer[4])
;break;

};
                                                        }
                                                        else   erroLRC();
                                                        //limpa       o
buffer de     dados
                                                        limpa_buffer();
                                ;};
              };
}
//---------------------    FIM    ------------------------------
```

O soft acima, apresenta um protocolo onde o pacote de dados tem um formato fixo. Caso fosse para receber dados no buffer até a ocorrência de um determinado caractere, como por exemplo o caractere 0x13, poderia, escrever algo como:

```
int    buffer[30];
int    next   =      0;
do{
              buffer[next   ++]=getc();
}while((      buffer[next] !=     0x13)  |      (next  <      30));
```

Utilizamos também o tipo de dados enumerados definido como “enum”, onde definimos uma variável para o tipo “enum” e atribuímos rótulos seqüênciais, como:

```
enum   COR_LED{RED,  GREEN, AMBER};
```

Assim (COR_LED)1 retorna GREEN, índice 1 no vetor enumerado. Outro tipo muito interessante é o tipo estrutura de dados que pode ser definido com um conjunto de tipos de dados de um ou mais tipos distintos organizados em forma de registro.

```
typedef      struct {
             int    posicao;
             int    buffer[10]
             int16  valor;
}TAmostra
...
TAmostra     exemplo;
...
exemplo.posicao     =     0x01;
exemplo.buffer[3]   =     (exemplo.valor      >>8)&0xFF
```

O comando typedef define a estrutura, seguido dos registros, finalizado pelo nome da estrutura “tamostra”. Quando for necessário, cria-se a variável “registro” do tipo “tamostra” e assim utilize o ponto para separar os campos deste registro.

Outro recurso utilizado é o ponteiros, de grande flexibilidade facilita muito o trabalho, pois tudo que está declarado está armazenado em algum lugar da memória ram. O ponteiro aponta para o endereço de armazenamento, sem que você precise saber o endereço real para manipular dados.

Declaramos um ponteiro utilizando o “*” asterisco antecedendo o de dado. como em:

```
static short  lrc(   int    *buffer,     int    size,  int    modo   ){
```

A função lrc não recebeu o buffer como parâmetro mas somente o endereço do buffer, ou seja apontou para a localização da memória onde está armazenado o dado.

Você já deve ter observado que ao declarar uma variável para sinalização de ocorrências internas e nomeando cada bit desta variável, traz uma grande economia de memória pois um byte pode sinalizar 8 flags diferentes. Neste último exemplo criamos uma flag apenas para demonstrar, já que utilizamos apenas 2 bits desta com intuito demonstrativo, como:

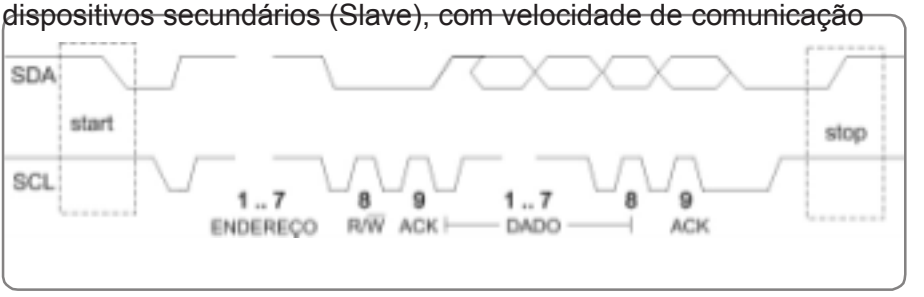
```
int    fag;
                                                        //fag  de
sinalização  interna
#bit   fag_use_led          =      fag.0  //0    não    -      1      sim
```

```
#bit   fag_led_on              =     fag.1 //    bit   1     do
byte   fag
```

Normalmente não tenho problemas com a forma de escrita de funções, mas se você utilizar uma função que faz referência a outra que está abaixo desta no código fonte, então o compilador irá apontar erro, pois não encontrou a referência. Mudando a ordem resolve este problema. No entanto a forma correta é declarar as funções utilizadas no início do código fonte, logo abaixo das declarações de variáveis, se houver, indicando desta forma ao compilador a existência destas funções.

7.2- Funções para comunicação I2C

I2C é um protocolo de comunicação em duas linhas de sinais, uma linha de clock chamada de SCL e uma linha de dados chamada de SDA, bidirecionais com coletor aberto, requerendo resistor de pull-upp com valores entre 4k7 a 10k para VCC. Um sistema I2C utilizado em grande número de dispositivos, é composto por um dispositivo principal (Master) e no máximo 112 dispositivos secundários (Slave), com velocidade de comunicação



padrão de 100kbits/s e 10 kbits/s em modo de baixa velocidade, A comunicação inicia com a condição “start”, seguido dos 7 bits de endereçamento e do 8º bit RW/ (leitura/escrita). O dispositivo slave envia o sinal de ACK (reconhecimento) e o dispositivo master (ou slave) envia pacotes de 8 bits de dados, sempre seguidos de um sinal ACK enviado pelo dispositivo slave (ou master) confirmando a recepção. O dispositivo master termina a comunicação com a condição “stop”.

Byte de seleção para memórias EEPROM série 24Cxx

	Device Code	Chip Enable	Block Select	RW/
bit	b7 b6 b5 b4	b3 b2	b1	b0
Dev. Select	1 0 0 1	E2 E1	A8	RW/

As funções disponíveis para I2C são:

Função	Descrição
I2C_Start()	Estabelece uma condição de início com o dispositivo I2C em modo master. Depois a linha de clock vai para nível baixo até o uso das funções read/write.
I2C_Stop()	Finaliza a condição de inicialização do dispositivo I2C
I2C_Read()	lê um byte no dispositivo I2C.
I2C_Write()	Envia um byte ao dispositivo I2C
I2C_pool()	Retorna TRUE quando o hard receber um byte no buffer para posteriormente chamar I2C_Read()

As funções I2C_read e I2Cwrite, em modo master gera seu próprio sinal de clock e em modo slave, permanece a espera destes. Antes de utilizar estas funções deve-se declarar sua utilização incluindo a diretiva: #use i2c(master,scl=PIN_B3,sda=PIN_B4).

Abaixo exemplo de utilização:

```
#include     <16F628A.h>
#FUSES NOWDT,XT,PUT,NOPROTECT,BROWNOUT,NOMCLR,NOLVP,NOCPD

#use   delay(clock=4000000,RESTART_WDT)
#use   i2c(master,scl=PIN_B3,sda=PIN_B4)
void   main(){
       int    buffer[33],   i=0;
       i2c_start();                        //condição    de
inicialização
       i2c_write(0xc1);//byte     de     seleção
       while( i      <=     32){
                     if(    i2c_poll()    )      buffer[i++]=
i2c_read();
       };
       i2c_stop();
}
//     outros exemplos
i2c_start();                                              //condição    de
inicialização
i2c_write(0xa1);                  //byte de     seleção      (7bits)
dado   =      i2c_read();  //realiza     leitura
i2c_stop();                                               //condição
de     término
//
```

```
i2c_start();                                    //condição de
inicialização
i2c_write(0xa0);                //byte de seleção (7bits)
i2c_write(    c      );                         //grava o byte
```

```
i2c_stop();                                                    //condição
de     término
```

7.3- Comunicação SPI

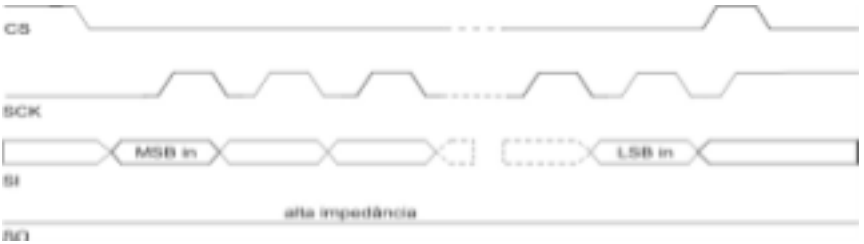
Assim como o I2C, a comunicação SPI (Serial Peripheral Interface) é um dos padrões mais difundido que necessita de apenas 3 linhas de controle para a comunicação entre os dispositivos, um principal que se comunica por uma linha com outro secundário que retorna por outra linha tendo a linha de sincronismo (clock) em comum.

Assim como o I2C, deve ser inicializada com a diretivas setup_spi(modo), que pode aceitar as seguintes constantes:

SPI_MASTER, SPI_SLAVE, SPI_L_TO_H, SPI_H_TO_L, SPI_CLK_DIV_4, SPI_CLK_DIV_16, SPI_CLK_DIV_64, SPI_CLK_T2, SPI_SS_DISABLED

Ex. setup_spi(spi_master | spi_l_to_h | spi_clk_div_16);

As funções disponíveis para comunicação SPI são:



função	descrição
SPI_DATA_IS_IN()	devolve TRUE se recebeu dados na SPI
SPI_READ()	aguarda até a leitura de um byte na SPI
SPI_WRITE(valor)	Escreve um valor na SPI

Exemplo de utilização:

```
if(    spi_data_is_in()    )      dado   =      spi_read();          //lê
o      dado
spi_write(    0x00   );
```

Capítulo

8

Captura, Comparação e PWM

Captura, comparação e PWM, também conhecido como CCP, está presente no PIC16F62x no módulo CCP1, pino Rb3/CCP1, realiza operações de captura de informações de 16 bits procedentes de TMR1, Comparação de um valor de registro com o TMR1 e modulação por largura de pulso com o TMR2.

O CCP1 é selecionado e configurado pelo registro CCP1CON, endereço 17h e têm os valores armazenados em dois outros registros de 8 bits denominados CCPR1L e CCPR1H.

endereço	nome	descrição
15h	CCPR1L	byte LSB de captura, comparação e pwm
16h	CCPR1H	byte MSB de captura, comparação e pwm

registro de configuração CCP1CON endereço 17h

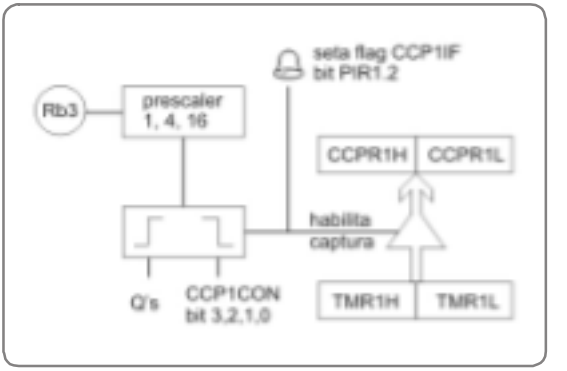
bit 7	bit 6	bit 5	bit 4	bit3	bit 2	bit 1	bit 0
-	-	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0

bit	descrição	
CCP1M3, CCP1M2 CCP1M1, CCP1M0	0000	captura, comparação e pwm desligados
	0100	modo captura pulso descendente
	0101	modo captura pulso ascendente
	0110	modo captura a cada 4 pulsos ascendente
	0111	modo captura a cada 16 pulsos ascend.
	1000	modo comparação seta saída p/ CCP1IF
	1001	modo comparação limpa saída
	1010	modo comparação gera interrupção em CCP1IF, CCP1 não é afetado
	1011	modo comparação, gatilho especial seta CCP1IF e CCP1 reseta timer1 (TMR1)
	11xx	modo PWM

8.1- Modo captura

Neste modo, quando um evento ocorre no pino Rb3, o valor do timer1 é transferido para os dois registros de 8 bits CCPR1L e CCPR1H. Quando se realiza a captura é ativada a interrupção CCP1IF bit 2 do registro PIR, e se após a interrupção houver outra captura e se ainda não se fez leitura do conteúdo de CCPR1, este se perderá e passará a ter o conteúdo da nova captura.

Uma possível utilização para este módulo, pode ser a medição dos intervalos de tempo que existem entre os pulsos presentes no pino RB3. Vemos assim que o pino RB3 deve ser configurado como entrada (TRISB<3>) se acaso RB3 estiver configurado como saída uma escrita na porta pode causar a condição de captura.



A captura depende de TMR1, este deve estar executando como tempo-

rizador ou como contador em modo síncrono. Se o TMR1 estiver como contador asíncromo o modo captura não irá funcionar.

A função que nos permite trabalhar com CCP1 é setup_ccp1 (modo), onde modo pode ser uma das seguintes constantes internas do compilador.

constantes	descrição
CCP_CAPTURE_FE	captura pulso descendente
CCP_CAPTURE_RE	captura pulso ascendente
CCP_CAPTURE_DIV_4	captura a cada 4 pulsos ascendente
CCP_CAPTURE_DIV_16	captura a cada 16 pulsos ascendente

```
/*
-----------------------------------------------------
             Projeto......:      programando  microcontrolador    pic
             Descrição....:      captura      a      largura      de
um     pulso  no     pino   RB3
-----------------------------------------------------
*/
#include     <16F628A.h>
#FUSES NOWDT,XT,PUT,NOPROTECT,BROWNOUT,NOMCLR,NOLVP,NOCPD
#use   fast_io(a)
#use   fast_io(b)
#use   delay(clock=4000000)
#use   rs232(baud=9600,parity=N,xmit=PIN_B2,rcv=PIN_B1,bits=8)
//
#byte  Rb     =      0x06
#defne trisb         0b00001010          //     1      input
//
#bit   capturou                          =      0x0C.2 //CCP1IF      -
ver    capítulo      3.12
#bit   interrupcao   =      0x8C.2 //CCP1IE
//
void   main(){
             int16  tempo;
             setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
             setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
             setup_timer_2(T2_DISABLED,0,1);
             setup_comparator(NC_NC_NC_NC);
             setup_vref(FALSE);
             //captura    pulso  ascendente
             setup_ccp1(CCP_CAPTURE_RE);
             //
             Set_tris_b(   trisb  );     //direção     dos    bits   do
portB
             Rb     =      0;
                                                  //zera todos  os     bits
```

```
                While( true   ){
                              capturou                    =      0;
//zera fag    de     captura
```

```
                                interrupcao= 0;            //zera fag    de
interrupção
                                set_timer1(0);      //zera o      timer1 ,
incremento    a      cada       1us.
                                while( !      capturou      );
//executa     enquanto          não    capturou
                                tempo  =      ccp_1;//houve um     evento em
RB3    capturou      o          tempo
                                printf("tempo(us)   =      %2X\n\r",
tempo);       //envia           ao     PC
                                delay_ms(200);
              };
}
```

8.2- Modo comparação

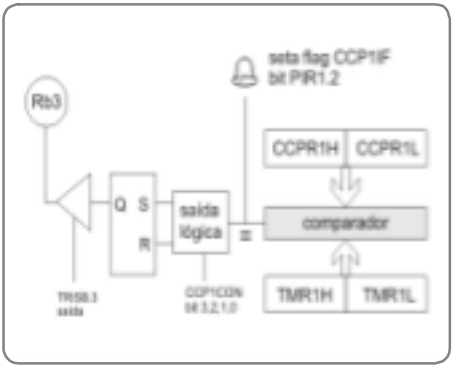
o conteúdo do registro CCPR1H e CCPR1L são é constantemente comparado com o valor do TMR1 que deve estar em modo temporizador ou em modo síncrono, até haver uma ocorrência no pino Rb3/CCP1. Quando o valor coincide a porta RB3 que previamente deve ser configurada como saída (TRISB<3>) pode apresentar 1, 0 ou não variar, sendo sinalizada pela interrupção CCP1IF.

A ação na porta é configurada pelos bits CCP1M3: CCP1M0 do registro CCP1COM endereço 17h e também aqui a função que nos permite trabalhar com CCP1 é setup_ccp1 (modo), onde modo pode ser uma das seguintes constantes internas do compilador.

constantes	descrição
CCP_COMPARE_SET_ON_MATCH	quando TMR1=CCP1, seta saída
CCP_COMPARE_CLR_ON_MATCH	quando TMR1=CCP1, zera saída
CCP_COMPARE_INT	sinaliza interrupção e não muda
CCP_COMPARE_RESET_TIMER	quando TMR0=CCP1, zera o timer

```
#include      <16F628A.h>
```

```
#FUSES NOWDT, XT,    PUT,   NOPROTECT,    BROWNOUT,     MCLR,  NOLVP,
NOCPD
#use   delay(clock=4000000)
```



```
#use   fast_io(b)

void   main(){
              setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
              setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
              setup_timer_2(T2_DISABLED,0,1);

              setup_ccp1(CCP_COMPARE_SET_ON_MATCH);

              setup_comparator(NC_NC_NC_NC);
              setup_vref(FALSE);

              set_tris_b(   0b00000001    );     //Rb.0 =      entrada
              while(TRUE){

                                                 while( input(PIN_B0) );
//aguarda     um     pulso  em     Rb0
                                                 //TMR0=CCP1   e      Rb3=1

setup_ccp1(CCP_COMPARE_SET_ON_MATCH);
                                                 CCP_1=0;
                                                 //agora      Rb3=0
                                                 set_timer1(0);
       //zera o      timer1
                                                 CCP_1  =      10000;
       //ajusta      o      tempo  limite em     10ms


              //tempo       /      (clock/4)


              //10.000      =      0.01*(4.000.000/4)
                                                 //TMR0=CCP1   e      Rb3=0

setup_ccp1(CCP_COMPARE_CLR_ON_MATCH);
                                                 //Debounce    -      permitindo
apenas 1      pulso  por     segundo
                                                 delay_ms(1000);


              };
}
```

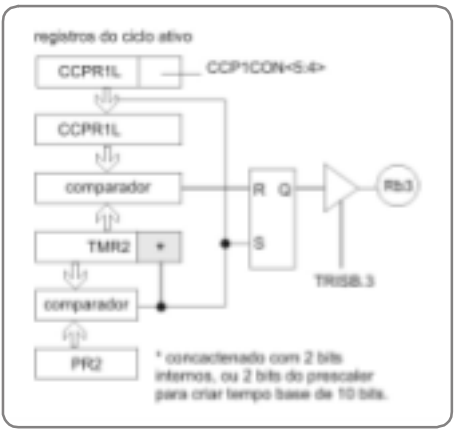
8.3- Modo PWM Modulação por Largura de Pulso

O PWM é uma técnica que permite o controle da potência aplicada a uma carga através da largura dos pulsos aplicados. Neste modo a porta RB3 (TRISB<3>), previamente configurada como saída, oscila entre 0 e 1 em períodos de tempo variável com

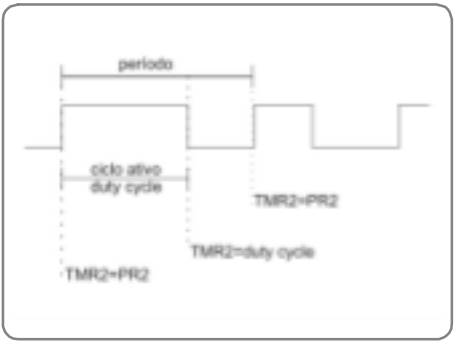
uma resolução superior a 10 bits. Este período ativo ou pulso de excitação chama-se “duty cycle” e quanto mais largo for, maior a potência aplicada na carga.

Quando o valor do registro PR2 coincide com o de TMR2 a porta passa para 1, o valor de TMR2 é zerado, o registro CCP1 é setado e o TMR2 passa a ser comparado com o valor de CCPR1L e quando coincide o latch da porta é resetado gerando o “duty

cycle". O Registro CCPR1H é usado para buffer interno apenas como leitura.



O timer2 deve ser configurado com a função setup_timer_2(modo, período, postscale). As constantes para o modo de funcionamento são: T2_DISABLED, T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_16, para o período um valor de 0 à 255 e para o postscale um valor de 0 à 16 que determina o tempo antes que aconteça interrupção do timer.



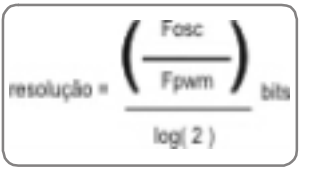
A função que determina o ciclo ativo é set_pwm1_duty(valor), onde valor é um inteiro que determina o tempo do ciclo ativo e a função set_ccp1(CCP_PWM), inicializa o módulo CCP no modo PWM.

Exemplificando a configuração do PWM para gerar um forma de onda de 38khz para um diodo de infravermelho para posterior utilização com um TSOP1738, primeiramente carregaríamos o valor do período da freqüência escolhida no timer2, registro PR2. A saída PWM tem um tempo base que é o período e o tempo de saída (duty cycle). Freqüência(f) e período(p) são inversamente proporcional então f=1/p e p= 1/f , assim, como TMR2 = PR2 que representa um ciclo completo, podemos determinar o período requerido para gerar a freqüência de 38khz, fazendo 1/38.000 resultando em 0,000.026316 ou 26.3 uS. Conhecendo o período determinamos o duty cycle, como por exemplo 50%, calculando a razão do período 50% de 26 = 13uS ativo e 13uS inativo.

Calculamos então o valor a ser carregado no registro PR2 com a equação

PR2 = (4MHz / (4 * TMR2 prescaler * f)) - 1
PR2 = (4.000.000 / 4 * 1 * 38.000) - 1
PR2 = 25.315

então:
setup_timer_2(T2_DIV_BY_1, 25, 1);
set_pwm1_duty(50);
set_ccp1(CCP_PWM);



A resolução desta configuração pode ser calculada pela equação ao lado, aplicando-a temos:

resolução = log(4.000.000/38.000)=2.022
resolução = 2.022/0,301 = 6,7 bits

Tanto a configuração do timer2 ou quanto ao ciclo ativo, podem ser alterados a qualquer momento “on the fly” para adequar as suas necessidades e lógicamente quanto maior for a resolução, quanto maior será a precisão do sinal gerado.

```
void   buzina(      void   ){
              setup_ccp1(CCP_PWM);
              setup_timer_2(T2_DIV_BY_4, 250,   0);
              for(x  =     0;     x      <=     9;     x++    ){
                                  set_pwm1_duty(      x      *      10);
                                  delay_ms(200);
              };
              set_pwm1_duty(0);
 }
```

Capítulo

9

Comparadores e Tensão de Referência

9.1- Modulo Comparador

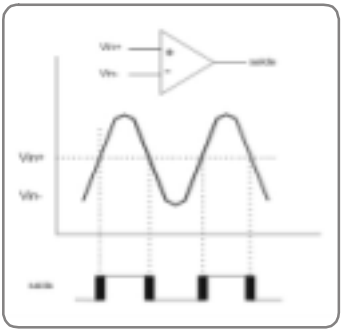
O módulo comparador contém dois comparadores analógicos com entradas multiplexadas em RA0 à RA3, podendo utilizar fonte de referência externa ou interna no chip com resolução de 4 bits permite 16 valores de referência. Por sua vez o comparador pode disparar no flanco positivo o negativo, tendo suas configurações controladas pelo registro CMCON no endereço 01Fh.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0

<table>
<tr><td>bit</td><td>descrição</td></tr>
<tr><td>C2OUT</td><td>Saída do Comparador 2
<table>
<tr><td>quando C2INV = 0</td><td>quando C2INV = 1</td></tr>
<tr><td>1 = C2 Vin+ > C2 Vin-</td><td>1 = C2 Vin+ < C2 Vin-</td></tr>
<tr><td>0 = C2 Vin+ < C2 Vin-</td><td>0 = C2 Vin+ > C2 Vin-</td></tr>
</table></td></tr>
<tr><td>C1OUT</td><td>Saída do Comparador 1
<table>
<tr><td>quando C1INV = 0</td><td>quando C1INV = 1</td></tr>
<tr><td>1 = C1 Vin+ > C1 Vin-</td><td>1 = C1 Vin+ < C1 Vin-</td></tr>
<tr><td>0 = C1 Vin+ < C1 Vin-</td><td>0 = C1 Vin+ > C1 Vin-</td></tr>
</table></td></tr>
<tr><td>C2INV</td><td>inversão da saída do comparador 2
1 = saída de C2 invertida
0 = saída de C2 normal</td></tr>
</table>

<table>
<tr><td>bit</td><td>descrição</td></tr>
<tr><td>C1INV</td><td>inversão da saída do comparador 1
1 = saída de C1 invertida
0 = saída de C1 normal</td></tr>
<tr><td>CIS</td><td>chaveamento da entrada do comparador
<table>
<tr><td>quando CM2:CM0 = 001</td><td>quando CM2:CM0 = 010</td></tr>
<tr><td>1 = C1 Vin- em Ra.3</td><td>1 = C1 Vin- em Ra.3 e
C2 Vin- em Ra.2</td></tr>
<tr><td>0 = C1 Vin- em Ra.0</td><td>0 = C1 Vin- em Ra.0 e
C2 Vin- em Ra.1</td></tr>
</table></td></tr>
<tr><td>CM2:CM0</td><td>como mostrado no grá_co abaixo:</td></tr>
</table>

A Operação de comparação (não confundir com comparador CCP) ocorre com dois sinais analógicos em sua(s) porta(s), quando a tensão analógica em Vin+ for menor que em Vin-, a saída do comparador será 0 (zero) e quando a tensão analógica em Vin+ for maior que em Vin-, a saída do comparador será 1 (um), saída esta digital que apresenta uma pequena área incerta (conforme vemos na figura ao lado) em resposta do tempo de aquisição que é 150 ns(típico e 400ns máximo).



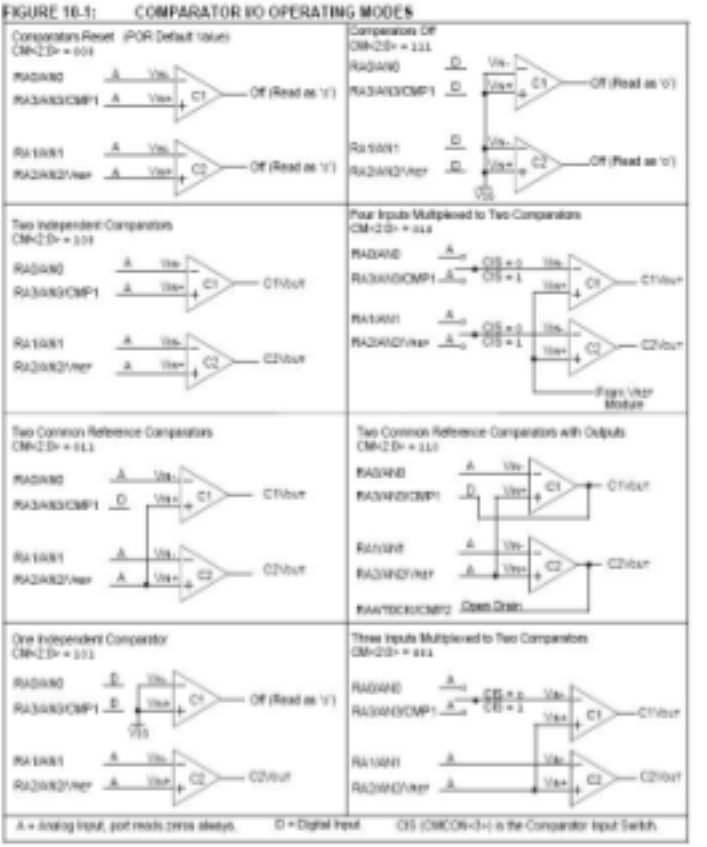
O Reset coloca os comparadores em default (CM2:CM0 = 000) e desconecta suas saídas. Quando em repouso ou SLEEP o comparador continua funcionando e tendo habilitado a interrupção, uma variação em qualquer de suas saídas irá gerar um pedido de interrupção, tirando o PIC do repouso.

Características do módulo comparador
Voltagem de referência interna selecionável em 16 níveis.
Tempo de ajuste da voltagem de referência é de 10 us(máx)
Tempo de resposta do comparador é de 400ns (máx)
Entradas analógicas (devem estar entre Vdd e Vss)
Possíveis valores para seleção de voltagem

V = Vdd - Vss
V = Vref(+) - Vref(-)
Saídas digitais

Como podemos variar, a qualquer momento, a tensão de referência temos uma possibilidade interessante que é a construção de gatilho via hardware, onde a tensão ao disparar o comparador aciona flag de interrupção CMIF bit 6 do registro PIR1, procedendo a captura de um sinal inclusive com ajuste de disparo na subida ou descida do pulso.

A função que nos permite ajustar as opções dos comparadores é Setup_comparator(modo), onde modo pode ser uma das seg-



uintes constantes internas.

constante	referência
A0_A3_A1_A2	C1-, C1+, C2 -, C2+
A0_A2_A1_A2	C1-, C1+, C2 -, C2+
NC_NC_A1_A2	C1-, C1+, C2 -, C2+
NC_NC_NC_NC	C1-, C1+, C2 -, C2+
A0_VR_A2_VR	C1-, C1+, C2 -, C2+
A3_VR_A2_VR	C1-, C1+, C2 -, C2+
A0_A2_A1_A2_OUT_ON_A3_A4	C1-, C1+, C2 -, C2+
A3_A2_A1_A2	C1-, C1+, C2 -, C2+

Exemplo:

setup_comparator (A0_A3_A1_A2);

setup_comparator(A0_VR_A1_VR); //A0 e A1 com voltagem de referência interna.

9.2- Tensão de Referência

Este módulo consiste em uma rede de 16 resistores de precisão em série que provém 16 valores de tensão com saída no pino RA2, controlados pelo registro VRCON, endereço 9Fh.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
VREN	VROE	VRR	-	VR3	VR2	VR1	VR0

bit	descrição
VREN	habilitação da voltagem de referência Vref 1 = habilitado 0 = desabilitado, sem consumo de corrente Idd
VROE	saída da voltagem de referência no pino Ra.2 1 = Vref. conectado 0 = Vref. desconectado
VRR	Seleção da faixa de referência 1= faixa baixa 0 = faixa alta

bit	descrição
VR3: VR0	seleção do valor 0<= VR[3:0] <= 15 quando VRR=1: VREF = (VR<3:0>/24) * VDD quando VRR=0: VREF = 1/4 * VDD + (VR<3:0>/32) * Vdd

Exemplo de utilização da tensão de referência para uma saída de 1,25 V.

Alimentação VDD = 5volts

VREn = 1 tensão de referencia habilitado

VRoe = 0 desconectado de RA2

VRr = 1 faixa baixa voltagem

VR3,2,1,0 = 0110 = 6 decimal

Como VRR está em 1, aplicamos a fórmula de baixa tensão:

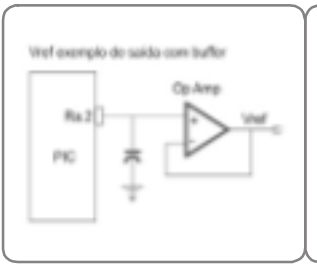
Vref = (Vr <3:0> / 24) x Vdd

Vref = (6 / 24) x 5

Vref = 1,25 Volts.

Os valores de tensão obtidos neste módulo além de utilizados como fonte de referência para os comparadores analógicos, estão também disponíveis para uso externo se o bit VROE estiver em 1. O reset desabilita a tensão de referência, limpando o bit VREN (VRCON <7>), desconecta a Vref do pino RA2 limpando o bit VROE (VRCON <6>),seleciona a faixa de alta voltagem limpando o bit VRR (VRCON <5>) e limpa os bit VR3,VR2,VR1, VR0. já o repouso ou o estouro de WDT não afeta o registro.

A função para que nos permite ajustar a tensão de referência é Setup_Vref(modo | valor), onde modo é uma das seguintes constantes internas do compilador.



constante	de_nição
False	desativa voltagem de referência
VREF_LOW	valores de 1,25 à 3,59
VREF_HIGH	valores de 0 à 3,13

Em combinação com a constante podemos opcionalmente utilizar uma constante para valor podendo ser um inteiro de 0 à 15, que em conjunto com a retorna o valor setado para referência.

Para VREF_LOW, temos Vref = (VDD * valor / 24)

Para VREF_HIGH, temos Vref = (VDD * valor/32)+VDD /4

Exemplo:

```
setup_vref (VREF_HIGH | 8); //saída em Ra.2
vdd = 5v
vref = ( 5 * 8 / 32 ) + 5/4
vref = 2,5 volts.
```

setup_vref(VREF_LOW | 15 | VREF_A2); //voltagem de referência para comparador, bits CM2:CM0 do registro CMCON endereço 01Fh.

Capítulo

10

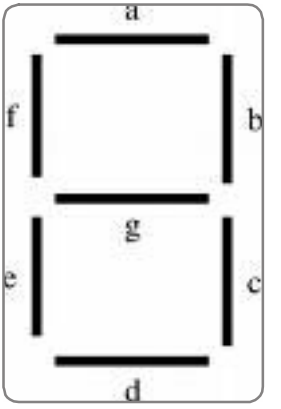
Displays

10.1- Display LED de sete segmentos:

Constituídos de LEDs estes displays apresentam elevado consumo de corrente do sistema e podem operar em tempos de centenas de nanosegundos. Isto significa que podemos opera-los em forma pulsada, com o tempo ativo bastante pequeno, reduzindo desta forma o consumo de corrente.

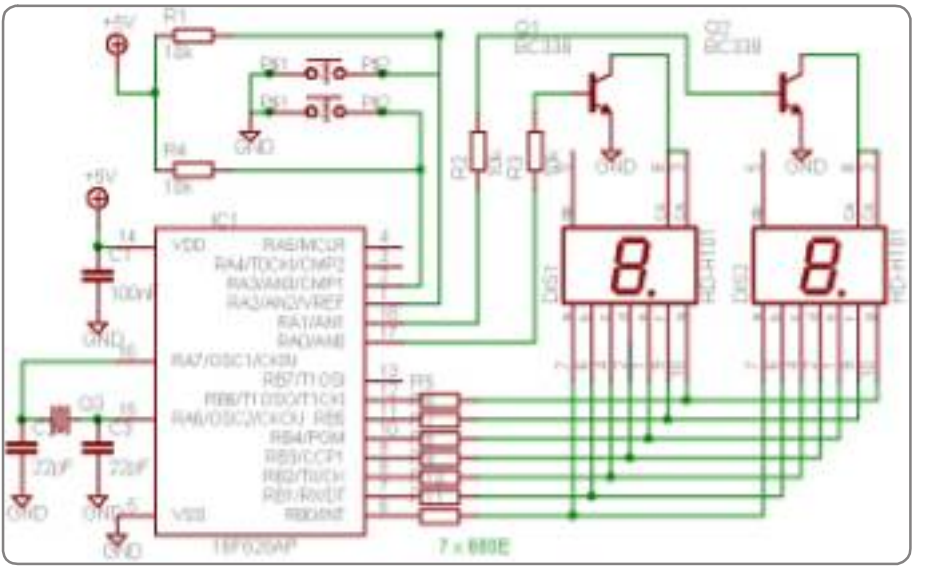
Estes displays apresentam internamente leds ordenados de forma a formar um caracter. Estes leds são chamados de segmentos, sendo 7 o número padrão de segmentos (A..G).

A	B	C	D	E	F	G	caracter
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	2
1	1	1	1	0	0	1	3
0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	5
1	0	1	1	1	1	1	6
1	1	0	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	1	0	1	1	9



Utilizando e esquema abaixo, faremos ua contagem e uma

animação, exemplificando a forma de utilização destes displays.



```
#include      <16F628A.h>

#FUSES NOWDT,XT,PUT,NOPROTECT,BROWNOUT,NOMCLR,NOLVP,NOCPD

#use   delay(clock=4000000)
              //-------------------------------------------------
              //     defnições     de     entrada/saída I/O
              //-------------------------------------------------
#byte  Ra     =      0x05
#byte  Rb     =      0x06
              //
#bit   catodo1                    =      Rb.7          //0    saída
#bit   catodo2                    =      Ra.1          //0    saída
#bit   botao_sim     =      Ra.2          //1    entrada
#bit   botao_nao     =      Ra.3          //1    entrada
              //
#use   fast_io(a)
#use   fast_io(b)
              //
#defne trisa         0b00001100          //     0      saída, 1
entrada
#defne trisb         0b00000000          //
              //valores     válidos       true   ou     false;
#defne catodo_comum  true
              //
              //tabela      para   displays      de     catodo comum
int    caracter[16]={0x3F,0x06,0x5B,0x4F,0x66,
```

```
                                        0x6D,0x7C,0x07,0x7F,0x67,

0x77,0x7C,0x39,0x5E,0x79,0x71};
            //tabela     animação
byte   animacao[6]={0x01,0x02,0x04,0x08,0x10,0x20};
            //
short  usar_animacao;
int    indice_animacao;
            //
struct Estrutura{
            int    dado;        //defne      o     contador
            int    ativo; //defne      o     display      ativo  no
momento
            int    temp;        //variável   temporária
}display;
            //-------------------------------------------------
            //interrupção do     timer0 a     cada   8     ms
            //-------------------------------------------------
#int_TIMER0
TIMER0_isr()  {
            if(    !     usar_animacao ){
                        display.temp  =     0;

                        //chaveia    o     display
ativo
                        switch(      display.ativo ){
                                    case   1     :
{

catodo2      =     0;                  //apaga      o     display
nr     2

catodo1      =     1;                  //ascende    o     display
nr     1

display.temp  =     (     display.dado  &     0x0F   );     //lsb

//passa      o     controle     para   o     próximo      display

display.ativo++;

};
                                    break;
                                    case   2     :
{

catodo1      =     0;           //apaga      o     display      nr
1

catodo2      =     1;           //ascende    o     display      nr
2
```

```
display.temp = ( display.dado >> 4 ) & 0x0F;

//passa o controle para o primeiro display

display.ativo = 1;

};
                                        break;
                                    };
                                    //se usar catodo comum inverte os bytes 1=>0 e 0=>1
                                    //caso contrário coloca o byte no portB
                                    #if catodo_comum
                                        Rb = ~caracter[ display.temp ];
                                    #else
                                        Rb = caracter[ display.temp ];
                                    #endif
                                    //
            }else{
                                    //chaveia o display ativo
                                    switch( display.ativo ){
```

```
                                                    case   1      :
{
catodo2      =      0;            //apaga      display      nr     2
catodo1      =      1;            //ascende    o      display      nr
1
//passa      o      controle      para   o      próximo      display
display.ativo++;
};
                                                    break;
                                                    case   2      :
{
catodo1      =      0;            //apaga      o      display      nr
1
catodo2      =      1;            //ascende    o      display      nr
2
//passa      o      controle      para   o      primeiro     display
display.ativo=      1;
};
                                                    break;
                                    };
                                    //se   usar   catodo comum  inverte
os     bytes  1=>0   e      0=>1
                                    //caso contrário     coloca o      byte
no     portB
                                    #if    catodo_comum
                                                    Rb     =
~animacao[indice_animacao];
                                    #else
                                                    Rb     =
animacao[indice_animacao];
                                    #endif
              };
}
              //-------------------------------------------------
              //função     principal
              //-------------------------------------------------
void   main() {
              //resolução   =      1/clock       *      (256   *
RTCC_DIV)
              //0,000.001*(256*32)       =      8,2ms.
              setup_timer_0(RTCC_INTERNAL|RTCC_DIV_32);
              //resolução   =      1/clock       *      (65536 *
T1_DIV)
              //0,000.001*(65536*1)      =      65,5ms.
```

```
            setup_timer_1(      T1_DISABLED  );
            setup_timer_2(      T2_DISABLED, 0,     1      );
            setup_comparator(   NC_NC_NC_NC  );
            setup_vref(   FALSE  );
            enable_interrupts(  GLOBAL );
            enable_interrupts(  INT_TIMER0   );
            //
            set_tris_a(   trisa  );     //defne      a      direção
do     portA
            set_tris_b(   trisb  );     //defne      a      direção
do     portB
            //
            Ra     =      0;           //limpa      todos  os     bits
do     portA
            Rb     =      0;           //limpa      todos  os     bits
do     portB
            //zera as     variáveis
            display.dado               =      0x00;
            display.ativo       =      0x01;
            indice_animacao=    0x00;
            usar_animacao       =      false;
```

```
              while( true   ){
                                    //
                                    delay_ms(     50     );
                                    if(    !      botao_sim     )
   usar_animacao=      true;
                                    delay_ms(     50     );
                                    if(    !      botao_nao)
   usar_animacao=      false;
                                    //
                                    if(    !      usar_animacao){
                                                  display.dado ++;
                                                  delay_ms(     300
   );
                                    }else{
                                                  indice_animacao
   ++;
                                                  if(
   indice_animacao     >      5      )      indice_animacao     =      0;
                                                  delay_ms(     200
   );
                                    };
              };
   }
```

Este software exibe a contagem da variável dado em ordem crescente e em hexadecimal e também uma animação no display. Na verdade este soft poderia ter sido escrito de forma mais compacta, porém para fins educativo apresenta algumas características. Encontramos display LED na configuração anodo comum e catodo comum.

Na configuração ânodo comum onde o ânodo dos 8 leds do display são ligados juntos, devendo aplicar nível lógico zero (0) para que ele acenda já que o catodo deste display é ligado no VCC. Já na configuração catodo comum, este deve estar ligado a massa e aplicar nível lógico um (1) no ânodo do segmento desejado.

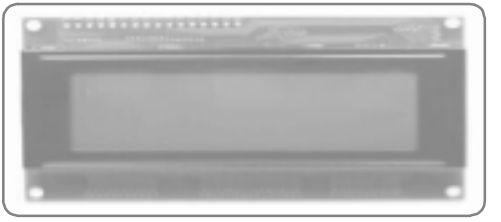
Por isso utilizamos a diretiva de pré compilação #if para que o compilador compile somente a parte que se refere, ignorando o contrário. Utilizamos também uma tabela para catodo comum, com 16 elementos com índice inicial em zero (0), onde a todo momento o PIC procura o lsb e msb do byte correspondente. Aqui também se aplica a diretiva de pré-compilação, pois se o display for de ânodo comum o PIC faz um complemento (~) do byte que é a mesma coisa que fazer XOR do byte com 0xFF. Esta operação inverte os bytes levanto todos os uns (1) para zero (0) e vice-

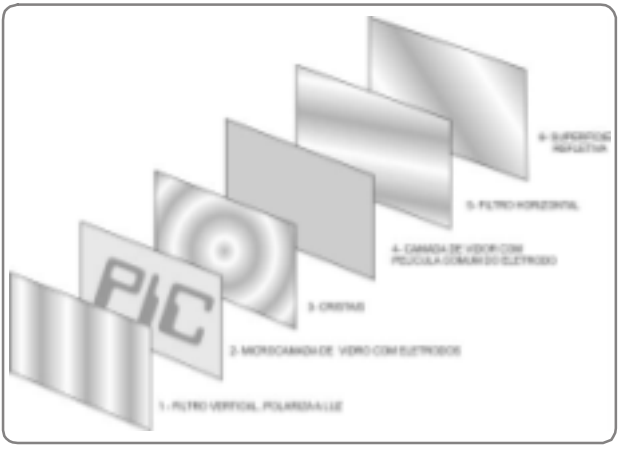
versa. Supondo que o byte seje por exemplo 0x3D (0b0011.1101), o primeiro digito fica com 0011.1101 AND 0000.1111 = 0000.1101

(13), procurando na tabela o décimo quarto elemento (lembre-se do zero) temos, 0x5E que representa os seguimentos do display referente a letra D. Com outro dígito fizemos um deslocamento de 4 casas para direita (display.dado >> 4) 0011.1101 >>4 = 0000.0011 e posteriormente um AND com 0x0F gerando 0011 que é o número 3 decimal. Procurando na tabela o terceiro elemento encontramos 0x4F referente aos seguimentos do número 3. Ao trabalhar com este tipo de display, envie os dados e depois acenda somente o display que receberá os dados. Fazendo isso de forma sucessiva a todos os display, fará a falsa ilusão que todos estão aceso ao mesmo tempo devido a persistência da imagem nos nossos olhos.

10.2- Display LCD

Conhecidos como LCD (liquid crystal display), tem como principal vantagem o baixo consumo e a exibição de caracteres alfanuméricos em várias linhas e até mesmo de forma gráfica. O display de LCD é basicamente constituído de duas placas de cristal muito finas, entre as quais há uma camada de cristal líquido. Esta camada apresenta uma





estrutura molecular cristalina que é capaz de mudar sua orientação sob a influência de um campo elétrico. Conforme a direção em que se organizem as moléculas, a camada de cristal líquido torna-se

transparente ou refletiva conforme a orientação das moléculas em função de uma tensão aplicada. Filtros polarizadores vertical e horizontal são colocados no conjunto, fazendo com que a luz polarizada verticalmente que atravesse os segmentos excitados aparecendo como imagens escuras sobre um fundo claro.

Todo display de cristal líquido utiliza um controlador embutido no display para facilitar o interfaceamento do display LCD com outros dispositivos, sendo o Hitachi 44780 o controlador mais comum do mercado. Com 16 pinos o LCD, apresenta os seguintes pinos:

pino	descrição
1	GND, ligado a massa
2	VCC, alimentação
3	contraste
4	RS - seleção de registro 0 = registro de controle, comando 1 = registro de dados
5	R/W leitura ou escrita 0 = operação de escrita no LCD 1 = operação de leitura no LCD
6	E - Enable 0 = módulo de LCD inibido 1 = módulo de LCD habilitado
7 à 14	via de dados
15	ãnodo do led para backlight
16	catodo do led de backlight

A via de dados do LCD é paralela com 8 bits de dados de escrita/leitura. Sua utilização pode ser em 8 bits ou 4 bits.

No modo 8 bits, utilizamos todo barramento do LCD, onde enviamos um byte de 8 bits ao LCD seguido. de um bit no pino “E”, para aceitação do byte.

No modo 4 bits, utilizamos as vias D7,D6,D5,D4 e enviamos o MSB do byte em 4 bits, seguido pelo bit “E” e posteriormente o byte LSB com 4 bits, seguido pelo bit “E”.

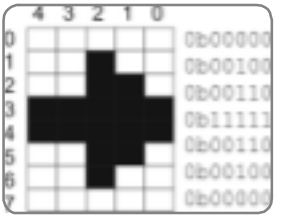
Seqüência para escrita no LCD

1	colocar o bit R/W em nível baixo (0)
2	colocar o bit RS de acordo com a operação (comando/dados)
3	colocar os dados na linha (4 ou 8 bits)
4	colocar o bit E em nível alto (1)
5	dar um delay 2 uS
6	Colocar o bit E em nível baixo (0)
7	para o caso de 4 bits repetir os passos de 3 a 6

tabela de comandos para o controlador Hitach 44780.

byte	descrição
0x01	limpa o lcd o posiciona o cursor em home 0,0
0x02	posiciona o cursor na linha 0 coluna 0
0x04	ao entrar caraceter, desloca cursor p/ esquerda
0x05	ao entrar caracter, desloca mensagem p/ direita
0x06	ao entrar caraceter, desloca cursor p/ direita
0x07	ao entrar caracter, desloca mensagem p/ esquerda
0x08	desliga o lcd sem perder os caracteres
0x0C	desliga o cursor
0x0D	liga o cursor piscando, como um sublinhado
0x0F	liga o cursor alternante
0x10	desloca o cursor para a esquerda
0x14	desloca o cursor para a direita
0x18	desloca a mensagem para a esquerda
0x1C	desloca a mensagem para a esquerda
0x80	endereço da primeira linha do lcd
0xC0	endereço da segunda linha do lcd

O módulo LCD utiliza um conjunto de caracteres ASCII, definido internamente e uma área geradora de caractere, chamada de CGRAM, onde é possível definir até 8 caracteres de 8 bytes que ficam armazenados na RAM de display de dados DDRAM, Para esta definição, inicialmente enviamos o endereço inicial do caractere, com o comando adequado, seguido dos 8 bytes que compõe



o caractere e apontando para a DDRAM do caractere, com tempo de execução de 40 microsegundos.

comando	R/S	R/W	d7	d6	d5,d4,d3,d2,d1,d0
apontar para DDRAM	0	0	1	x	dados
apontar para CGRAM	0	0	0	1	
escrever dado	1	0	x	x	dados
ler dado	1	1	x	x	dados

O compilador mantém as definições do LCD em um arquivo na pasta drives, que deve ser incluído no programa com a diretiva #include <lcd.c>, como no exemplo:

```
#include      <16F628A.h>
#fuses HS,NOWDT,NOPROTECT
#use   delay(clock=4000000)
#defne use_portb_lcd//defne       o      portB  para   o      LCD
#include      <lcd.c>                                  //inclui
rotinas       de     funic. do     LCD
void   main() {
              int16  i                                        //variável
temporária
              lcd_init();          //inicializa o      LCD
              delay_ms(6); //tempo       necessário    para
inicialização
              while(TRUE){  //loop
                                   i++;
//incrementa  a      variável
                                   printf(lcd_putc,"\fcontador:      %U
x);    //escreve     no     LCD
                                   delay_ms(500);             //tempo
para   repetição
              };
}
```

Este drive cria uma estrutura e direciona-a inteiramente para o portB. O portB no entanto pode estar ocupado, então alguns bits do lcd como R/W, R/S e E podem ser direcionados para o portA minimizando o uso do portB ou em alguns casos os bits para o lcd não são seqüênciais. Para tal resolvi escrever este drive que pode utilizar os 4 bits de dados e os 2 bits de controle em qualquer porta ou seqüência, dando mais versatilidade à aplicação. No entanto a critério do leitor pode ser adaptado

para outras situações.

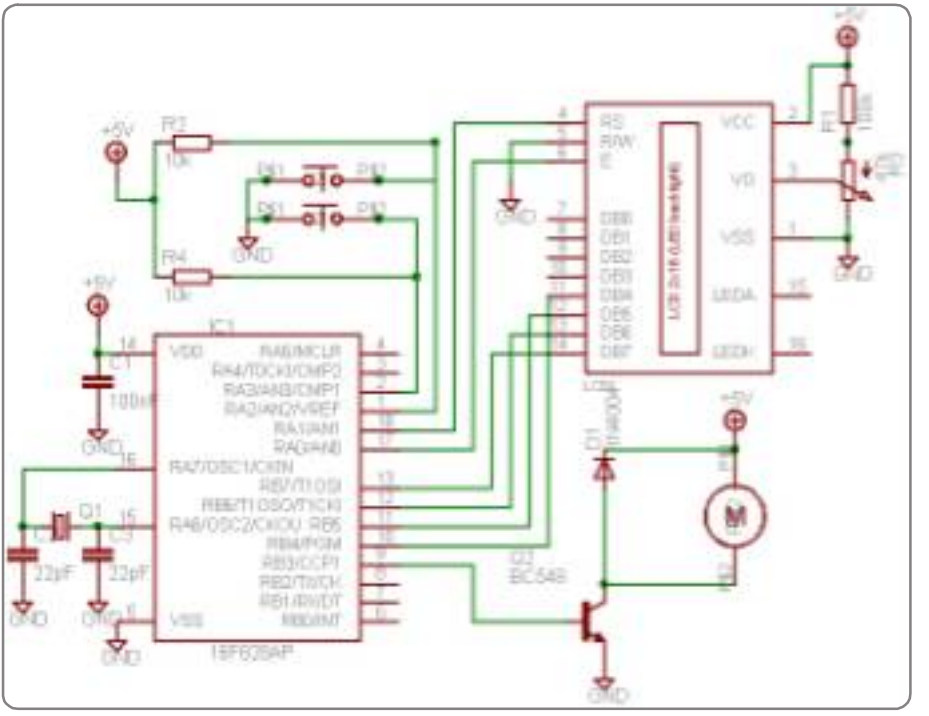
```
//*****************************************************
//     Drive  LCD    -      versão 0.01
//*****************************************************
void   lcd_inicializa(void);
byte   lcd_leitura(  void   );
void   lcd_escreve(  int    c,     int    rs     );
void   lcd_putc(     int    c);
void   lcd_pos(      int    l,     int    c);
             //-------------------------------------------------
             //defnições  de     interfaceamento
             //os   bits   do     lcd    podem  ser    defnidos      aqui
ou     no
             //prog principal.    portA  =      0x05   e      portB  0x06
             //-------------------------------------------------
//#bit lcd_EN =      0x06.0              //lcd  pino   6
//#bit lcd_RS =      0x06.2              //lcd  pino   4
//#bit lcd_d4 =      0x06.7              //lcd  pino   11
//#bit lcd_d5 =      0x06.6              //lcd  pino   12
//#bit lcd_d6 =      0x06.5              //lcd  pino   13
//#bit lcd_d7 =      0x06.4              //lcd  pino   14
             //-------------------------------------------------
             //pulsa      o      bit    enable
             //-------------------------------------------------
#defne strobe_lcd    (LCD_EN      =      1,LCD_EN     =      0)
       //-------------------------------------------------
       //defnições  de     ajustes
       //-------------------------------------------------
#defne lcd_dl 0      //     0=     4      bits,         1=     8      bits
#defne lcd_n         1      //     0=     1      linha, 1=     2
linhas
#defne lcd_f         0      //     0=     5x7,   1=     5x10
#defne lcd_sc 0      //     0=     move   cursor,       1=     shift
#defne lcd_rl 1      //     0=     desloc esquerda,     1      desloc
direita
#defne lcd_d         1      //     0=     display      off,   1=
display       ligado
#defne lcd_c         1      //     0=     cursor desligado,    1=cursor
ligado
#defne lcd_b         1      //     0=     cursor normal,       1=cursor
piscando
#defne lcd_id 1      //     0=decrementa cursor,       1=increm
cursor
       //-------------------------------------------------
       //defnições  de     comandos      HITACHI      44780
       //-------------------------------------------------
#defne lcd_clear
lcd_escreve(0x01,0);       //limpa       lcd
#defne lcd_home
lcd_escreve(0x02,0);       //cursor      1,1
#defne lcd_normal                                lcd_escreve(0x06,0);
//modo normal
#defne lcd_mode_shift      lcd_escreve(0x07,0);       //com  desloc
```

```
#defne lcd_off
lcd_escreve(0x08,0);       //desliga     lcd
#defne lcd_on
lcd_escreve(0x0C,0);       //liga lcd
#defne lcd_cursor_on       lcd_escreve(0x0E,0);       //liga cursor
#defne lcd_cursor_off      lcd_escreve(0x0C,0);       //desliga
cursor
#defne lcd_shift
lcd_escreve(0x14,0);       //desloca     linha
#defne lcd_cur_shift       lcd_escreve(0x18,0);       //desloca
cursor
#defne lcd_4_bit           lcd_escreve(0x28,0);       //4bit,2
linhas,       5x7
#defne lcd_8_bit           lcd_escreve(0x38,0);       //8bits,2
linhas,5x7
```

```
#defne lcd_linha1   lcd_escreve(0x80,0);       //posição    1,1
#defne lcd_linha2   lcd_escreve(0xC0,0);       //posição    2,1
              //-------------------------------------------------
              //seqüência  de     inicialização do     LCD
              //-------------------------------------------------
void   lcd_inicializa(void){
              delay_ms(20);
              lcd_rs =      0;
              lcd_d4 =      1;
              lcd_d5 =      1;
              strobe_lcd;  delay_ms(5);
              strobe_lcd;  delay_us(100);
              strobe_lcd;  delay_ms(5);
              //
              lcd_4_bit;          delay_us(40);//0x28
              lcd_off;                         delay_us(40);//0x08
              lcd_clear;          delay_ms(2);         //0x01
              lcd_on;                                  delay_us(40);
//0x0C
              lcd_normal;  delay_us(40);//0x06
}
              //-------------------------------------------------
              //escreve    um     byte   no     lcd    p/     comando
rs=0,  dados  rs=1
              //-------------------------------------------------
void   lcd_escreve( int    c,     int    rs     ){
              lcd_en =      0;     lcd_rs =      rs;
if(c   &      0x80)  lcd_D7=1;     else   lcd_D7=0;
if(c   &      0x40)  lcd_D6=1;     else   lcd_D6=0;
if(c   &      0x20)  lcd_D5=1;     else   lcd_D5=0;
if(c   &      0x10)  lcd_D4=1;     else   lcd_D4=0;
strobe_lcd;
if(c   &      0x08)  lcd_D7=1;     else   lcd_D7=0;
if(c   &      0x04)  lcd_D6=1;     else   lcd_D6=0;
if(c   &      0x02)  lcd_D5=1;     else   lcd_D5=0;
if(c   &      0x01)  lcd_D4=1;     else   lcd_D4=0;
strobe_lcd;
}
              //-------------------------------------------------
              //posiciona  o      cursor na     linha(l),    coluna(c)
              //-------------------------------------------------
void   lcd_pos(      int    l,     int    c){
              if(l   !=     1)     lcd_escreve(  0xC0   +      c,     0);
              else   lcd_escreve(0x80    +      c,     0);
}
              //-------------------------------------------------
              //direciona  um     caractere    ao     LCD
              //-------------------------------------------------
void   lcd(   int    c){
                     lcd_escreve(c,1);
}
```

```
//*********************************************************
```

Esta é uma aplicação de controle de motor de corrente contínua, utilizando modulação por largura de pulso PWM, que exibe o percentual do ciclo ativo e o tempo de utilização no LCD. O motor utilizado no exemplo, foi um motor para 6 volts e um LCD de 16 caracteres e 2 linhas.




```
#include      <16F628A.h>
#FUSES NOWDT                        //No   Watch  Dog    Timer
#FUSES XT                                                //Crystal    osc
<=     4mhz
#FUSES PUT                                          //Power      Up
Timer
#FUSES NOPROTECT//Code     not    protected    from   reading
#FUSES BROWNOUT     //Reset      when   brownout     detected
#FUSES NOMCLR              //Master     Clear  pin    used   for    I/O
#FUSES NOLVP                        //No   low    voltage      prgming
#FUSES NOCPD                        //No   EE     protection
       //
#use   delay(clock=4000000)
       //-------------------------------------------------
```

```
        //defnições  de     portas I/O
        //-------------------------------------------------
#byte   Ra      =       0x05    //defne       um      rótulo para   o
portA
#byte   Rb      =       0x06    //defne       um      rótulo para   o
portB
                //
#bit    botao_mais              =       Ra.3            //      1       entrada
#bit    botao_menos     =       Ra.2            //      1       entrada
#bit    lcd_EN  =       0x05.0                  //lcd   pino    6
#bit    lcd_RS  =       0x05.1                  //lcd   pino    4
#bit    lcd_d4  =       0x06.7                  //lcd   pino    11
#bit    lcd_d5  =       0x06.6                  //lcd   pino    12
#bit    lcd_d6  =       0x06.5                  //lcd   pino    13
#bit    lcd_d7  =       0x06.4                  //lcd   pino    14

                //
#use    fast_io(a)
#use    fast_io(b)
                //
#defne trisa            0b00001100              //
#defne trisb            0b00000000              //      0=saída
                //incluir       o       drive   lcd     depois das      defnições
do      port    e
                //principalmente        depois da       atribuição      dos
bits.
#include        <lcd_44780.c>
        //-------------------------------------------------
        //declaração das    funções       utilizadas
        //-------------------------------------------------
void    atualiza(void);
void    mais(void);
void    menos(void);
        //
struct E_pwm{
                int     indice;                                 //duty
cycle   do      pwm
                int     percentual;     //1     a       100%
}pwm;
        //
struct E_tempo{
                int     contador;               //contador      de
interrupções    do      tmr0
                int     minutos;                        //minutos,      utilizado
no      relógio
                int     segundos;               //segundos,     utilizado       no
relógio
}tempo;
        //
#int_timer0
void    timer0_isr(void){
                        //resolução     do      timer   =       4ms,    1       seg
```

```
=      1000   ms
                    //     então  1000/4 =      250    vezes  0.004  =
1      segundo
                    ++tempo.contador;
                           //incrementa  o      contador
                    if(    tempo.contador      >=     250){//contador
>=     250    ?
                                  ++tempo.segundos;
                           //sim, incrementa    segundos
                                  tempo.contador      =      0;
                           //zera o      contador     para


                                                      //marcar
outro  segundo
                                  if(tempo.segundos   ==     60){//deu
1      minuto ?
```

```
                                                ++tempo.minutos;
                                    //sim incrementa   minuto
                                                tempo.segundos
=     0;                    //segundo    =     0
                                    };
                                    lcd_pos(    2,    11    );
                              //posiciona linha 2     coluna 11
                                    //imprime    variável     "minutos",
c/    duas  casas  decimais
                                    //ex. 1m     =     01    e     15m
=     15
                                    printf(     lcd,  "%02u",
tempo.minutos);
                                    lcd_pos(    2,    14    );
                              //posiciona linha 2     coluna 14
                                    //imprime    variável
"segundos",  c/    duas  casas  decimais
                                    //ex. 5seg   =     05    e
20seg =     20
                                    printf(     lcd,  "%02u",
tempo.segundos     );
                                    };
}
      //-------------------------------------------------
      //coloca     o     resultado    no    LCD
      //-------------------------------------------------
void  atualiza(void){
            lcd_pos(    1,    12    );
//posiciona  linha 1     coluna 12
            //imprime    a     variável     indice com    3
casas sem   o     zero   a
            //esquerda.  ex    25    =     25    e     não   025
            printf(     lcd,  "%3u", pwm.indice   );
}
      //-------------------------------------------------
      //botão      aumenta      passo de    incremento
      //-------------------------------------------------
void  mais(void){
            pwm.indice   +=    2;    //incrementa de    2     em
2     (indice=indice+2)
            //se   for   =     100   então  não   incrementa,
redefne     para  100
            if(   pwm.indice   >=    100)  pwm.indice   =     100;
            //ajusta     o     duty  cycle  (ciclo ativo) do    pwm
            set_pwm1_duty(     pwm.indice   *     pwm.percentual
);
            //coloca     o     resultado    no    LCD
            atualiza();
}
      //-------------------------------------------------
      //    botão  diminui     passo de    incremento
      //-------------------------------------------------
void  menos(void){
```

```
            pwm.indice    -=     2;     //decrementa de     2      em
2     (indice=indice-2)
            //se   for    =      100    então  não    incrementa,
redefne       para   100
            if(    pwm.indice    <=     2)     pwm.indice    =      1;
            //ajusta     o      duty   cycle  (ciclo ativo) do     pwm
            set_pwm1_duty(      pwm.indice    *      pwm.percentual
);
            //coloca     o      resultado     no     LCD
            atualiza();
}
      //-------------------------------------------------
      //     função principal
      //-------------------------------------------------
void  main() {
```

```
            //resolução  =     1/clock     *     (256  *
RTCC_DIV)
            //0,000.001*(256*16)      =     4ms.
            setup_timer_0(RTCC_INTERNAL|RTCC_DIV_16);     //timer0
            setup_timer_1(T1_DISABLED);
//timer     1     desabilitado
            //resolução         =     16us
            //overfow           =     62
            //interrup   period =     1
            //freqüência =     992hz
            //    1/992  =     0.001  (100ms)     então  1%    =
1ms
            setup_timer_2(T2_DIV_BY_16,62,1);      //timer     2
            setup_ccp1(CCP_PWM);
                                                         //defne
CCP   como  pwm
            //
            setup_comparator(NC_NC_NC_NC);         //desabilita o
comparador
            setup_vref(FALSE);
                                          //desabilita tensão ref.
            enable_interrupts(int_timer0);         //habilita
interrupção
            enable_interrupts(GLOBAL);

            //
            set_tris_a(  trisa  );          //ajusta     a
direção     do    portA
            set_tris_b(  trisb  );          //ajusta     a
direção     do    portB
            //
            Ra    =     0;          //limpa     todos  os    bits
do    portA
            Rb    =     0;          //limpa     todos  os    bits
do    portB
            pwm.indice                    =     20;    //defne
valor  inicial      para   PWM
            pwm.percentual=     1;          //    1/992=0.001
(100ms)     então  1%=1ms
            tempo.contador=     0;          //zera a     contagem
de    interrupções
            tempo.minutos=      0;          //zera a     variável
minutos
            tempo.segundos=     0;          //zera a     variável
segundos
            //delay     para   estabilização de    periféricos
            delay_ms(5);
            lcd_inicializa();   //inicializa o     LCD
            //
            lcd_pos(1,0);
                                          //posiciona  linha  1
coluna 0
            printf(     lcd,   "PWM   Duty:" );    //imprime
            lcd_pos(1,15);
```

```
                                        //posiciona  linha  1
coluna 15
            printf(     lcd,   "%%"   );
                  //imprime    o     caractere    %
            lcd_pos(2,0);
                                        //posiciona  linha  2
coluna 0
            printf(     lcd,   "Tempo:"     );
//imprime
            lcd_pos(    2,     13     );
                                        //posiciona  linha  2
coluna 13
            printf(     lcd,   ":"    );
                        //imprime    caractere    :
            //
            atualiza();               //coloca     o      valor
inicial     do     pwm    no     LCD
            //
            while( true   ){
//loop contínuo
                                    //se   botao_mais   for    diferente
de    1     chama  subrotina
                                    if(    !     botao_mais    )
mais();
                                    //delay      para   debounce
                                    delay_ms(     50     );
                                    //se   botao_menos  for    diferente
de    1     chama  subrotina
                                    if(    !     botao_menos)  menos();
                                    //delay      para   debounce
```

```
                                    delay_ms(     50     );
                };
}
```

A utilização do módulo de LCD é bastante simples, a função lcd(int c) chama a função lcd_escreve(c,1) para cada caractere enviado pela função printf. Desta forma temos o envio de uma string formatada para o módulo LCD com facilidade, no entanto este processo aumenta o consumo de memória de programa fazendo um código mais longo. A critério do leitor pode-se chamar a função lcd_escreve(c,1) diretamente passando os caracteres individualmente a função. As duas estruturas utilizadas no exemplo também contribuíram para o aumento da memória de programa, poderia ter sido utilizada uma estrutura ou somente variáveis sem esta forma de organização estruturada. No entanto como o objetivo é demonstrativo e o programa relativamente pequeno, optei por esta opção.

O drive de LCD, quando escrito, levou-se em consideração o datasheet do controlador de LCD Hitach 44780 e funcionou corretamente inclusive com outros controladores de LCD, como NT3881. Pode ser que com outros modelos possa ser necessário alguma alteração para compatibilização, principalmente das strings de inicialização.

Capítulo

11

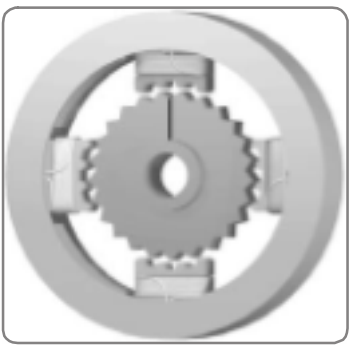
Motores de Passo

11.1- Definição

O moto de passo é um tipo de motor elétrico que é usado quando um movimento exige precisão ou quando deve ser rotacionado em um ângulo exato, estes são encontrados com imã permanente (divididos em; Unipolar, Bipolar e Multiphase) e relutância variável.

Motores de relutância variável usualmente têm três ou às vezes quatro bobinas, com um comum. Já os motores de imã permanentes usualmente têm duas bobinas independentes, com ou sem o comum central, controlado por uma série de campos eletromagnéticos que são ativados e desativados eletronicamente.

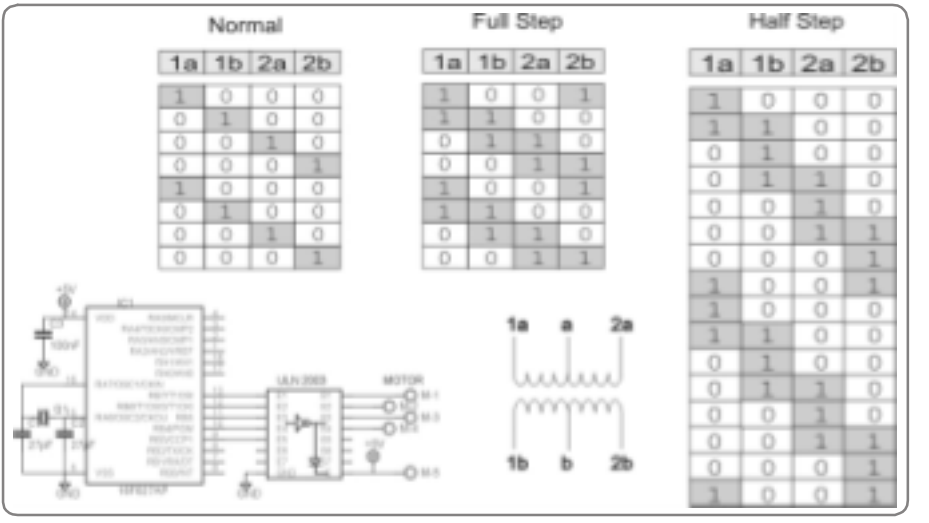
Motores de passo não usam escovas ou comutadores, possuem um número fixo de pólos magnéticos que determinam o número de passos por revolução. Ao se alimentar uma das bobinas é gerado um campo magnético no estator o que induz o rotor a alinhar-se com o campo, gerando um movimento chamado “passo”. Normalmente estes motores possuem 200 passos por volta sendo 360º de rotação, cada passo avança em 1,8º, com alguns controladores pode-se posicionar o estator em micropassos obtendo-se maior precisão



no movimento. Quanto ao torque (que é a medida de quanto uma força que age em um objeto faz com que o mesmo gire), uma característica única deste tipo de motor é a sua habilidade de poder manter o eixo em uma posição segurando o torque sem estar em movimento. Para atingir todo o seu torque, as bobinas de um motor de passo devem receber toda a sua corrente marcada durante cada passo provenientes do controlador do motor. O torque do motor é uma grandeza vetorial da física, definido como a fração da força aplicada sobre um objeto que é efetivamente utilizada para fazer ele girar em torno de um eixo, conhecido como ponto pivô. A distância do ponto do pivô ao ponto onde atua uma força 'F' é chamada braço do momento e é denotada por 'r', dado pela relação vetorial modular |T| = |R||F|sen(Ø).

11.2- Motor de Passo Unipolar

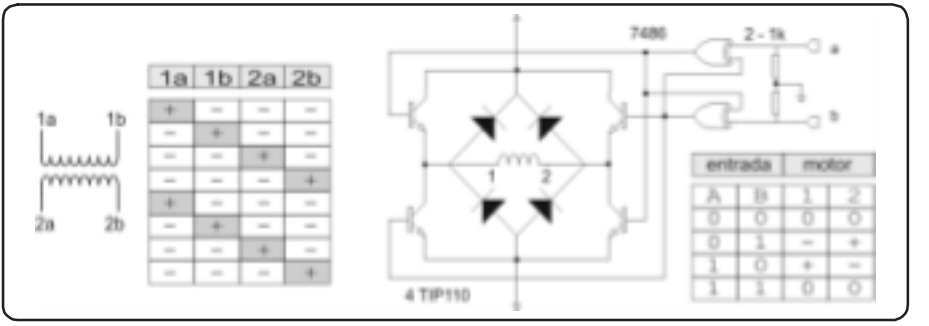
Motores Unipolares, apresentam 5 ou 6 linhas com uma linha comum, sendo relativamente fáceis para controlar com uma simples seqüência de pulsos provenientes de um contador. Ativando-se duas bobinas por passo, consegue-se alto-torque, com consumo de duas vezes mais corrente do circuito. O meio-passo



é alcançado ao combinar as duas bobinas seqüencialmente, duplicando o número de passos e cortando o número de graus de revolução pela metade.

11.3- Motor de Passo Bipolar

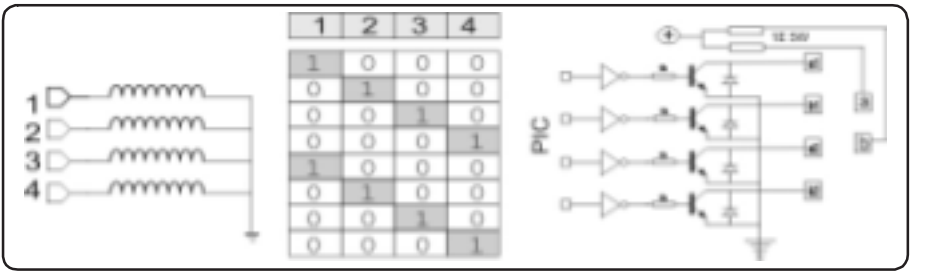
Motores de passo bipolar, embora sejam construídos com os mesmos mecanismos do motor unipolar, estes diferentemente requerem um circuito mais complexo, apresenta uma ótima relação entre tamanho e torque, fornecendo mais torque para seu tamanho que o tipo unipolar. Motores do tipo bipolar são projetados com as bobinas separadas onde a energia flui em uma ou outra direção (necessita inverter a polaridade durante operação) para ocorrer o passo. estes modelos de motor usam o mesmo modelo binário do motor de passo unipolar, unicamente os sinais '0' e '1' correspondem à polaridade da voltagem aplicada a bobina e não simplesmente os sinais 'ligado-desligado'.



O controle destes motores é feito por ponte H, cada bobina do motor de passo necessita de um circuito "ponte H". normalmente estes motores têm 4 fases, conectadas para dois comuns isolados do motor. Os populares L297/298 série do ST Microelectronics, e o LMD18T245 da Nacional Semiconductor são circuitos projetados especificamente para controlar estes motores (ou motores de DC).

11.4- Motor de Passo de Relutância variável

Às vezes referido como motores híbridos, motores de passo de relutância variáveis são os mais simples para controlar em comparação com outros tipos de motores de passo. Sua seqüência dirigida está simplesmente ao energizar cada das bobinas em seqüência



1	2	3	4
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

11.4- Modos de acionamento

Modo Wave Drive

Neste modo uma bobina do motor será acionada de cada vez de modo seqüencial, obtendo um movimento com passo de 90º.

Modo Full Step Drive

Neste modo de acionamento tem-se duas bobinas energizadas ao mesmo tempo o que nos dá um torque muito mais alto com a desvantagem de alto consumo de corrente pelo motor, pelo fato de termos sempre duas bobinas energizadas por vez.

Modo Half Step Drive

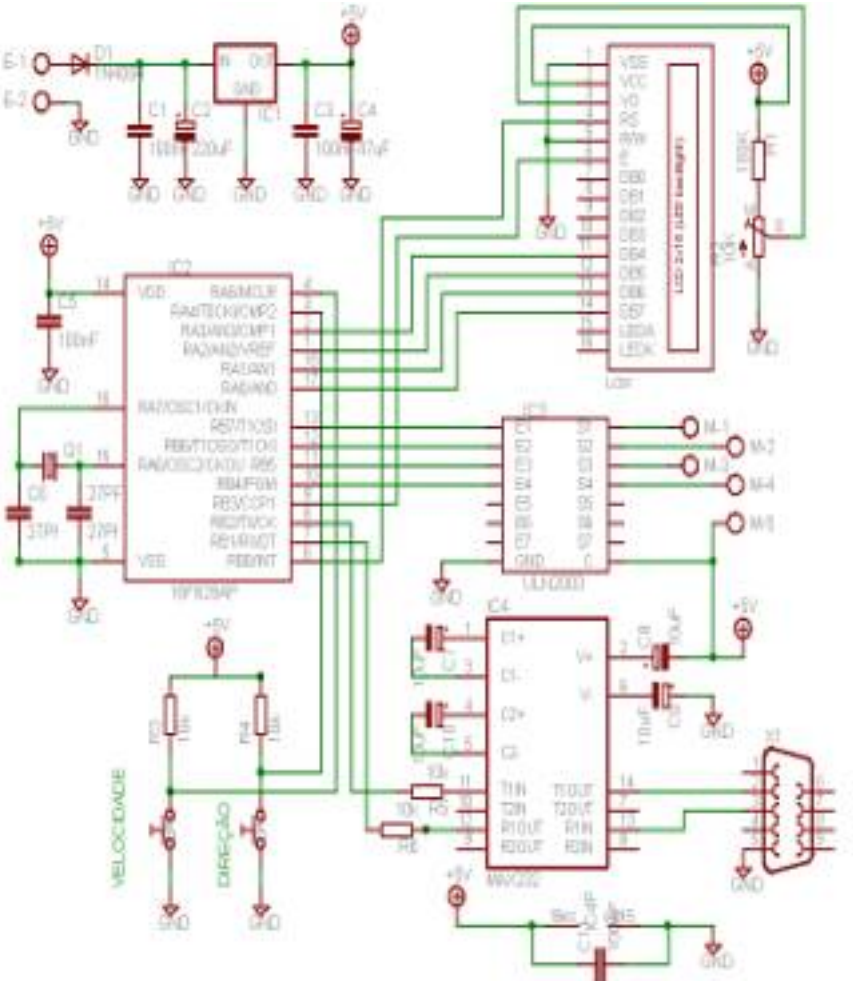
Este modo de acionamento é a combinação dos dois modos anteriores, tendo como vantagem o movimento do motor com meio passo.

Neste exemplo foi utilizado um motor de passo unipolar com controle por meio de dois botões e pelo computador, utilizando o

software desenvolvido para esta finalidade.

```
#include     <16F628A.h>
       //
#FUSES WDT                                    //No   Watch  Dog
Timer
#FUSES XT                                     //Internal   RC     Osc,
no     CLKOUT
#FUSES PUT                                                    //Power
Up     Timer
#FUSES NOPROTECT           //Code not    protected    from   reading
#FUSES BROWNOUT                   //Reset      when   brownout
detected
#FUSES NOMCLR                             //Master     Clear  pin    used
for    I/O
```

```
#FUSES NOLVP                                    //No   low    voltage
prgming
#FUSES NOCPD                                    //No   EE     protection
       //
```



```
      //
#use   delay(clock=4000000,RESTART_WDT)
#use  rs232(baud=9600,parity=N,xmit=PIN_B2,rcv=PIN_
B1,bits=8,RESTART_WDT,     STREAM=PC)
#use   fast_io(a)
#use   fast_io(b)
      //-------------------------------------------------
      //defnições  de     portas I/O
      //-------------------------------------------------
#byte  porta_a      =      0x05   //defne      um     rótulo para   o
portA
#byte  porta_b      =      0x06   //defne      um     rótulo para   o
portB
      //
#bit   botao_dir           =      porta_a.4                //     1
entrada
#bit   botao_vel           =      porta_a.5                //     1
entrada
#bit   lcd_d4                             =      porta_a.3
//     0      saída, lcd    pino   11
#bit   lcd_d5                             =      porta_a.2
//     0      saída, lcd    pino   12
#bit   lcd_d6                             =      porta_a.1
//     0      saída, lcd    pino   13
#bit   lcd_d7                             =      porta_a.0
//     0      saída, lcd    pino   14
      //
#bit   lcd_EN                             =      porta_b.3
      //     0      saída, lcd    pino   6
#bit   lcd_RS                             =      porta_b.0
      //     0      saída, lcd    pino   4
      //
#defne trisa        0b00110000          //0x30 0=saída,     1
entrada
#defne trisb        0b00000010          //0x02 0=saída,     1
entrada
      //
      //incluir    o      drive  lcd    depois das    defnições    do
port
#include     <lcd_44780.c>

      //-------------------------------------------------
      //defnição   dos    comandos     enviados     pelo   computador
      //-------------------------------------------------
#defne CMD_TEMPO                        0x7A
#defne CMD_DIRECAO         0x7B
#defne CMD_MOVE                               0x7C
#defne CMD_SEQ                                      0x7D
      //
#defne PACKET_SOF                             0x01   //cabeçalho   do
pacote
#defne tamanho_buffer      0x1E   //30   bytes
#defne VERIFICA                                          0X00   //
```

```
#defne ADICIONA                                          0X01   //
#defne COMANDO_OK                                 0xFA   //comando    com
sucesso
#defne ERRO_LRC                                          0x08
//reporta    erro   de     lrc
      //-------------------------------------------------
      //declaração das    funções      utilizadas
      //-------------------------------------------------
static short  lrc(   int    *buffer,     int    size,  int    modo   );
void   responde(    int    len    );
void   erroLRC(void);
void   comando_efetuado(   int    cmd    );
void   altera_sequencia(   void   );
```

```
int    timed_getc();
       //
       //
byte   buffer[tamanho_buffer];//buffer   de     recepção      e
transm.
       //
int    contador;                  //contador    de     interrupções  do
timer0
int    velocidade;   //valor      *      1ms,   máx.   255ms
int    indice;                                 //indice      se
sequência    de     passos
int    direcao;                           //0    esquerda     1
direita
int    fag;                                      //opções
internas
int    i;

#bit   fag_movimenta                                          =
fag.0        //motor      parado/em
movimento

int    sequencia[]   =      {0b10000000,

                                                 0b01000000,

                                                 0b00100000,

                                                 0b00010000   };

enum   opcao{ ESQUERDA,     DIREITA      };     //0=esquerda,1=direita
       //
#int_timer0
void   timer0_isr(void){
              //pode movimentar    o      motor?
              if(    fag_movimenta ){
                                  contador      ++;    //incrementa  o
contador      de     interrupções
                                  //núm. de     interrupções  =
velocidade    desejada?
                                  if(    contador      >=     velocidade
){
                                                       contador      =
0;     //sim, zera   o      contador
                                                       //aumenta     ou
diminui       o      indice de     acordo com    a
direção
                                                       if(
(opcao)direcao      !=     DIREITA      )      indice ++;
                                                       else   indice --;
                                                       //não  importa
o      valor  do     indice pois   sempre fará
                                                       //um   AND    com
3      ex:    201    0b11001001    &      3      =      01
                                                       //sempre
```

```
terei  valores      entre  00(0)  e      11(3)
                                                  //após localizar
o      elemento     do     vetor  seqüência    faz
                                                  //AND  com    F0
selecionando somente      o      byte   msb
                                                  //ex:
sequencia[2] &      0xF0   =      0010
                                                  porta_b      =
sequencia[   indice &      0x03   ]      &      0xF0;

                           };
             };
}
             //-----------------------------------------------------
             //     LRC    longitudinal redundance   ciclical
             //-----------------------------------------------------
static short lrc(   int    *buffer,     int    size,  int    modo   ){
             int    x;
```

```
                int     lsb     =       0;
                int     msb     =       0;
                restart_wdt();
                //para cálculo          desconsidera os     dois    últimos
bytes
                for(    x       =       0;      x       <=      (size   -       2)
;       x++     ){
                                        lsb     ^=      buffer[         x       ];
        //      byte    lsb     XOR     buffer[x]
                };

                //
                msb     =       lsb     ^       0xFF;
                //      byte    msb     XOR     lsb
                if(     modo    !=      VERIFICA){
//      adiciona
                                        buffer[size     -1]     =       lsb;
//      byte    lsb
                                        buffer[size                     ]       =
msb;            //      byte    msb
                }
                else

                        //checagem
                                        if(     buffer[size     -1]     ==      lsb
&&

buffer[size                     ]       ==      msb     )       return TRUE;
                                        else    return FALSE;
}
                //
                //-----------------------------------------------------
                //      retorna         o       pacote para     o       PC
                //-----------------------------------------------------
void    responde(       int     len     ){
                int     j;
                lrc(    buffer,         len,    ADICIONA        );      //adiciona
o       LRC
                for(    j       =       0;      j       <       len;    j++     ){
                                        printf("%X",    buffer[         j       ],
PC);
                };
}
                //-----------------------------------------------------
                //responde      ao      PC      informando      ocorrencia      de
erro
                //-----------------------------------------------------
void    erroLRC(void){
                restart_wdt();
                buffer[         0       ]       =       PACKET_SOF;
//cabeçalho
                buffer[         1       ]       =       6;
                                                        //tamanho
                buffer[         2       ]       =       ERRO_LRC;
```

```
	//ocorreu	erro
		buffer[	3	]	=	0;
							//fag
		responde(	buffer[	6	]	);
//envia	ao	PC
}
	//-------------------------------------------------
	//devolve	pacote ao	PC	indicando	recebimento	do
comando
	//-------------------------------------------------
void	comando_efetuado(	int	cmd	){
		restart_wdt();
		buffer[	0	]	=	PACKET_SOF;
//cabeçalho
		buffer[	1	]	=	6;
							//tamanho
		buffer[	2	]	=	cmd;
						//devolve	o	comando
		buffer[	3	]	=	COMANDO_OK;
//fag	comando	entendido
		responde(	buffer[	6	]	);
//envia	ao	PC
}
```

```
	//-----------------------------------------------
	//recebe	nova	seqüência	de	passos do	computador
e
	//preenche	o	vetor	para	sequenciar
	//-----------------------------------------------
void	altera_sequencia(	void	){
		sequencia[0]	=	buffer[3];
		sequencia[1]	=	buffer[4];
		sequencia[2]	=	buffer[5];
		sequencia[3]	=	buffer[6];
}
	//-----------------------------------------------
	//aguarda	um	tempo	limite para	recepção	de
caractere
	//no	UART	do	pic.	aguarda	100	ms
	//-----------------------------------------------
int	timed_getc(){
		long	timeout;	//variável	para	incremento	de
ciclo	de	máq.
		int	retorno;		//uso	temporário,	auxíliar

		timeout=0;		//valor	inicial
		//kbhit()	retorna	true	se	detectar
start	bit	no	uart
		//aguarda	100.000	ciclos de	máquina
		while( !	kbhit()	&&	(	++timeout	<
100000 ))
		//aguarda	10	uS	para	recepção	completa
do	caracter
		delay_us(10);
		//	testa	novamente	e	captura	o
caratere	e	retorna
		if(	kbhit()	)	retorno	=	getc();
		else	retorno	=	0;
		//
		return(	retorno	);
}
	//-----------------------------------------------
	//coloca	os	dados	no	LCD
	//-----------------------------------------------
void	informa(void){
		lcd_pos(1,0);
		printf(LCD,	"Direcao:	%U",	(opcao)direcao	);
		lcd_pos(2,0);
		printf(LCD,	"Veloc.(ms):	%3U",	velocidade);
}
	//-----------------------------------------------
	//	função principal
	//-----------------------------------------------
void	main() {
		//resolução	=	1/clock	*	(256	*
RTCC_DIV)
```

```
              //0,000.001*(256*4) =      1ms.
              setup_timer_0(RTCC_INTERNAL|RTCC_DIV_4);      //timer0
              //
              setup_timer_1(T1_DISABLED);
//timer       1      desabilitado
              setup_timer_2(T2_DISABLED,0,1);   //timer      2
desabilitado
              setup_comparator(NC_NC_NC_NC);          //desabilita o
comparador
```

```
              setup_vref(FALSE);
                                                  //desabilita tensão ref.
              //
              enable_interrupts(int_timer0);          //habilita
interrupção
              enable_interrupts(GLOBAL);
              //
              //
              set_tris_a(  trisa  );           //ajusta     a
direção       do     portA
              set_tris_b(  trisb  );           //ajusta     a
direção       do     portB
              //
              porta_a      =      0;           //limpa      todos  os
bits   do     portA
              porta_b      =      0;           //limpa      todos  os
bits   do     portB
              //
              delay_ms(15);                           //aguarda    15ms
para   estabilização
              lcd_inicializa();   //inicializa o      LCD
              //
              //
              contador                    =      0;
//contador    de     interrupções  de     TMR0
              velocidade    =      100;   //tempo      em     ms     =
(contador     *      1ms)
              direcao                           =      0;
              indice                            =      0;
              fag_movimenta=      1;
              //
              //
              while( true   ){
//loop contínuo
                                    //     testa  os     botões
                                    if(    !      botao_dir     ){

direcao       ++;                                   //incrementa a
direção

direcao       &=     0x03;  //seleciona   os     valores       válidos

                                    };
                                    if(    !      botao_vel     )
velocidade    +=     10;

                                    //zera o      buffer[0]
                                    buffer[0]     =      0;
                                    buffer[0]     =      timed_getc();
//recebe      caractere     ou      zero
                                    //se   caractere     inicial       for
início de     cabeçalho,    então
                                    if(    buffer[0]     ==     PACKET_SOF
){
```

```
                                                                    //
      formato      do     pacote de     dados
                                                                    //     +---------
---------------------+
                                                                    //     |      SOF
|     len    |     cmd    |      DATA   |      LRC    |
                                                                    //     +-----+---
--+-----+-----+------+
                                                                    //     |
1            |            1            |            1            |
      n                   |            2            |      bytes
                                                                    //     +---------
---------------------+
                                                                    //     |
0            |            1            |            2            |
3..          |                                 |      buffer
                                                                    //     +---------
---------------------+
                                                                    //recebe     o
próximo      byte   que    informa      a      quantidade   de
                                                                    //bytes      do
pacote de    dados
                                                                    buffer[1]    =
fgetc(PC);
                                                                    //-1   é      a
correção     buffer que    começa em    0
                                                                    for(   i     =
2;     i     <=     buffer[1]-1;  i++    ){
//recebe     o      restante     do     pacote
```

```
buffer[ i ] = fgetc(PC);
                                        };
                                //se verifcação de
LRC for válida. chaveia o comando
                                        if( lrc(
buffer, buffer[1]-1, VERIFICA)){
                        switch( buffer[ 2 ] ){
                                        case CMD_TEMPO :
velocidade = buffer[ 3 ] ;break;
                                        case CMD_DIRECAO :
direcao = buffer[ 3 ] ;break;
                                        case CMD_MOVE :
fag_movimenta= buffer[ 3 ] ;break;
                                        case CMD_SEQ :
altera_sequencia(); break;
                        };
                        //entendi o comando, sinalizo com
um ok
                        comando_efetuado( buffer[2] );
                                        }
                                //recebí comando com
erro, respondo informando o erro
                                        else erroLRC();
                                };
                };
}
```

Diferentemente do que foi feito no capítulo 10, onde definimos uma variável, incrementamos e depois avaliamos se esta estava dentro dos parâmetros aceitos, desta vez preferimos incrementar uma variável e depois fazer uma máscara com AND e avaliar se o resultado, já que sempre estará dentro dos parâmetros.

antes

```
                        pwm.indice -= 2;
                        //se for = 100 então não
incrementa, redefne para 100
                        if( pwm.indice <= 2) pwm.indice =
```

```
1;
```

agora

```
indice++;
x = sequencia[ indice & 0x03 ];
```

Realmente não importa o valor da variável indice pois com a máscara sempre irá variar nos dois primeiros bits, indo de 00 até 11, ou de 0 à 3, dentro da faixa do vetor definido anteriormente.

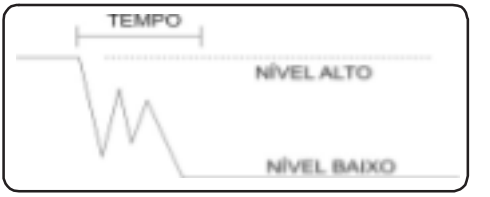
Sem dúvida esta forma mais elegante e clara, consegue-se com um pouco mais de experiência. Não se apreocupe se inicialmente seus programas apresentarem uma forma menos elegante, importa que este funcione para os fins que foi escrito e a medida que for adiquirindo experiência, seus programas, certamente se tornarão mais claros.

Outra diferença em relação ao soft do capítulo 10, foi o fato deste monitorar a porta serial através de uma função com um tempo limite (timeout). Anteriormente colocamos dentro do loop principal a função getc(), agora preferí fazer uma função que testa se há um start bit no buffer de recepção com um tempo limite incrementado em ciclos de máquina. Ao detestar o start bit aguarda outros 10uS para recepção do caracter e o retorna ou em caso de estouro deste tempo retorna 0. Com esta forma resolvemos dois problemas de imediato. O loop pode verificar se há recebimento de caractere na RS232 e posteriormente fazer outras coisas e criamos uma rotina de de bounce para as chaves (botões) com 100ms. "Bouncing" é o nome dado quando se fecha e abre um botão/chave. Em vez de uma mudança de estado do interruptor, temos muitos pontos curtos que são resultados dos contatos da tecla "que saltam" durante a transição. Normalmente um tempo de 20 milisegundos é o suficiente para eliminar este efeito.

Uma derivação do soft anterior pode ser simplificada da seguinte forma:

```
#include     <16F628A.h>
       //
#FUSES WDT                                          //Watch      Dog
Timer  bilitado
#FUSES XT
//oscilador  cristal      <=     4mhz
#FUSES PUT                                          //Power      Up
Timer
#FUSES NOPROTECT    //sem  proteção     para   leitura      da
eprom
#FUSES BROWNOUT            //Resetar    quando detectar     brownout
#FUSES NOMCLR                     //Reset      desabilitado
#FUSES NOLVP                             //prog.      baixa  voltagem
desabilitado
#FUSES NOCPD                             //sem  travar o      chip
       //
#use   fast_io(a)
#use   fast_io(b)
#use   delay(clock=4000000,RESTART_WDT)
       //
#defne tempo  40
```

```
        //
int     passo[]       =      {0b10000000,              //0x80
                      0b01000000,               //0x40
```



```
                    0b00100000,                 //0x20
                    0b00010000   };     //0x10
int    posicao;
       //
void   PulsaMotor(  int    direcao)     {
       if(    direcao      !=     0)     posicao      ++     ;
       output_b(     passo[ posicao      &      0x03   ]);
       delay_ms(     tempo  );
}

void   main(void)   {
              int16  i;                  //defne      variável     para
contagem      de     passos

              set_tris_b(   0x00   );           //defne      a
direção       do     portB
              while( TRUE   ){
//loop permenente
                                    //executa    1000   vezes  para   a
esquerda
                                    for(i  =     0;     i      <
1000;  i++)   {
                                                  PulsaMotor(   0
);
                              };
                                    //executa    1000   vezes  para   a
direita
                                    for(i  =     0;     i      <
1000;  i++)   {
                                                  PulsaMotor(   0
);
                                    };

              };
}
```

Apêndice

Tabela de funções do compilador CCS

comandos e operadores C

comandos	IF, ELSE, WHILE, DO, SWITCH, CASE, FOR, RETURN, GOTO, BREAK, CONTINUE
operadores	! ~ ++ -- + - & | * / % << >> ^ && || ?: < <= > >= == != = += -= *= /= %= >>= <<= &= ^= |=
tipos	TYPEDEF, STATIC, AUTO, CONST, ENUM, STRUCT, UNION

Funções genéricas para manipulação de caracter

atoi(string)	Convertte uma string passada como argumento em um inteiro de 8 bits
atol(string)	Convertte uma string passada como argumento em um inteiro de 16 bits
atoi32(string)	Convertte uma string passada como argumento em um inteiro de 32 bits
atof(string)	Convertte uma string passada como argumento em um n_mero de ponto _utuante
tolower(s1)	Retorna um caractere min_sculo de _a .. z_ para o argumento
toupper(s1)	Retorna um caractere mai_sculo de _A .. Z_ para o argumento
isalnum(s1)	Retorna 0 para falso 1 para verdadeiro se o argumento está entre “0 .. 9”, “A .. Z” ou “a .. z”
isalpha(s1)	Retorna 0 para falso 1 para verdadeiro se o argumento está entre “A .. Z” ou “a .. z”
isamoung(s1)	Retorna 0 para falso 1 para verdadeiro se o argumento valor é um caractere do argumento string
isdigit(s1)	Retorna 0 para falso 1 para verdadeiro se o argumento está entre “0 .. 9”
islower(s1)	Retorna 0 para falso 1 para verdadeiro se o argumento está entre “a .. z”
isspace(s1)	Retorna 0 para falso 1 para verdadeiro se o argumento é um espaço “ “
isupper(s1)	Retorna 0 para falso 1 para verdadeiro se o argumento está entre “A .. Z”

atoi(string)	Convertte uma string passada como argumento em um inteiro de 8 bits
isxdigit(s1)	Retorna 0 para falso 1 para verdadeiro se o argumento está entre “0 .. 9”, “A .. Z” ou “a .. z”
strlen(s1)	Retorna o n_mero de caracteres em s1
strcpy(dest, src)	Copia uma constante ou uma string (src) em RAM para uma string (dest) em RAM terminada com 0
strncpy(s1, s2, n)	Copia uma constante ou uma string (src) em RAM para uma string (dest) em RAM terminada com 0
strcopy()	Copia n caracteres de s1 para s2
strcmp(s1, s2)	Compara s1 com s2
stricmp(s1, s2)	Compara s1 com s2 ignorando mai_sculas e minusculas
strncmp(s1, s2, n)	Compara s1 com s2 n bytes
strcat(s1, s2)	Agrupa s2, com s1
strstr(s1, s2)	Pesquisa s2 em s1
strchr(s1, c)	Pesquisa se c está em s1
strrchr(s1, c)	Pesquisa se c está em s1 em ordem inversa
strtok(s1, s2)	Resulta parte de s1, delimitada pelo caractere delimitador de s2
strspn(s1, s2)	Conta os caracteres iniciais de s1 e também de s2
strcspn(s1, s2)	Conta os caracteres iniciais de s1 que não estão em s2
strpbrk(s1, s2)	Pesquisa se o primeiro caractere de s1 está em s2
strlwr(s1)	Converte s1 para uma sequencia em min_sculo
sprintf(string, cstring, val)	Formata saída de val em string conforme cstring, semelhante a printf()

Standard C Memory

memset(dest, vlr, n)	preenche destino com copia n bytes vlr
memcpy(dest, ori, n)	copia n bytes de origem (ori) para destino (dest)

rotinas de tempo - delay

delay_cycles(valor)	rotina de tempo em ciclos de máquina de 1 á 255, semelhante a um ou vários NOP
delay_us(valor)	rotina de tempo em microsegundos de 0 á 255

delay_ms(valor)	rotina de tempo em milisegundos de 0 á 255

rotinas para CCP captura / comparação e PWM

setup_ccpX()	inicializa o módulo de captira/comparação/pwm
set_pwmX_duty(valor)	ajusta o “Duty Cicle” de PWM com um valor de 10 bits. O valor pode ser 8 ou 16 bits

funções para controle do processamento

sleep()	coloca a CPU no modo de baixo consumo de energia
reset_cpu()	reseta a CPU zerando o registro power-up state
restart_cause()	retorna a causa do _ltimo reset da CPU
disable_interrupts(int)	desabilita a interrupção especi_cada
enable_interrupts(int)	habilita interrupção especi_cada
ext_int_edge()	indica modo de ocorrência H->L ou L->H
read_bank(banco, offset)	Lê um byte no banco RAM especi_cado pelo offset / endereço ex. x = Read_Bank(1,5)
write_bank(banco, offset)	Escreve no banco RAM especi_cado em banco, offset, valor. - ex. Write_Bank(1,5,0)
label_address(label)	Retorna o endereço em ROM da próxima instrução após o label
goto_address(loc)	salta para o endereço em ROM especi_cado pelo argumento
getenv()	retorna informação sobre o ambiente de trabalho

funções para comunicação serial RS232

set_uart_speed(baud)	ajusta a taxa de baud rate
kbhit()	retorna true se houver caracter no buffer de recp
printf(string, vlr)	formata uma string e nvia ao buffer de transm
getc()	recetorna um caracter do buffer de recepção
putc()	coloca uma string no buffer de transmissão
gets(str),puts(str)	recebe uma string do buffer de recepção
puts(string)	envia uma string ao buffer de transmissão

funções para comunicação I2C

i2c_start()	inicializa a condição r/w no barramento I2C

i2c_read()	lê um byte no barramento I2C
i2c_write(dado)	escreve um byte no barramento I2C
i2c_poll()	retorna 0 ou 1 se recebeu um byte no buffer
i2c_stop()	_naliza a condição R/W no barramento

funções para comunicação SPI

setup_spi(modo)	inicializa a interface em modo master ou slave
spi_read(dado)	lê um dado (int 8 bits) na interface
spi_write(valor)	grava um dado (int 8 bits) na interface
spi_data_is_in()	Retorna 0 ou 1 se um dado foi recebido na SPI

funções para manipulação de entrada e saída I/O

output_low(pin)	coloca o pino especi_cado em nível baixo
output_high(pin)	coloca o pino especi_cado em nível alto
output__oat(pin)	especi_ca o pino como entrada
output_bit(pin,vlr)	coloca o pino (pin) com valor (0/1) alto/baixo
input(pin)	retorna o estado de um pino
output_X(valor)	coloca um byte na porta especi_cada
input_X()	retorna um byte na porta especi_cada (paralelo)
port_b_pullups()	ativa resistores de pull-up na porta especi_cada
set_trix_X()	ajusta direção dos pinos (entrada/saída)

funções paramanipulação de timers

setup_timer_X(valor)	con_gura o timer especi_cado
set_timer_X(valor)	especi_ca valor inicial para o timer
get_timer_X(valor)	obtém o valor instantâneo do timer
setup_counters(t,psl)	ajusta tmr0 como contador e prescaler
setup_wdt(valor)	ajusta o tempo de estouro de wdt
restart_wdt()	zera o timer e evita o reset da cpu

funções matemáticas

abs(x)	Retorna o valor absoluto de um n_mero
acos(val)	Retorna o valor do arco coseno
asin(val)	Retorna o valor do arco seno

sin(rad)	Retorna o seno de um ângulo
tan(rad)	Retorna atangente de um ângulo em radiano
atan(val)	Retorna o valor do arco tangente
atan2(val1, val2)	Retorna o arco tangente de Y/X
ceil(x)	Retorna o valor inteiro mais próximo do argumeto - ex. Ceil(21,75) = 22,00
cos(rad)	Retorna o coseno de um ângulo em radiano
exp(x)	Retorna o valor exponencial do arqumento
_oor(x)	Retorna o valor inteiro mais próximo do argumento - ex. Floor(12,56) = 12,00
labs(x)	Retorna o valor absoluto de um inteiro longo
log()	Retorna o Logaritmo natural do argumento
log10()	Retorna o logaritmo em base 10 do argumento - ex. db = log10(read_adc()*(5.0/255))*10
pwr(x)	
fabs(x)	Retorna o valor absoluto de um n_mero de ponto _utuante
sqrt(x)	Retorna a raiz quadrada de um n_mero positivo
div(num, denom)	Retorna o quociente e dizima da divisão do numerador pelo denominador. ex- x=Ldiv(300,250) = 1 e 50
ldiv(inum, idenom)	Retorna o quociente e dizima da divisão do numerador pelo denominador. ex- x=Ldiv(300,250) = 1 e 50

funções para converssão analógica digital

setup_adc_ports(valor)	Ajusta os pinos para ADC com valores diferentes para cada chip, de_nido no arq device.h
setup_adc(modo)	Ajusta Adc em adc_off, adc_clock_internal, adc_clock_external
set_adc_channel(canal)	Especi_ca o canal para a pr_x. leitura. Ex. set_adc_channel(AN0);
read_adc(modo)	Obtem leitura do ADC, “modo” é const. opcional

função para comparação analógica

setup_comparator()	ajusta os módulos dos comparadores analógicos

função para tensão de referência

setup_vref()	estabiliza a volt. vreferência, p/ o comparador analógico ou saída no pino Ra.2

funções para manipulação de bit

shift_right(end,b,vlr)	desloca um bit (lsb) a direita no vetor end
shift_left(end,b,vlr)	desloca um bit (msb) p/ direita no vetor espec.
rotate_right(end,n)	desloca n bit a direita em um vetor ou estrutura
rotate_left(end,n)	desloca n bit a esquerda em um vetor ou est.
bit_clear(var,bit)	coloca o bit da variável (var) em 0
bit_set(var,bit)	coloca o bit da variável (var) em 1
bit_test(var,bit)	retorna true se o bit da variável = 1
swap(byte)	rotaciona o byte. ex 0x45 = 0x54

funções especiais

rand()	retorna um número aleatório
srand(n)	usa o argumento parainiciar uma nova seq.

funções para uso da memória EEPROM

read_eeprom(adr)	lê um byte da eprom no endereço especi_cado
write_eeprom(adr, val)	grava um byte (val) no endereço (adr)
read_program_ eeprom(adr)	Lê um byte da eprom no endereço especi_cado em adr para a memória de programa
write_program_ eeprom(adr)	Escreve em uma área especí_ca da eprom de programa
read_calibration(n)	Lê a posição n da mem. de calibração. Somente PIC 14000

diretivas do compilador - define

#de_ne id	de_ne uma string/comando para subst. no fonte
#if #else, #elif, #endif	expressão condicional de compilação
#error texto	especi_ca uma condição de erro reportado pelo C
#ifdef	testa se uma #de_ne foi de_nida
#include	inclui um arquivo na compilação
#pragma cmd	o mesmo que #de_ne, p/ compatib com outros C
#undef	desfaz uma de_nição efetuada por #de_ne

diretivas do compilador - definições de funções

#use delay clock	determina um valor clock para rotinas de tempo
#use fast_io(porta)	modo de acesso a porta, deve-se utiliz. set_tris( )
#use _xed_io(p,pin)	coloca pino (pin) da porta (p) com direção _xa
#use i2c	utiliza rotinas do drive I2C
#use rs232	utiliza rotinas de comunicação RS232
#use standard_io	modo padrão de direção das portas, o compila-dor ajusta automaticamente a direção.

diretivas do compilador - comtrole de memória

#asm	inícia um bloco com assembler
#bit id=x.y	rotula (nomeia) o bit y do byte x
#byte id=x	especif. o endereço x ao byte id
#endasm	_naliza um bloco com assembler
#locate id=x	parecido com byte, mas impede o uso do ender.
#reserve end	reserva espaço na ram
#rom end={ }	especif. bytes para armaz. na eeprom
#zero_ram	zera a memória ram

diretivas do compilador - dispositivos e interrupções

#device (chip)	especif. o MCU alvo e modo de ponteiro de ram
#fuses	ajusta os fusíveis de funcionamento
#id checksum	inf checkagem do arquivo armaz em id
#id number	identi_ca um identi_cador na palavra de identif.
#type tipo, n	de_ne um tipo var para outro para compatib
#int_default	engatilha um interrupção padrão
#int_global	de_ne int forma global uso restrito
#int_xxx	de_nição de interrupção
#separate	implementa um rotina separadamente

diretivas do compilador - controle da compilação

#case	força compilador diferenciar mai_sc e min_scula
#opt nivel	vível de optimização. padrão é 5
#list	lista compilação no arquivo .LST
#nolist	não lista compilação no arquivo .LST

#org	coloca a função, const em endereço especí_co na eeprom
#ignore_warnings	ignora aviso especi_cado na compilação.

Tabela de conversão de caracteres

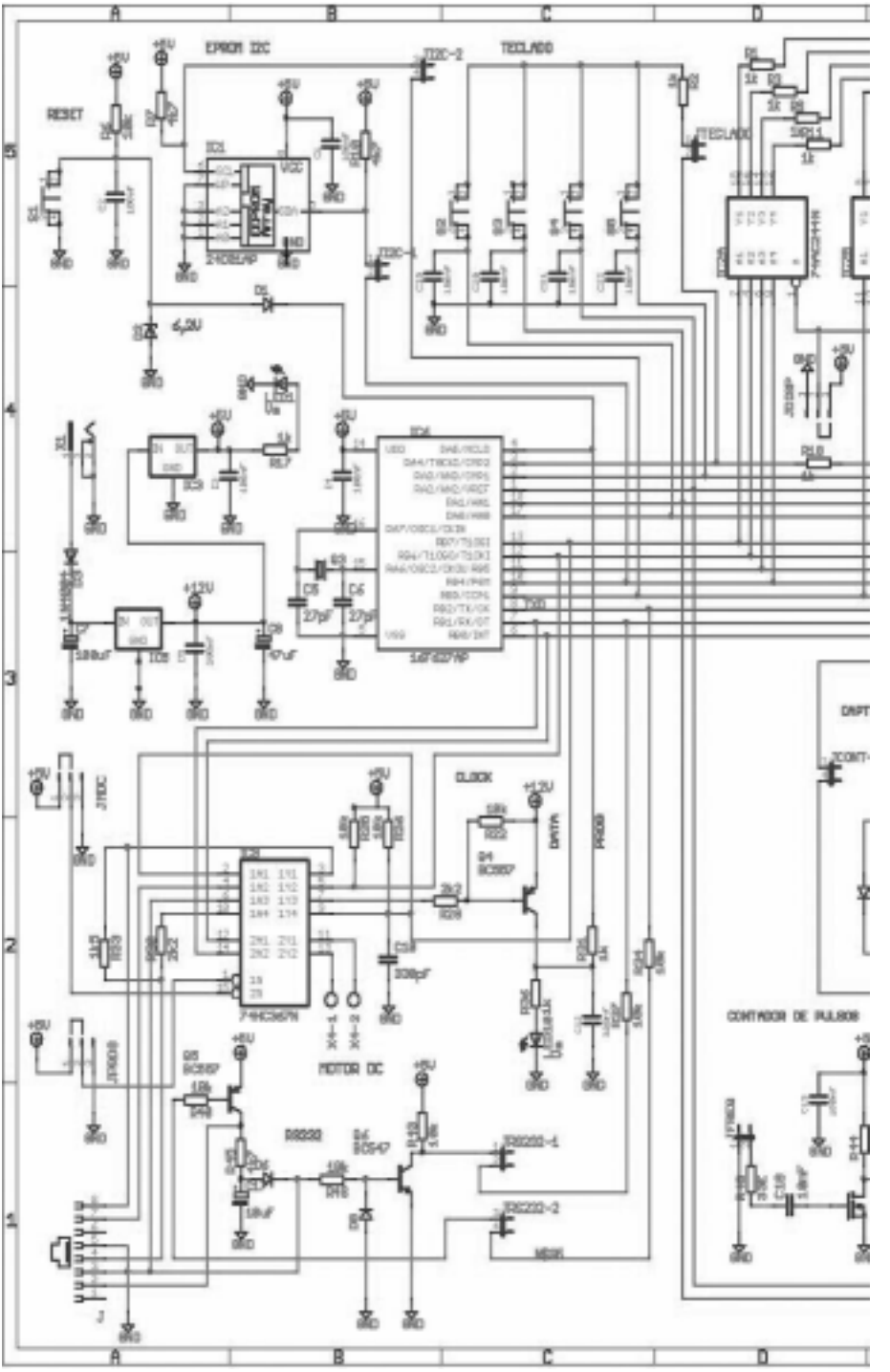
Dec	Hexa	Binário	ASCII	Dec.	Hexa	Binário	ASCII
0	00	00000000	NUL	31	1F	00011111	US
1	01	00000001	SOH	32	20	00100000	espaço
2	02	00000010	STX	33	21	00100001	!
3	03	00000011	ETX	34	22	00100010	“
4	04	00000100	EOT	35	23	00100011	#
5	05	00000101	ENQ	36	24	00100100	$
6	06	00000110	ACK	37	25	00100101	%
7	07	00000111	BEL	38	26	00100110	&
8	08	00001000	BS	39	27	00100111	‘
9	09	00001001	TAB	40	28	00101000	(
10	0A	00001010	LF	41	29	00101001	)
11	0B	00001011	VT	42	2A	00101010	*
12	0C	00001100	FF	43	2B	00101011	+
13	0D	00001101	CR	44	2C	00101100	´
14	0E	00001110	SO	45	2D	00101101	-
15	0F	00001111	SI	46	2E	00101110	.
16	10	00010000	DLE	47	2F	00101111	/
17	11	00010001	DC1	48	30	00110000	0
18	12	00010010	DC2	49	31	00110001	1
19	13	00010011	DC3	50	32	00110010	2
20	14	00010100	DC4	51	33	00110011	3
21	15	00010101	NAK	52	34	00110100	4
22	16	00010110	SYN	53	35	00110101	5
23	17	00010111	ETB	54	36	00110110	6
24	18	00011000	CAN	55	37	00110111	7
25	19	00011001	EN	56	38	00111000	8
26	1A	00011010	SUB	57	39	00111001	9
27	1B	00011011	ESC	58	3A	00111010	:
28	1C	00011100	FS	59	3B	00111011	;
29	1D	00011101	GS	60	3C	00111100	<
30	1E	00011110	RS	61	3D	00111101	=

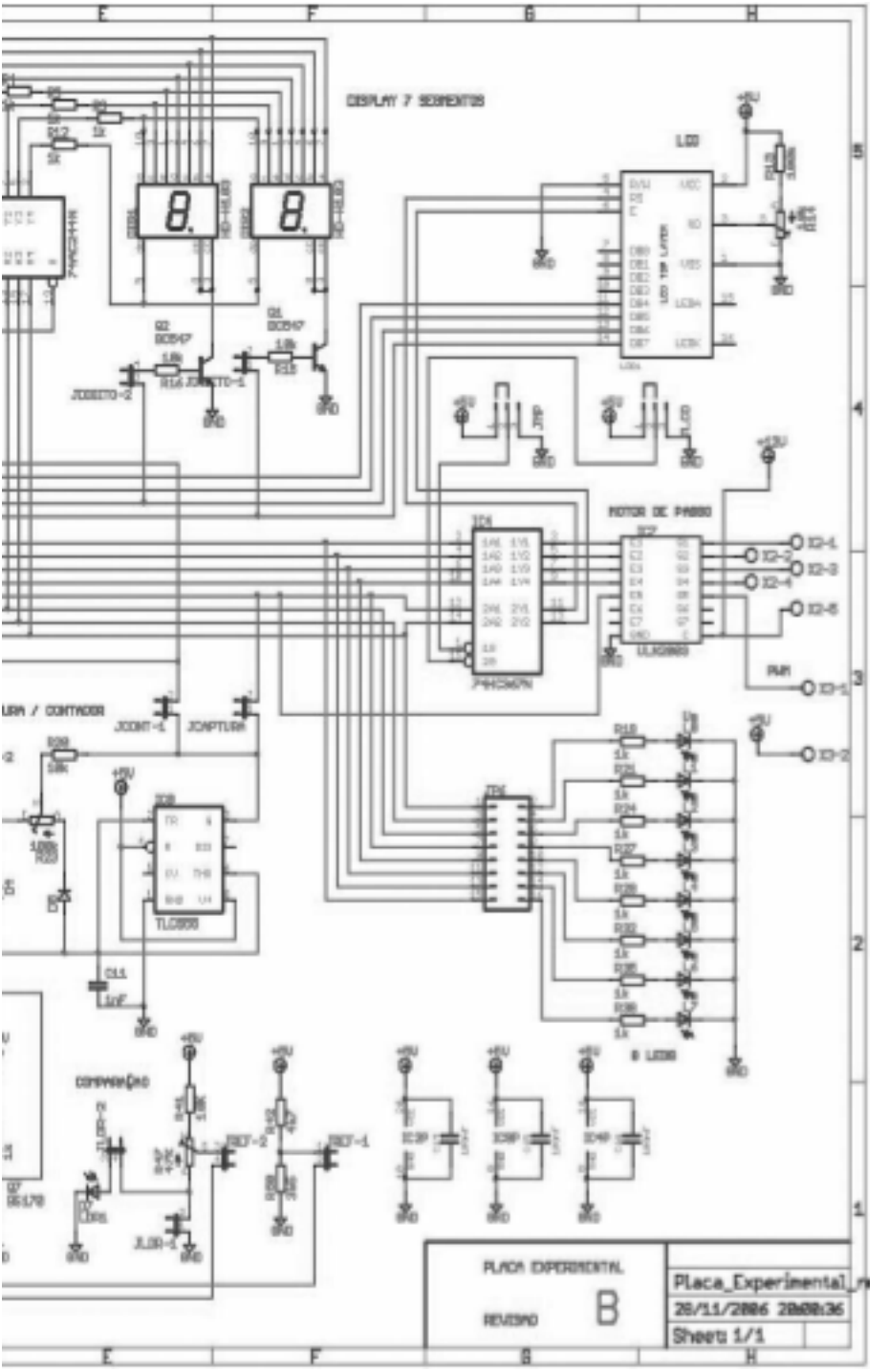
Dec	Hexa	Binário	ASCII	Dec.	Hexa	Binário	ASCII
62	3E	00111110	>	95	5F	01011111	_
63	3F	00111111	?	96	60	01100000	`
64	40	01000000	@	97	61	01100001	a
65	41	01000001	A	98	62	01100010	b
66	42	01000010	B	99	63	01100011	c
67	43	01000011	C	100	64	01100100	d
68	44	01000100	D	101	65	01100101	e
69	45	01000101	E	102	66	01100110	f
70	46	01000110	F	103	67	01100111	g
71	47	01000111	G	104	68	01101000	h
72	48	01001000	H	105	69	01101001	i
73	49	01001001	I	106	6A	01101010	j
74	4A	01001010	J	107	6B	01101011	k
75	4B	01001011	K	108	6C	01101100	l
76	4C	01001100	L	109	6D	01101101	m
77	4D	01001101	M	110	6E	01101110	n
78	4E	01001110	N	111	6F	01101111	o
79	4F	01001111	O	112	70	01110000	p
80	50	01010000	P	113	71	01110001	q
81	51	01010001	Q	114	72	01110010	r
82	52	01010010	R	115	73	01110011	s
83	53	01010011	S	116	74	01110100	t
84	54	01010100	T	117	75	01110101	u
85	55	01010101	U	118	76	01110110	v
86	56	01010110	V	119	77	01110111	w
87	57	01010111	W	120	78	01111000	x
88	58	01011000	X	121	79	01111001	y
89	59	01011001	Y	122	7A	01111010	z
90	5A	01011010	Z	123	7B	01111011	{
91	5B	01011011	[	124	7C	01111100	|
92	5C	01011100	\	125	7D	01111101	}
93	5D	01011101	]	126	7E	01111110	~
94	5E	01011110	^	127	7F	01111111	DEL

Dec	Hexa	Binário	ASCII	Dec.	Hexa	Binário	ASCII
128	80	10000000	Ç	161	A1	10100001	í
129	81	10000001	ü	162	A2	10100010	ó
130	82	10000001	é	163	A3	10100011	_
131	83	10000011	â	164	A4	10100100	ñ
132	84	10000100	ä	165	A5	10100101	Ñ
133	85	10000101	à	166	A6	10100110	ª
134	86	10000110	å	167	A7	10100111	º
135	87	10000111	ç	168	A8	10101000	¿
136	88	10001000	ê	169	A9	10101001	⌐
137	89	10001001	ë	170	AA	10101010	¬
138	8A	10001010	è	171	AB	10101011	½
139	8B	10001011	ï	172	AC	10101100	¼
140	8C	10001100	î	173	AD	10101101	¡
141	8D	10001101	ì	174	AE	10101110	«
142	8E	10001110	Ä	175	AF	10101111	»
143	8F	10001111	Å	176	B0	10110000	_
144	90	10010000	É	177	B1	10110001	_
145	91	10010001	æ	178	B2	10110010	_
146	92	10010010	Æ	179	B3	10110011	│
147	93	10010011	ô	180	B4	10110100	┤
148	94	10010100	ö	181	B5	10110101	_
149	95	10010101	ò	182	B6	10110110	_
150	96	10010110	û	183	B7	10110111	_
151	97	10010111	ù	184	B8	10111000	╕
152	98	10011000	ÿ	185	B9	10111001	_
153	99	10011001	Ö	186	BA	10111010	╫
154	9A	10011010	Ü	187	BB	10111011	_
155	9B	10011011	¢	188	BC	10111100	_
156	9C	10011100	£	189	BD	10111101	_
157	9D	10011101	¥	190	BE	10111110	_
158	9E	10011110	Pts	191	BF	10111111	┐
159	9F	10011111	□	192	C0	11000000	□
160	A0	10100000	á	193	C1	11000001	□

Dec	Hexa	Binário	ASCII	Dec.	Hexa	Binário	ASCII
194	C2	11000010	□	225	E1	11100001	□
195	C3	11000011	□	226	E2	11100010	□
196	C4	11000100	□	227	E3	11100011	_
197	C5	11000101	□	228	E4	11100100	□
198	C6	11000110	_	229	E5	11100101	□
199	C7	11000111	_	230	E6	11100110	□
200	C8	11001000	_	231	E7	11100111	□
201	C9	11001001	□	232	E8	11101000	□
202	CA	11001010	_	233	E9	11101001	□
203	CB	11001011	_	234	EA	11101010	□
204	CC	11001100	_	235	EB	11101011	□
205	CD	11001101	□	236	EC	11101100	_
206	CE	11001110	_	237	ED	11101101	□
207	CF	11001111	_	238	EE	11101110	□
208	D0	11010000	_	239	EF	11101111	□
209	D1	11010001	_	240	F0	11110000	□
210	D2	11010010	_	241	F1	11110001	□
211	D3	11010011	_	242	F2	11110010	_
212	D4	11010100	_	243	F3	11110011	_
213	D5	11010101	□	244	F4	11110100	□
214	D6	11010110	□	245	F5	11110101	□
215	D7	11010111	_	246	F6	11110110	□
216	D8	11011000	_	247	F7	11110111	_
217	D9	11011001	□	248	F8	11111000	□
218	DA	11011010	□	249	F9	11111001	□
219	DB	11011011	_	250	FA	11111010	□
220	DC	11011100	_	251	FB	11111011	_
221	DD	11011101	_	252	FC	11111100	□
222	DE	11011110	_	253	FD	11111101	□
223	DF	11011111	_	254	FE	11111110	_
224	E0	11100000	□	255	FF	11111111	

Os caracteres extendidos de 128 à 255 estão no formato padrão Unicode® IPM PC.


EPROM I2C
TECLADO
RESET
4,0V
+12V
CLOCK
DATA
MOTOR DC
CONTADOR DE PULSOS
74HC367N
16F627AP
27pF
27pF
330pF


DISPLAY 7 SEGMENTOS
LCD
MOTOR DE PASSO
PWM
TLC5615
PLACA EXPERIMENTAL
REVISÃO
B
Placa_Experimental_r
20/11/2006 20:00:36
Sheet: 1/1

Referência

Lucent Tecnologies, história do transistor
acesso em http://www.lucent.com/minds/transistor em 08/02/06

Invenção do Transistor
acesso em library/inventors/bltransistor.htm?rnk=r1&terms=what+is+a+transistor%3F em 12/02/06

Invenção do circuito integrado
acesso em http://inventors.about.com/education/inventors em 03/03/06
acesso em library/inventors/blmicrochip.htm?rnk=r3&terms=microchip em 03/03/06
acesso em http://www.ti.com/corp/docs/kilbyctr/jackbuilt.shtml em 05/03/06

U.S. Patent 3,138,743 to J.S. Kilby, 1959, datasheet
acesso em http://www.eepatents.com/feature/3138743.html?m02d01 em 05/03/06

Jack Kilby
acesso em http://web.mit.edu/invent/www/inventorsI-Q/kilby.html 05/03/06
acesso em http://www.ti.com/corp/docs/kilbyctr/interview.shtml em 05/03/06
acesso em http://www.ti.com/corp/docs/kilbyctr/jackbuilt.shtml em 06/03/06

J. Kilby, bibliografia
acesso em http://www.ti.com/corp/docs/kilbyctr/jackstclair.shtml 06/03/06

Histórico dos microprocessadores
acesso em http://www.cs.uregina.ca/~bayko/cpu.html em 12/03/06

Fairchild Semiconductor, história
acesso em http://www.fairchildsemi.com/company/history.html em 28/03/06.

Bell Labs, história

acesso http://www.lucent.com/corpinfo/history.html em 28/03/06.

Memória eletrônica

acesso em http://pt.wikipedia.org/wiki/Memória_(eletrônica)

Boole, George

acesso http://scienceworld.wolfram.com/biography/Boole.html em 03/04/06.

Boolean algebra

acesso em http://en.wikipedia.org/wiki/Boolean_algebra em 03/04/06

Ken Bigelow, digital logic

acesso em http://www.play-hookey.com/digital/ em 04/04/06

Ritchie, Dennis MacAlistair, bibliografia

acesso em http://en.wikipedia.org/wiki/Dennis_Ritchie em 08/04/06.

Unix, história

acesso em http://en.wikipedia. org/wiki/U nix em 04/06/06

PIC16F627/8 Datasheet

acesso em ww1.microchip.com/downloads/en/DeviceDoc/40044C.pdf em 22/04/06

MPLAB 7.0 - download e notas técnicas

acesso em www.microchip.com em 22/04/06

CCS PIC C Compiler - manuais e datasheets

acesso em http://ccsinfo.com em 22/04/06

RS232 interface

acesso em http://www.arcelect.com/rs232.htm em 03/07/06

LCD, funcionamento

acesso em http://pt.wikipedia.org/wiki/LCD em 13/07/06

Sixca Knowledge on the cyber

acesso em http://www.sixca.com/micro/mcs51/lcd/index.

html em 13/07/06

Hitachi 44780 - LCD Controler

acesso em http://www.datasheetcatalog.com em 13/07/06

acesso em NT3881 LCD controler

http://www.datasheetcatalog.com em 13/07/06

Jones, Douglas W. Basic Stepping Motor Control Circuits

acesso em http://www.cs.uiowa.edu/~jones/step/circuits.
html em 21/07/06

Johnson, Jason

acesso em http://www.eio.com/jasstep.htm em 21/07/06

Motores de passo

acesso em http://pt.wikipedia.org/wiki/Motor_de_passo
em 22/07/06

tabela de caracteres ASCII

acesso em www.asciitable.com em 28/10/06

JDM, pic programer

acessoe em http://www.jdm.homepage.dk/newpic.htm

IC-prog, pic programer

acesso em http://www.ic-prog.com/index1.htm

Agradecimentos:

à Deus, primeiramente,

a todos que me incentivaram,

Ao Sr. Antonio Ilídio Reginaldo da Silva,
diretor da escola Senai de Catalão-Go.

O sol e o vento discutiam sobre qual dos dois era o mais forte.

O vento disse:

- Provarei que sou o mais forte. Vê aquela mulher que vem lá embaixo com um lenço azul no pescoço? Vou mostrar como posso fazer com que ela tire o lenço mais depressa do que você.

O sol aceitou a aposta e recolheu-se atrás de uma nuvem. O vento começou a soprar até quase se tornar um furacão, mas quanto mais ele soprava, mais a mulher segurava o lenço junto a si. Finalmente, o vento acalmou-se e desistiu de soprar.

Logo após, o sol saiu de trás da nuvem e sorriu bondosamente para a mulher. Imediatamente ela esfregou o rosto e tirou o lenço do pescoço. O sol disse, então, ao vento:

- Lembre-se disso: “A gentileza e a amizade são sempre mais fortes que a fúria e a força”.

www.padremarcelorossi.com.br, 01/11/2006