Microcontroladores PIC16F84 e PIC16F628

Esta apostila foi escrita no final de Julho de 2004 e teve a colaboração de:

Derli Bernardes
Ivon Luiz
José Domingos
Luiz Cláudio
Edson Koiti
Luciana Petraites

Para completar o aprendizado desta apostila baixe os seguintes arquivos da página www.luizbertini.net/microcontroladores/microcontroladores.html

- telas do MPLAB 5.7.40
- sirene.rar
- teclado_bas.rar
- MT8870.rar
- DELAY.rar
- PIC16F84A.rar
- PIC16F62XA.rar
- Manual Pratica.rar
- Conversão do 84 para o 628.rar
- PIC16F627_PIC16F628
- comandos.rar – toda as instruções em uma só tabela.
- display7seg.rar
- picdisp.rar

Todo este material é fornecido gratuitamente.

Siga as instruções da página e desta apostila.

O autor: o autor é um cara maluquinho que gosta muito de eletrônica e estuda microcontroladores desde 1997. Já leu um monte de livros, já escreveu um monte de livros, já fez um monte de projetos com e sem PICs. Mas não é nem um pouco melhor do que ninguém.

Agradeço a você que fez o download desta apostila e espero que visite minha página de livros e compre meus outros livros e baixe minhas outras apostilas.

Espero que você compre meu livro sobre PICs. Sai em breve e é bem mais completo que está apostila.

Nem todos os projetos apresentados estão completos dentro dela, ou seja, você terá que baixar mais arquivos da minha página sobre microcontroladores. Lá na página você encontrará algumas dicas de como usar este material. Esta apostila não é perfeita, pois eu também não sou perfeito.

Lembre-se: se eu aprendi microcontroladores, você também consegue.

Abraços.

Luiz Bertini

Capítulo 1
MICROPROCESSADORES E MICROCONTROLADORES

E tudo começou com os **microprocessadore**s.
Primeiro eram de **4 bits**, depois de **8 bits** e assim por diante.
Mas o que é um microprocessador? É um chip, com alta integração de componentes, que precisa de **memória RAM externa**, **memória EEPROM externa**, **HD** para armazenamento de programas e outros diversos periféricos. Ou seja, você usa um microprocessador dentro de um microcomputador e ele, em conjunto com um monte de outros CIs, trabalham direitinho.
Por outro lado um **microcontrolador** é um componente completo por si só. Basta um programa e ele executa uma função específica.
Dentro do microcontrolador temos memória RAM, FLASH, EEPROM ou E^2PROM.
Mas, tudo isto começou com a lógica digital baseada na condução ou não de transistores, **Fets** ou diodos entre outros. Mas para não entrar neste detalhe, vamos simplificar resumindo esta lógica em dois números ou como falamos muito, em dois dígitos: O **0** (*zero*) e o **1** (*um*).
Quando falamos em **0** (*zero*), estamos nos referindo a **zero Volt DC** e quando falamos em **1** (*um*) estamos falando em + Vcc. No caso dos **microcontroladores PIC**, geralmente, estaremos falando que:

0 = GND e 1 = 5Vcc

Tenha em mente, então, que a lógica digital se baseia em níveis de tensão contínua. E que os microcontroladores usam a lógica digital, e que um programa de computador ou para um microcontrolador, por mais complexo que seja, se resume em uma enorme quantidade de **0** (*zero*) e **1** (*um*) ordenados corretamente.

Capítulo 2
TIPOS DE MEMÓRIAS

Já falamos sobre alguns tipos de memórias, vamos agora ver com mais detalhes o que elas são e como funcionam além de suas aplicações, é claro.

2.1. Memória ROM:
É um tipo de memória que só pode ser gravada uma vez e não pode ser apagada. Existem **PICs** que só possuam este tipo de memória e que se você gravar um programa errado já era.
Onde temos uma memória deste tipo? Dentro de um computador pessoal, é a famosa "*BIOS*". Mantém a gravação sem alimentação.

2.2. Memória RAM:
É um tipo de memória que pode ser gravada, apagada, regravada, mas, que tem um porém, desligando a alimentação dela todos os dados se perdem. É muito utilizada para gravar informações temporárias que serão utilizadas em um programa. Não mantém a gravação sem alimentação.

2.3. Memória EPROM:
Este tipo de memória permite a gravação, mas, para apagar os dados é necessário um **banho de luz ultravioleta**, para isto, este tipo de memória tem uma janela. Existem PCs com este tipo de memória e que são popularmente chamados de "*PICs janelados*". Mantém a gravação sem alimentação.

2.4. Memória EAROM:
Este tipo de memória pode ser gravada e para apagá-la basta um nível de tensão adequado. Um inconveniente deste tipo de memória usada em muitos TVs antigos, é que ela precisa de uma tensão de **- 30 Volts** para funcionar corretamente. Mantém a gravação mesmo sem alimentação.

2.5. Memória EEPROM ou E2PROM:
Está é a evolução das memórias anteriores. Podem ser gravada e desgravada centenas de vezes apenas com a aplicação de níveis corretos de tensão. Um exemplo prático deste tipo de memória é a famosa "*24C04*" e toda a sua família. Mantém a gravação mesmo sem alimentação.

2.6. Memória FLASH:
É a suprasumo das memórias, mais rápida de gravação e a regravação muito mais fácil. Permite uma infinidade de usos. Desde a gravação de áudio e vídeo digitalizado até um programa em um **PIC**. Mantém os dados e a gravação mesmo sem alimentação.

Capítulo 3
O QUE É UM REGISTRADOR

Um registrador é um endereço na memória, que em nosso caso são formados por **8 números**, ou melhor, **8 bits**. Cada **bit** deste pode assumir o valor de **1** (*um*) ou **0** (*zero*).
Temos então um endereço que podemos chamar de registrador. Este registrador terá **8 bits**. Estes bits podem ser **0** (*zero*) ou **1** (*um*).
Muitas vezes chamamos estes **bits** de **Flags**. Flags então é um bit dentro de um registrador.
Mas o que é um **Bit**?
Imaginemos **8 posições**:

Tabela 3.1

Cada posição pode ter um valor de **0** (*zero*) ou **1** (*um*). Esta posição com um valor é um bit, ou melhor, bit é o valor que há dentro da posição.
Quando juntamos **8 bits**, como no exemplo a seguir:

"00110011" ⇒ Chamamos isto de **byte**. Um **byte** então é um conjunto de **8 bits**.

Às vezes os bits podem ser agrupados em números de **4**, veja:

"0101" ⇒ Damos a isto o nome de "*nibble*".

Outras vezes os bits podem se agrupar com números maiores que **8**, veja:

"001100110011" ⇒ Chamamos isto de "*Word*":

Nos pics existem registradores especiais, vamos citar os três mais importantes:

Intcon Option-reg Status

Podemos através dos **bits** ou **flags** destes registradores "*ajustar*" o funcionamento do **PIC** ou ler como ale está operando, para isto basta escrever ou ler os **flags**.

Capítulo 4
ARQUITETURA DE CONSTRUÇÃO

Os primeiros microcontroladores usavam uma estrutura interna que tinha apenas um caminho para trafegar os dados e endereços.
Lembre-se, **dados** são informações e **endereços** são os locais onde serão armazenados os dados.
O nome mais correto para este "*caminho*" é **Bus**. Com apenas um **bus** tínhamos que trafegar hora endereços, hora dados.
Esta arquitetura de construção recebe o nome de "*Von Neumann*".
Hoje em dia, os **PICs** particularmente, trabalham com dois **bus**. Um para dados e outro para endereços. Desta forma ele fica mais rápido, pois podem fazer duas coisas ao mesmo tempo.
Este tipo de arquitetura recebe o nome de "*Harvard*".
Os **PICs** possuem ainda um outro recurso, chamado de "*Pipeline*". Com este recurso o microcontrolador consegue buscar uma informação enquanto processa outra.
Os microcontroladores mais antigos usavam um conjunto de instruções conhecidos por "*CISC*". Este conjunto, ou **set** de instruções possuía mais de **100 instruções**, o que dificultava a memorização do programador.
Os **PICs** usam um **set** ou conjunto de instruções reduzidas, chamados de "*RISC*". Este set tem entre **33** a **35 instruções** dependendo do **PIC**. Isto ajuda na memorização, mas, exige mais do programador. É como falar fluentemente um outro idioma, conhecendo poucas palavras, all right?

Capítulo 5
PIC 16F84-04

Vamos agora ver a pinagem e algumas características do **PIC 16F84-04**.

5.1. Pinagem:

Fig. 5.1

Esta pinagem corresponde ao encapsulamento **Dual In Line** ou **DIP**.
O pino 15 também pode ser usado como saída do oscilador.
O pina 16 também pode ser usado como entrada do oscilador.

5.2. Características Elétricas:

Alimentação de **2** a **6Vcc**, o mais comum é **5Vcc**. Consumo de corrente entre **26µA** a **2mA**.
A corrente varia, pois o **PIC** tem um consumo diferente de acordo com a freqüência do **clock** e sua condução de funcionamento.
- Com clock de 4MHz ≅ 2mA;
- Com clock de 32 kHz ≅ 150µA;
- Em stand-by ≅ 70µA.

É bom lembrar que este é o consumo do **PIC**, caso tenhamos **4 Leds** acesos em suas saídas e cada led consuma **10mA**, deveremos somar **40mA** para saber o consumo total de corrente e quanto a fonte deve fornecer.
Normalmente uma fonte com o **CI LM7805** é suficiente para a maioria dos projetos.
Cada pino tem uma função, mas antes de falarmos deles individualmente, vamos ver as duas portas que ele apresenta. Mas, você pergunta: o que é porta (é o que tem na entrda da sua casa??!!!)?
Porta é um conjunto de terminais que podem funcionar como entrada ou saída ou ambos e que tem um registrador próprio. Neste **PIC** temos o "*Port A*" e "*Port B*" que chamamos de "*porta e portb*".
O **portb** é composto pelos **pinos 6**, **7**, **8**, **9**, **10**, **11**, **12** e **13** e que correspondem a:

Ra0, Ra1, Ra2, Ra3, Ra4, Ra5, Ra6 e Ra7.

Podemos perceber que são **8 bits**. Esta porta terá seus terminais definidos como entrada ou saída através dos valores que colocarmos nos **flags** do registrador **Trisb**.
O porta é composta pelos **pinos 17**, **18**, **1**, **2** e **3**, que correspondem a: **Ra0**, **Ra1**, **Ra2**, **Ra3** e **Ra4**.
Esta porta terá seus pinos definidos como entrada ou saída de acordo com os ajustes dos **flags** do registrador **Trisa**.
Podemos perceber que o porta só tem **5 bits**, mas, o registrador deve ser ajustado como se ele tivesse **8 bits**.
É comum usarmos o porta, ou parte dela, como entradas onde serão ligadas chaves de pressão, por exemplo.
Também é comum usarmos o portb, ou parte dela, como saída.

5.3. Função de Cada Pino:

Pino 1 ⇒ I/O – Porta, bit 2 ou Ra2;
Pino 2 ⇒ I/O – Porta, bit 3 ou Ra3;
Pino 3 ⇒ I/O ou entrada do timer0 – Porta, bit 4 ou Ra4;
Pino 4 ⇒ Reset – MCLR\ - a barra indica que o reset é feito colocando este pino em 0 Volt, portanto, para que o Pic funcione, ele deve estar em + Vcc;
Pino 5 ⇒ Vss ou terra;
Pino 6 ⇒ I/O ou interrupção externa – Port B, bit 0 ou Rb0;
Pino 7 ⇒ I/O – Port B, bit 1 ou Rb1;
Pino 8 ⇒ I/O – Port B, bit 2 ou Rb2 ;
Pino 9 ⇒ I/O – Port B, bit 3 ou Rb3 ;
Pino 10 ⇒ I/O – Port B, bit 4 ou Rb4 – pode ser usado como interrupção;
Pino 11 ⇒ I/O – Port B, bit 5 ou Rb5 – pode ser usado como interrupção;
Pino 12 ⇒ I/O – Port B, bit 6 ou Rb6 – pode ser usado como interrupção;
Pino 13 ⇒ I/O – Port B, bit 7 ou Rb7 – pode ser usado como interrupção;
Pino 14 ⇒ Vcc ou Vdd;

Pino 15 ⇒ Osc 2 onde deve ser ligado um terminal do ressonador ou cristal. Caso seja usado um clock formado por uma constante RC, ele será a saída do clock dividido por 4;
Pino 16 ⇒ Osc 1 onde deve ser ligado o outro terminal do cristal ou ressonador. Caso se use um circuito RC este pino será a entrada.

Fig. 5.2

Pino 17 ⇒ I/O – Port A, bit 0 ou Ra0;
Pino 18 ⇒ I/O – Port A, bit 1 ou Ra1.

5.4. Outras Características:

1K de espaço para palavras ou **Words** de **14 bits** para programa. 1K corresponde a **1024 posições** e em cada posição desta, pode ser colocada uma instrução do programa. Esta memória é a **Flash**.
64 bytes de memória **EEPROM** para dados (*como senhas e outras informações*).
68 bytes (*lembre-se um byte é um conjunto de 8 bits*) de memória **RAM**, que é onde guardamos nossas variáveis.

Pilhas ou Stack: Com **8 níveis** (*imagine que você quer guardar 8 caixas iguais uma sobre a outra, cada caixa contém algumas coisas. Pois bem, cada caixa é um nível*). Há de se ter cuidado para não "derrubar" esta pilha ou como se diz em programação, estourar a pilha (*vai que dentro de uma pilha você colocou dinamite.*).

15 registradores na memória **RAM**: para controle do **PIC** e de tudo o que estiver conectado com ele, também chamado de "*periféricos*".

Timer: de 8 bits que pode ter sua contagem (um timer conta a passagem de tempo) através de um divisor chamado de "*prescaler*" (*se você conhece o funcionamento de um PLL isto é fácil*).

13 – I/O: treze pinos que podem ser configurados, definidos como entrada ou saída e isto individualmente.

Pode usar vários tipos de **osciladores**.

Pode entrar em modo **Sleep** (*dormindo, mas atento*).

Capacidade suficiente de corrente em suas saídas para acender os **leds** e controlar o corte e a saturação de transistores.

Pode gerenciar interrupções (*interrupção é um comando interno ou externo, que manda o PIC ir para um endereço específico do programa, fazer o que estiver lá e depois voltar*).

Pode ter o seu programa protegido de forma a evitar que "*alguém*" copie o seu programa.

Tem também um **Watch dog** *isto seria, se traduzindo, um cão de guarda com um relógio que está de olho nos horários e tempos do PIC. (Como se já não bastava, o cartão de ponto, o controle de acesso por RFID e os chefes chatos que, graças a Deus eu nunca tive* Mas chega de falar bobagem).

O **Watch dog** é um contador independente de tudo dentro do **PIC**, que reseta o mesmo, após um certo período. Para que serve isto? Digamos que você fez um projeto com um **PIC** para controlar um elevador, dando há um pico de tensão o **PIC** travou fazendo com que o elevador pare. Depois de um curto período o **Watch dog** reseta o **PIC**, o programa volta a funcionar e o elevador volta a funcionar também.
Há de saber como se travar bem este cão, mas isto é, literalmente, outro capítulo.

5.5. Resumo:

A memória de programa pode ter **12**, **14** ou **16 bits**, dependendo do PIC. Nesta memória é que escrevemos nossos programas. No **PIC 16F84** ela tem **14 bits** e a esses **14 bits** damos o nome de "*Word*" (*palavra*). Com **14 bits** e **1K** de espaço o número máximo de bits que conseguiremos armazenar, será de $\mathbf{2^{14}}$ que é igual a **16.384 bits**.
A memória de programa é **Flash** e não volátil, ou seja, não se apaga quando desligamos o aparelho.
A memória de dados é formada por **8 bits**, que recebem o nome de "*byte*". Todo os dados (*aquele cubo com pontinhos?*) armazenados nela se perdem quando a alimentação é desligada. E é ela quem define de quantos bits é o microcontrolador, como ele tem **8 bits** o **PIC 16F84** é de **8 bits**.
Você não acredita que os dados se apagam quando você desliga o circuito?
Tá bom, então faça um programa onde você possa gravar uma senha (*mas sem usar a EEPROM seu espetinho ou espertinha*) e usá-la depois para comandar uma das saídas do PIC. Grave a senha, teste se ela funciona. Desligue o PIC e tente usá-la novamente.
Mas, você não sabe fazer um programa?
Isto é só uma questão de tempo ou como eu costumo dizer, de quilometragem.
Procure ler todos os livros que dizem respeito ao PIC e programação.

5.6. Observação Interessante:

Para saber quantas instruções cabem na memória de programa, devemos conhecer o tamanho da memória de programa. Será? Veremos. Lembre-se que a memória de programa é de **2 elevado** a **14**. Dois, pois, estamos trabalhando em binário, **0** e **1** e **14**, pois este é o tamanho, em **bits**, de cada posição. Desta forma teremos:

$2^{14} = 16.384$

$16.384 \div 14 = 1.170$

Podemos ter na memória de programa 1.170 instruções. Dá um "*programão*".
Vimos no começo deste capítulo que eu chamei o **PIC** de **PIC 16F84-04**, mas por quê? Vamos ver em partes:

Alguns temas estão meios confusos? (*No problem, don't worry, by happy*). Sem, problemas vamos voltar a falar sobre eles.

Capítulo 6
PIC 16F628-04

Este capítulo deverá ser mais curto do que o anterior, mas vamos lá, a vida é uma agradável surpresa (*basta saber olhar para o lado certo*).

2K de memória para instruções de **14 bits**. Lembre-se que esta memória é **Flash** (*não é um Flash, portanto não façam pose*). São **2,048** posições, dá para perceber que podemos gravar um programa maior nele do que no **16F84**.

128 bytes de memória **EEPROM** (*o processo asmático de ensino alerta: 1 byte é constituído de 8 bits*).

224 bytes de **RAM** para que você encha de variáveis.

Um **Stack** (*não de frango*) ou pilha com **8 níveis**. Não se preocupe que você terá um capítulo só de **Stack** para ler.

15 I/O ou **15 pinos** que você pode configurar como entrada ou saída a seu bel prazer, se você conhecer bem os registradores **TRISA** e **TRISB** e, tenha certeza, você conhecerá.

1 pino só de entrada.

Timer contador com 8 bits.

Timer contador com 16 bits.

Timer contador com 16 bits.

Teoricamente quanto maior o número de bits de um **timer** maior a contagem que você poderá fazer.

Uma aplicação **PWM** que permite captura e amostragem. Veja o pino **CCP1**.
Só para lembrar PWM quer dizer modulação por largura de pulso, com este recurso você conseguirá variar o brilho de uma lâmpada ou até fazer uma fonte chaveada (*leiam o meu livro de fontes chaveadas publicado pela Antenna*).

Um Timer de 8 Bits.

Uma **USART** serial, mas o que é isto?

É um recurso que permitirá que você converse com outros equipamentos, como uma porta serial de um microcomputador.
USART significa "*Universal Synchronous Asynchrous Receiver transmiter ou Transmissor*" Universal Síncrono ou Assíncrono.

2 comparadores analógicos com referência interna, programável, de tensão. Quem gosta de amplificadores operacionais prepare-se para se divertir, que não as conhece vamos ler o apêndice sobre "*AOs*".

O famoso **Watch dog**.

10 possibilidades de interrupção.

Um set de **35 instruções**, ou seja, um grupo de 35 instruções. Com elas você trabalhará com os PICs citados até agora.

Cada pino de I/O com capacidade de fornecer ou consumir **25mA**.

Vamos agora estudar este PIC pino a pino.

Pino 1 – Ra2 – port A, bit 2 – I/O:
- AN2 – entrada analógica 2 para os comparadores;
- Vref – saída de tensão de referência (muito útil quando trabalhamos com AOS).

Pino 2 – Ra2 – port A, bit 3 – I/O:
- AN3 – entrada analógica 3 para os comparadores;
- CMP1 – saída do comparador 1.

Pino 3 – Ra4 – port A, bit 4 I/O:
- CMP2 – saída do comparador 2;

- TOCKI – entrada do comparador TMR0 (vulgo timer0).

Pino 4 – Ra5 – port A, bit 5 – I/O:
- MCLR – Reset ou Mosfet Clear externo. Caso você deixe este pino no terra ele não funcionará, portanto ele deve sempre estar em Vcc.

- Vpp – Esta é a entrada para a tensão de programação que deve ser de 13Vcc sem ripple.

Pino 5 – GND ou terra, também pode ser chamado de "*Vss*".

Pino 6 – Rb0 – port B, bit 0 – I/O:
- INT – entrada para interrupção.

Pino 7 – Rb1 – port B, bit 1 – I/O:
- RX – pino de recepção da comunicação serial assíncrona (USART);
- DT – pino de dados para comunicação serial síncrona (USART).

Pino 8 – Rb2 – port 2 – I/O:
- TX – pino para comunicação, transmissão, assíncrona (USART);
- CK – clock que permite a comunicação serial síncrona (USART).

Sem saber nada de comunicações serial, mas apenas por observação, podemos perceber que:

- a comunicação serial precisa de dois pinos;
- ela pode ser síncrona (*de sincronizado, transmissor e receptor fazendo as coisas ao mesmo tempo ou em sincronismo*);
- ela pode ser assíncrona ("*me perdoem os experts*" *sem sincronismo entre o Tx e o Rx*).

Se for **síncrona**, o **pino 7** será **DT** e transmitirá e receberá dados, e o **pino 8** será o **CK** que manterá o sincronismo entre **Tx** e **Rx**.

Se for **assíncrona**, o **pino 7** será o **Rx** que receberá os dados e o **pino 8** será o **Tx** que enviará os dados.

Pino 9 – Rb3 – port B, bit 3 – I/O:
- CCP1 – pino para o capture, compare e PWM (não tem nada a ver com a extinta União Soviética).

Pino 10 – Rb4 – port B, bit 4 – I/O:
- PGM – usado para programação com baixa tensão que corresponde a 5 Vcc.

Pino 11 – Rb5 – port B, bit 5 – I/O:
- é só.

Pino 12 – Rb6 – port B, bit 6 – I/O:
- T1050 – saída para xtal ou cristal externo;
- T1CK/ - entrada do contador TMR1 (vulgo timer1);
- PGC – usado para programação serial (ICSF). Usado como clock.

Pino 13 – Rb7 – port 7, bit 7 – I/O:
- T1OSI – entrada para xtal ou cristal;
- PGD – entrada de dados da programação serial (ICSP).

Pino 14 – Vcc – alimentação positiva e sem ripple (*na dúvida coloque um capacitor de 100nF x 250Vcc no seu circuito*).

Pino 15 – Ra6 – port A, bit 6 – I/O:
- OSC2 – saída para oscilador a cristal;
- CLKOUT – teremos também neste pino a possibilidade de ler, usar ou, retirar a freqüência de sinal, que entra em OSC1, corresponde aos ciclos de máquina interno (*veremos estes ciclos de máquina um pouquinho mais adiante*).

Pino 16 – Ra7 – port A, bit 7 – I/O:
OSC1 – entrada do oscilador onde, normalmente eu aconselho se ligar um dos pinos do cristal a outro terminal do cristal, vai no pino 15.
- CLKIN – aqui pode ser ligado um oscilador externo desde um formado por uma constante RC até um VCO ou coisa parecida.

Pino 17 – Ra0 – port A, bit 0 – I/O:
- AN0 – entrada analógica zero para os comparadores.

Pino 18 – Ra1 – port A, bit 1 – I/O:
- AN1 – entrada analógica 1 para os comparadores.

Podemos perceber que cada pino tem mais de uma função, então de acordo com o nosso software e de como "*setamos*" ou ajustamos os registradores, um pino pode fazer uma função ou outra. Fique atento a isto, veja um exemplo:
O pino 18 pode tanto ser um **I/O** da porta, como a entrada para um dos comparadores.

Vamos analisar o nome de dois **PICs**, o **16F628-04** e o **16F628-20**.

Capítulo 7
TIPOS DE OSCILADORES

Um microcontrolador precisa de um sinal de clock para funcionar e o responsável por gerar este sinal e o oscilador.
Nos **PICs 16F84** e **16F628** os pinos que correspondem ao oscilador são os **pinos 15** e **16**.
O **pino 15** é normalmente a saída e o **pino 16** é a entrada. Mas, antes de continuarmos com isto, eu pergunto:
O que vem a ser **Clock**?
Na prática um **clock** é um sinal de onda quadrada que serve para fazer um circuito lógico funcionar ou para sincronizar diversos dispositivos ou circuitos.

Quando você vê a configuração de um microcomputador e ele tem um processador.

P4 de 2,8GHz

Quer dizer que o **clock** do processador tem uma freqüência de **2,8 GHertz** ou **2.800 MHz**.

O nível alto do clock deve corresponder à tensão de alimentação do **PIC** (*+Vcc*) e o nível baixo deve corresponder ao terra.
O período alto (*+Vcc*) é chamado de "*Ton*" e o período baixo de "*Toff*".
É interessante que estes dois períodos sejam iguais. A soma dos dois períodos resulta em "*T*" que é o período total da onda quadrada e pelo qual podemos definir qual é a freqüência ou vice-versa.

$T = Ton + ToFF$

$T = \frac{1}{F}$ ou $F = \frac{1}{T}$

Ao usarmos um cristal de 4MHz teremos uma freqüência de 4MHz e o período de:

$$T = \frac{1}{F} = \frac{1}{4.000,000} = 250ns \text{ ou } 250 \text{ nano segundos}$$

Ou **0,000. 000.25** segundos.

Agora que já conhecemos o **clock**, vamos falar sobre **ciclo de máquina**.

7.1. Ciclo de Máquina:

O microcontrolador PIC pega o sinal de clock e o **divide internamente por 4**. Disto resultam quatro períodos que receberão o nome de: "*Q1*, *Q2*, *Q3* e *Q4*", cada período destes "*Qs*" será igual ao período do clock externo, ou seja, **250ns**.
Podemos também dizer que o clock de 4MHz é igual à freqüência de 1 MHz.
Veja:

$$\text{Clock interno} = \frac{\text{Clock externo}}{4} = \frac{4MHz}{4} = 1MHz$$

O período do clock interno será:

$$T = \frac{1}{F} = \frac{1}{1MHz}$$

T = 1µs ou 1 micro segundo ou 0,000. 001 segundos.

Para executarmos uma instrução, o PIC precisa passar por **Q1**, **Q2**, **Q3** e **Q4**, portanto, uma instrução demora para ser executada, estes **4 tempos somados**:

Q1 + Q2 + Q3 + Q4 = 250ns + 250ns + 250ns + 250ns = 1µs

Vemos, então que cada instrução demora **1µs** para ser executada.
A forma mais fácil de sabermos o tempo que uma instrução leva para ser executada é pegar a **freqüência do cristal** ou oscilador externo, **dividir por 4** e calcular seu período.

$$\text{Freqüência da instrução} = \frac{\text{freqüência do xtal}}{4}$$

$$\text{Freqüência da instrução} = \frac{4MHz}{4} = 1MHz$$

$$\text{Tempo ou período para executar a instrução} = \frac{1}{1MHz} = 1\mu s$$

O PIC usa um recurso chamado de "*Pipeline*", que faz com que em um ciclo máquina ele busque a instrução e em outro ele a execute.
Resumindo, o **ciclo de máquina** é o nome das **4 fases Q1**, **Q2**, **Q3** e **Q4**, e cada instrução é executada em um ciclo, pois, enquanto uma é executada outra ao mesmo tempo é buscada na memória.
Isto é uma característica da estrutura **Harvard** que permite o **pipeline** de uma forma fácil.

Você pode estar pensando que todas as instruções, usando um **cristal de 4MHz**, demorarão **1µs** para serem executados, mas, instruções que geram "*saltos*" dentro do programa gostam mais de **1µs**, pois precisam de **dois ciclos de máquina**. Gastarão, então, **2µs**.

Você também pode estar pensando que **1µs** é um "*tempo muito pequeno*", mas, se você for construir, por exemplo, um gerador de barras para **monitor** ou **TV**, com um ciclo de máquina de **1µs** poderá ter problemas. O que fazer então? Usar um **PIC 16F628-20**, por exemplo. Com um **cristal de 20 MHz** teremos o período de execução de cada instrução.

$$\text{Freqüência da instrução} = \frac{20\text{MHz}}{4} = 5\text{MHz}$$

$$\text{Período para executar a instrução} = \frac{1}{5\text{MHz}} = 200\text{ns}$$

ou 200 nano segundos ou 0,000. 000.2 segundos.

Desta forma ficou muito mais rápido. Você não acha?
Agora que já vimos osciladores, clocks e ciclos de máquinas vamos ver os tipos de osciladores que estes PICs aceitam.

XT – cristal ou ressonador com freqüência maior do que 200 kHz e que vá até 4MHz.

HS – cristal ou ressonador com freqüência acima de 4Mhz. Neste caso a freqüência máxima será definida pelo PIC.

LS – cristal ou ressonador com freqüência abaixo de 200 khz. Pausa: você sabia que existem cristais que oscilam em 15 khz?

RC_CLKOUT – oscilador RC externo que deve estar ligado no pino 16. No pino 15 teremos este sinal dividido por 4.

RC_I/O – Oscilador externo mais que usa o pino 15 como I/O. Neste caso não teremos no pino 15 Fo ÷ 4, pois ou ele faz uma coisa ou outra. Isto só é válido para o PIC 16F28 (**relativo ao PICs que estamos estudando neste livro*).

INTOSC_I/O – oscilador interno com o pino 15 operando como I/O (**só é válido para o 16F28*).

EC_I/O – usado com clock externo e pino 15 funcionando como I/O (**válido para 16F628*).

RC – usando um circuito RC externo (# válido *para 16F84*).

LP – usando cristal de freqüência abaixo de 200 kHz (# válido para 16F84).

Estes tipos de osciladores recebem estes nomes, pois, é com eles que vamos trabalhar ao fazermos nosso software ou ao configurarmos o MPLAB para a gravação. Observe que algumas opções servem para os dois PIC estudados e outra apenas para alguns deles. Para facilitar vemos:

*** = válido para PIC 16F628 no nosso estudo.**
 # = válido para PIC 16F84 no nosso estudo.

Reescrever:

PIC 16F84 – osciladores: XT

	LP
	HS
	RC
PIC 16F628 – osciladores:	XT
	LS
	HS
	RC_CLKOUT
	RC_I/O
	INTOSC_I/O
	INTOSC_CLKOUT
	EC_I/O

Normalmente eu uso na prática ou no hardware, como queiram, cristais de **4 MHz** com capacitores de **15pF**. Caso você use capacitores acima de **33pF** com um cristal de **4 MHz** o oscilador poderá não funcionar adequadamente (o valor da XC será muito baixo e atenuará muito as oscilações do cristal).
Com cristais de **20 MHz** eu uso capacitores de **15pF** e nunca tive problemas. Na hora de comprar os capacitores opte por capacitores de *disco cerâmicos* ou *capacitores plate*.

Não vá fazer um software e um hardware que use um **cristal de 20 MHz** e na hora de gravar o PIC usar **opção XT** que é para cristais de até 4 MHz. Se você fizer isto, não aparecerá erro na gravação, mas, o seu circuito hora funciona, hora não. Eu sei disto na prática pois também erro.

Nunca confunda a hardware com o software, quando falamos no componente **cristal de 4 MHz** ou de **20 MHz** no **capacitor cerâmico** ou **plate**, estamos nos referindo ao hardware, a placa de circuito, ao cheiro da solda, a bancada.

Quando nos referimos a XT, LP, RC_I/O, entre outros, estamos nos referindo ao software. Mas, especificamente ao comando ou diretriz que devemos incluir no software para que o projeto funcione.

Na prática, se usarmos no nosso hardware, um componente chamado de cristal, com freqüência de 4 MHz, feito de cristal, envolto por metal e com dois terminais, devemos usar XT no nosso software.

Dica:
Fazer experiências com uma placa padrão já pronta é muito mais fácil que desenvolver o software e a hardware para uma determinada aplicação. Estudar eletrônica é muito importante e, como disse anteriormente, ler bons livros de eletrônica, é realmente muito importante em nossa área. Ao menos leiam todos os apêndices existentes neste livro.

Capítulo 8
O WATCH DOG

Como eu já havia dito anteriormente neste livro, o **watch dog** é um timer ou temporizador independente do clock do PIC ou de qualquer outro componente externo.
Ele sempre esta contando e o seu tempo total de contagem é de, aproximadamente, **18ms**.
Este tempo pode variar de acordo com a temperatura e flutuação na alimentação (*olha o ripple novamente*). Mas, o importante é saber que quando o tempo de contagem excede **18ms**, ele estoura (*daí você vê uma pequena fumaça subindo da bancada*), ou melhor, dizendo, ocorre um *overflow* e ele *reseta* o PIC e assim o programa começa tudo de novo.

A função dele é evitar que algum travamento no programa, causado por hardware ou software, seja resolvido depois de um **reset** do microcontrolador.
Imagine um **dimmer** (*controlador de intensidade luminosa*) constituído com um PIC. Em determinado momento um pico de energia faz com que a luz, que estava bem fraquinha, fique com o seu brilho máximo devido há um "*travamento*" do PIC. O **watch dog** "*reseta*" ele e o brilho volta ao normal se o seu programa foi feito para isto.
Isto apenas é um exemplo, imagine a importância do **watch dog** em sistemas ligados à segurança.
Mas, meus programas vão ficar limitados a rodarem em **18ms**? É claro que não. Basta você usar a instrução CLRWDT, e o registrador WDT, que faz a contagem da **watch dog**, será **resetado** e não acontecerá o **overflow** nem o **reset** do PIC. Mas, se o seu programa travar, ele não passará por esta instrução e o **overflow** acontecerá.

A instrução CLRWDT é extremamente útil quando precisamos criar sub-rotinas de tempo, traduzindo para o português correto, ela é importante se pretendermos fazer um timer com o PIC.
O **watch dog** pode ser ligado ou desligado, ou melhor, dizendo, pode ser ativado ou desativado em apenas duas condições:

- com uma linha de comando no cabeçalho do programa;
- na hora da gravação do PIC.

Se você usar a instrução **sleep** (*acorda o meu filho ou filha*) e se acontecer um estouro, o microcontrolador retornará na instrução seguinte ao **sleep**.

Você pode associar o **watch dog** ao **prescaler**, que é um divisor ajustável, e aumentar o período dele para até **2,2 segundos** aproximadamente (*não tenho cronômetro*).

Vamos estudar o **prescaler** daqui a pouco. Agora vamos salientar que apenas duas instruções zeram o watch dog e fazem com que ele recomece a contar.

As funções são:

SLEEP $\Rightarrow$ zera o watch dog e coloca o PIC em modo econômico.

CLRWDT $\Rightarrow$ zera o watch dog evitando o overflow ou estouro e o reset do PIC.

Capítulo 9
O PRESCALER

Antes de tudo o **prescaler** é um divisor. No caso dos PICs é um divisor que pode ter o seu fator de divisão ajustado.

O **prescaler** que é um divisor pode ser atribuído, ou seja, pode estar conectado ao **TMR0** (*timer 0*) ou ao **watch dog**. Para definirmos isto precisamos atribuir um valor ao **bit 3** ou **flag 3** do registrador **Option**.

Se o valor do **bit 3** for **1** a **prescaler** estará ligado e dividindo a contagem do **watch dog**.
Se for **0** o **prescaler** será atribuído ao **TMR0**.

Podemos perceber a importância dos registradores, vamos estudá-los mais profundamente em um próximo capítulo. Tudo ao seu tempo.

O **bit 0,** o **bit 1** e o **bit 2** do registrador **Option** definem a taxa de divisão do **prescaler**. Já aviso que esta taxa é diferente para a **watch dog** e para o **TMR0**.

Estes bits tem nomes próprios, vamos vê-los:

O bit 3 recebe o nome de PSA.
O bit 0 tem o nome de PS0.
O bit 1 tem o nome de PS1.
O bit 2 tem o nome de PS2.

Você pode ler ou escrever o valor nestes **flags** do registrador **Option**.

Podemos perceber que de acordo com os valores nas posições PS2, PS1 e PS0, teremos um fator de divisão.
Também podemos perceber que a máxima divisão do **Timer0** será por **256**, e a máxima divisão do **watch dog** será por **128**.
Nas nossas experiências iremos "*ver*" estas divisões na prática.

Capítulo 10
STACK

O **stack** consiste em uma pilha com **oito posições** diferentes. Também podemos dizer que ele tem oito níveis diferentes. O **stack** nesta família de **PICs** não é acessível ao programador.
Sua função é armazenar a posição em que o programa parou ou foi desviado, para executar uma sub-rotina e fazer o programa voltar para a posição imediatamente seguinte, após realizar a sub-rotina.
Ele trabalha junto com o **PC** que é o **programa Counter** ou **Contador de Programa**.
Basicamente falando o **PC** conta as linhas do programa que estão sendo executadas e seu funcionamento é praticamente transparente ao usuário ou ao programador.
Toda vez que uma instrução **Call** é usada, o **PC** armazena o valor **PC+1** na **Stack**, isto para saber em qual linha do programa deve voltar.
A mesma coisa acontece quando usamos interrupções.
Como temos apenas **8 níveis**, não podemos ter mais de **8 instruções** de desvio acontecendo ao mesmo tempo, pois as chamadas acima da oitava serão armazenadas sobre as outras e aí o programa não saberá para onde ir.
Os **oitos níveis no Stack** são montados da seguinte forma:

- nível 8 $\Rightarrow$ 8ª chamada call.
- nível 7 $\Rightarrow$ 7ª chamada call.
- nível 6 $\Rightarrow$ 6ª chamada call.
- nível 5 $\Rightarrow$ 5ª chamada call.
- nível 4 $\Rightarrow$ 4ª chamada call.
- nível 3 $\Rightarrow$ 3ª chamada call.
- nível 2 $\Rightarrow$ 2ª chamada call.
- nível 1 $\Rightarrow$ 1ª chamada call.

- terminada a 8ª chamada este espaço ficará vago.
- terminada a 7ª chamada este espaço ficará vago e assim, sucessivamente.

A pilha é montada de baixo para cima e desmontada de cima para baixo.

Como na série 16 dos microcontroladores PIC não podemos ver o estado da pilha, é muito importante prestar atenção em quantas chamadas se está usando. Caso o **Stack** esteja cheio e uma chamada seja feita, o **nível 1** já era, ou melhor, o endereço que estava no **nível 1** já era, e o seu programa também.
O **PC** não voltará para o lugar correto ao chegar ao **nível 1**.
As únicas instruções que tem acesso à pilha são **Call**, **Return**, **Retlw** e **Retfie** além da interrupção.
Um **Call** e uma interrupção guardam endereços de retorno no **Stack**.
A instrução "*GOTO*" não armazena endereço no **Stack**.
Se você for usar interrupções em seu programa, considere a pilha com apenas **7 níveis**, pois se não fizer isto e acontecer uma interrupção com os **8 níveis** ocupados, haverá um estouro da pilha e o seu programa não funcionará.
Usou interrupção, deixe um nível vazio no **Stack**.

Capítulo 11
O PC

O **PC**, sigla para "***Program Counter***", ou contador de programa e é o responsável pela seqüência exata da execução das instruções dos programas. Quando temos uma **instrução Call** ou uma interrupção, ele "*fica mais importante*". Veja no exemplo a seguir:

1 – instrução 1;
2 – instrução 2;
3 – instrução 3;
4 – Call **XX** -> instrução 4 $\Rightarrow$ nesta hora o valor **PC+1** é armazenado no **Stack**, ou seja, a instrução **4 + 1**, que é igual a **5** será armazenada no **Stack** e quando terminar a sub-rotina, chamada de **XX,** a instrução **return** saberá, graças ao **PC** e ao **Stack** que deverá voltar para a **posição 5**.
5 – instrução 5;
6 – instrução 6;
7 – instrução 7.

Capítulo 12
O REGISTRADOR W

O **registrador W** é extremamente útil no PIC, pois é através dele que fazemos diversas partes de um programa.

O nome **registrador W** vem de "*Registrador Work*", que traduzindo, quer dizer trabalho.

É através dele que carregamos os outros registradores com valores diversos e corretoss para um perfeito funcionamento de nosso projeto.

Caso desejamos colocar um determinado valor em um registrador de uso geral (*GPR*), primeiro temos que "*carregar*" o **registrador W** e depois passar para o outro registrador.

Para carregarmos o registrador W usamos as instruções:

MOVWF e MOVLW.

Capítulo 13
CATEGORIA DE REGISTRADORES

Os microcontroladores PICs possuem dois tipos de registradores, além dos registradores W, estes registradores são chamados de "*GPR e SFR*", e normalmente, são indicados pela **letra f** (*minúsculo*).

Ao contrário do registrador W, eles estão implementados dentro da **memória RAM**. As siglas significam o seguinte:

GPR = General Purpose Register = Registrador de Propósito Geral (*mais fácil -> Registrador de Uso Geral*).

SFR = Special Function Register = Registrador para Funções Especiais.

f = file register = Registrador de Arquivo.

Normalmente se usa "*f*" tanto para identificar registros "*SFR*" como "*GPR*".

Um registro é um endereço de memória, que pode receber um nome.
Como os microcontroladores usados são de **8 bits** o registro pode ter um valor entre **0** a **255** em decimal ($2^8 = 255$) ou de **0000 0000** a **1111 1111** em binário ou de **0** a **FFH** em hexadecimal.

No caso do **PIC 16F84** ou do **16F628**, este registro tem que estar em uma posição da memória entre **0** a **127** em decimal ou **0000 0000** a **0111 1111** em binário ou **0** à **7FH** em hexadecimal.

Temos um espaço na **memória RAM**, de uso geral, que vai de **12** a **79** em decimal ou **0CH** a **4FH** em hexadecimal o que nos deixa um espaço de **68 bytes** para o **PIC 16F84** no **banco 0** espelhado no **banco 1**.

Temos um espaço de **memória RAM**, de uso geral, que vai de **32** a **128** no **banco 0** o que dá **96 bytes**, temos um espaço de **160** a **240** no **banco 1** o que dá **80 bytes** e de **288** a **335** no **banco 2**, o que dá **47 bytes** para o **PIC 16F628**. Temos assim o valor de 96 + 80 + 47 bytes disponíveis no **PIC 16F628**.

Importante: Um registrador é um endereço na memória RAM.

O ideal é darmos um nome a este registrador. Todo registrador pode assumir um valor entre **0** a **255** em decimal.
Também é importante perceber que podemos definir um endereço ou uma variável em decimal, binário ou hexadecimal, exemplo:

128 em decimal é igual a **10000000**, em binário que é igual às **80H**.

O mais comum para nós é o decimal, porém quando desejamos alterar ou ajustar o valor de uma ou mais **flags** em uma registrador de **8 bits** o mais fácil é usar o binário.

Em binário fica mais fácil visualizarmos o estado de cada **bit**, por exemplo, você saberia como estariam os bits se coloca-se
no "*INTCON*" o **valor 154** em decimal?

Podemos escrever nosso programa usando como base numérica o decimal, o octal, o binário ou o hexadecimal. Mas, precisamos avisar o microcontrolador em qual base vamos trabalhar.

O número máximo de registradores de um microcontrolador irá depender do tipo de microcontrolador.

Cabe lembrar que teremos os **registradores SFR** que definirão as características de funcionamento do microcontrolador (*como exemplo podemos citar o INTCON, o TRISA, o STATUS*) e os **GPR** que são registradores de uso geral e normalmente "*criados*" pelo programador. No caso do PIC 16F84 estes registradores de uso geral se limitam a 15.

Capítulo 14
DECIMAL / HEXADECIMAL / BINÁRIO

No capítulo anterior vimos duas coisas raras e importantes. Uma delas é a base numérica e a outra são os bancos de memória.

Neste capítulo estudaremos as bases numéricas. No próximo capítulo estudaremos os bancos da praça. Melhor dizendo, os bancos de memória.

Vamos estudar algumas conversões (*à direita e a esquerda, sempre com o uso da seta*).
O resultado em decimal é a somatória deles todos.

1F9H = 9 + (15 x 16) + (1 x 256)
1F9H = 9 + 240 + 256
1F9H = 505 em decimal.

Mas, e para transformarmos decimal em hexadecimal como faremos? Existem algumas formas, se você tiver uma calculadora científica, com esta função, por exemplo, é bico. Mas, deixando a graça de lado, vamos ver uma forma simples:

1º) Pegue o valor em decimal e divida por 16.
2º) Transforme a parte inteira do resultado em hexadecimal.
3º) Pegue a parte fracionada e multiplique por 16.
4º) Junte as duas partes e tá aí o resultado em hexadecimal.

Exemplos:

O valor 254 em decimal corresponde a quanto em hexadecimal?

1º) 254 / 16 = 15,875

2º) 15 = F

3º) (16 x 0,875) = 14 = E

4º) 254 decimal = FEH

O valor 255 em decimal corresponde a quanto em hexadecimal?

1º) 255 / 16 = 15,9375

2º) 15 = F

3º) (16 x 0,9375) = 15 = F ⇒ junte os dois, F com F

4º) 255 em decimal = FFH

Quanto vale 18 em hexa?

1º) 18 / 16 =1,125

2º) 1 = 1

3º) (16 x 0,125) = 2 ⇒ junte os dois, 1 com 2

4º) 18 = 12H

Caso o resultado do lado esquerdo da vírgula (*parte interna*) seja maior do que 16 será necessário dividir esta parte por 16 novamente.

Veja os exemplos a seguir:

300 / 16 = 18,75 ⇒ (16 x 0,75) = 12 = C

300 / 16 = 1,125 ⇒ (16 x 0,125) = 2

1

Juntando os três temos 12CH.
Portanto 300 em decimal corresponde a 12 CH em hexadecimal.

Exemplo:

505 / 16 = 31,5625 ⇒ (16 x 0,5625) = 9

31 / 16 = 1,9375 ⇒ (16 x 0,9375) = 15 = F

1

Juntando as três partes teremos o número 1F9H.
Portanto 505 em decimal corresponde a 1F9 em hexadecimal.

Exemplo:

1000 / 16 = 62,5 ⇒ (16 x 0,5) = 8

62 / 16 = 3,875 ⇒ (16 x 0,875) = 14 = E

3

Juntando os três números na duração indicada pela seta teremos o número 3E8 em hexadecimal ou 3E8H que corresponde a 1000 em decimal.

Exemplo:

4096 / 16 = 256,0 $\Rightarrow$ (16 x 0) = 0

256 / 16 = 16,0 $\Rightarrow$ (16 x 0) = 0

16 / 16 = 1,0 $\Rightarrow$ (16 x 0) = 0

1

4096 em decimal é igual a 1000H em hexadecimal. Veja por este exemplo, que quanto o resultado é apenas inteiro (sem parte após a vírgula) o número 16 deve ser multiplicado por 0.

Exemplo:

4095 / 16 = 255,9375 $\Rightarrow$ (16 x 0,9375) = 15 = F

255 / 16 = 15,9375 $\Rightarrow$ (16 x 0,9375) = 15 = F

15 = F

4095 em decimal é igual à FFFH.

Agora chega de exemplos, a finalidade de tanto é fazer você perceber uma lógica entre estas conversões e ver que com uma calculadora comum tudo isto é muito fácil.

Agora vamos ver como passar de binário para decimal e vice-versa. Vimos que em decimal a base é **10** em hexadecimal a base é **16** e em binário a base será **2**.

No cabeçalho de um programa com um PIC teremos uma diretriz que indicará com que base o programa trabalhará, esta diretriz recebe o nome de radix.

Uma forma simples de perceber a relação entre decimal e binário é através do uso de uma tabela, que pode receber o nome da tabela da verdade.

Como a base do binário é **2** só teremos dois dígitos, o **zero** (*0*) e o **um** (*1*) e todos os números serão representados por um conjunto de "*zeros*" e "*uns*

Vamos fazer a tabela da verdade e falar um pouco sobre ela. Mas, antes vamos perceber o seguinte:

$2^0 = 1 \Rightarrow$ dois elevado a zero é igual a um;
$2^1 = 2 \Rightarrow$ dois elevado a um é igual a dois;
$2^2 = 4 \Rightarrow$ dois elevado a dois é igual a quatro;
$2^3 = 8 \Rightarrow$ dois elevado a três é igual a oito;
$2^4 = 16 \Rightarrow$ dois elevado a quatro é igual a dezesseis;
$2^5 = 32 \Rightarrow$ dois elevado a cinco é igual a trinta e dois;

Usando esta tabela, podemos transformar qualquer número entre **0** a **16** em decimal em binário, temos que perceber o seguinte: para encontrarmos o valor em decimal de um número em binário, devemos elevar a **base 2** a potencia correspondente e multiplicar por **1** ou **0** dependendo da posição do número e/ou valor em binário.

Exemplo:

100100 é igual a quanto em decimal.

O resultado é igual a 32 + 4 = 36 em decimal. Mas, no caso dos registradores dos PICs estudados poderemos ter um número de 8 bits ou 1 byte, como calcular:

O resultado é: 128 + 64 + 8 + 4 = 204 em decimal.

Podemos perceber que todo número é múltiplo de dois. Também é bom lembrar que qualquer número elevado à **zero** é igual a **1**.

O resultado é **igual** a **145** em decimal.
Podemos perceber que não é necessário se multiplicar os números por **0** (*zero*).
É importante saber qual o resultado do valor de **2 elevado** a "*x*" (2^x), onde "*x*" é um número que corresponde à posição do digito. Lembre-se de que o primeiro "*x*" será "*0*".
Se quisermos colocar em uma porta o **valor 32** em decimal, basta carregar na mesma o **valor 10000** em binário, ou melhor, como a porta pode ter **8 bits** (*lembre-se em consideração a port B*), o número seria assim:

00010000

Para convertermos um número decimal em binário precisamos dividir este número por 2. Devemos fazer isto até o resultado ser fracionário. Enquanto isto, não acontece, devemos pressupor que a parte fracionária é "*0*" e **multiplicar 2** por **0**. Quando houver uma parte fracionária, devemos **multiplicar** por **2**, veja:

Exemplo:

8 em binário é igual a:

8 / 2 = 4 $\Rightarrow$ (2 x 0) = 0

4 / 2 = 2 $\Rightarrow$ (2 x 0) = 0

2 / 2 = 1 $\Rightarrow$ (2 x 0) = 0

1

8 em decimal = 1000 em binário.

Exemplo:

17 / 2 = 8,5 $\Rightarrow$ (2 x 0,5) = 1

8 / 2 = 4 $\Rightarrow$ (2 x 0) = 0

2 / 2 = 2 $\Rightarrow$ (2 x 0) = 0

2 / 2 = 1 $\Rightarrow$ (2 x 0) = 0

1

10001 em binário = 17 em decimal.

Exemplo:

255 / 2 = 127,5 ⇒ (2 x 0,5) = 1

127 / 2 = 63,5 ⇒ (2 x 0,5) = 1

63 / 2 = 31,5 ⇒ (2 x 0,5) = 1

31 / 2 = 15,5 ⇒ (2 x 0,5) = 1

35 / 2 = 7,5 ⇒ (2 x 0,5) = 1

7 / 2 = 3,5 ⇒ (2 x 0,5) = 1

3 / 2 = 1,5 ⇒ (2 x 0,5) = 1

1

1111111 em binário = 255 em decimal

Quando a parte inteira da divisão é igual a **1** (*ou menor do que 2*) terminaram as divisões e este **1** fará parte do conjunto.

Resumo:

- decimal = base decimal composta de **10 dígitos** de **0** a **9**.

- hexadecimal = base hexadecimal composta de **16 dígitos** de **0** a **F**.

- binário = base binária composta por **2 dígitos** de **0** a **1**.

Capítulo 15
BANCOS DE MEMÓRIA DE DADOS E CONTROLE DO PIC 16F84

Será que isto é um velho banco de praça, onde um velho homem se senta para lembrar o passado? Acho que não.

Chamamos de banco de memória de controle, um espaço na memória RAM que é reservado para os registros para funções especiais, como o **Option**, **Status**, **Intcon**, **Trisa**, **Trisb** e etc. Este espaço recebe o nome de "*memória de controle*", pois os registros que ocupam estas posições controlam o funcionamento e a comunicação do PIC.

Junto com o banco de memória de controle há um espaço de memória de dados. É neste espaço que colocaremos as nossas "*variáveis*".

No **PIC 16F84** existem **2 bancos** de memória e no **PIC 16F28** existem **4 bancos**. Muitos registradores especiais estão presentes em mais de um banco. Isto pode parecer estranho, se repetir uma mesma informação (*um registrador guarda uma informação*) em dois ou mais endereços diferentes, mas, isto pode ajudar na programação.

Podemos perceber que existem registradores que estão no **banco 0**, registradores que estão no **banco 1** e registradores que estão nos dois bancos. Muitas vezes precisamos, em um programa,

mudar de banco para pegar uma informação que está em outro banco. O banco onde normalmente o "*PIC deve estar*" quando um programa é rodado é o **banco 0**.
A **memória RAM** usada para se gravar variáveis, vai do **endereço 12** em decimal até **79** em decimal o que dá um espaço de **68 bits**. Mas, e o espaço do **banco 1**?
Como está escrito este espaço é um espelho do espaço correspondente no **banco 0**, ou seja, uma informação caso estiver na **posição 12** em decimal do **banco 0**, também estará na **posição 140** em decimal do **banco 1**.

Indicamos os endereços da memória em **hexadecimal** (*xxH*), em **decimal** (*xxd*) e em **binário** (*xxb*) para que você vá se acostumando com estas três formas de numeração que são comuns em circuitos digitais micro-controlados.

É importante lembrar, que esta memória guarda dados de **8 bits** por endereço ou **1 byte** por endereço.

Também é importante lembrar, que ela está dividida em bancos devido à construção interna do PIC.

Todos os valores nestes bancos estão armazenados em **memória RAM**, ou seja, se a alimentação for cortada, estes dados se perderão.
Para mudar de um banco para outro, usaremos f**lags** em registradores especiais, convém lembrar que algumas pessoas podem chamar estes **flags** de "*chaves*".

Capítulo 16
MEMÓRIA DE PROGRAMA DO PIC 16F84

A memória de programa pode ter **12,14** ou **16 bits**, dependendo do PIC. Nesta memória é que escreveremos os programas.
No **PIC 16F84**, ela tem **14 bits**, ou seja, cada endereço da memória de programa pode ter até **14 bits**. A esses **14 bits** damos o nome de "*Word*" (*Palavra*).
Com 14 bits o número máximo de bits será igual há: $\mathbf{2^{14} = 16.384}$ bits, ou seja, teremos 16.384 combinações diferentes de bits.
A memória de programa, normalmente em PICs reagraváveis, é FLASH e não perde as informações quando a alimentação é cortada. Caso contrário perderíamos o programa cada vez que desligássemos o nosso circuito.

Caso você ache que já leu isto em alguma outra parte deste livro, não ache, tenha certeza. Vamos repetir para decorar, mas, sem stress, apenas de uma forma natural.
É bom lembrar que o "*F*" do **PIC 16F84** quer dizer que na memória de programa deste PIC é **Flash** ou **EE-FLASH**.

Capítulo 17
BANCOS DE MEMÓRIAS DE DADOS E CONTROLE DO PIC 16F628

Neste PIC teremos **4 bancos** e memórias ao invés dos **2 bancos** do **PIC 16F84**.

Teremos nele os bancos:
- banco 0.
- banco 1.
- banco 2.

- banco 3.

Caso você já não tenha percebido, em digital, tudo começa com **0** (*zero*).

Chamamos de banco de memória de controle um espaço na memória RAM que é reservado para os registradores para as funções especiais, como o Option, Status, Intcon, Trisa, Trisb, Cmcon, Vrcon, etc. Este PIC possui mais registradores que o 16F84. Isto se deve ao fato de que ele possui mais recursos que a 16F84. Temos dentro dele dois comparadores que podem ser utilizados em diversas configurações e permite associar uma variação analógica de uma tensão ao processamento digital feito pelo PIC.
No 16F628 existem 4 bancos de memória, e que estes bancos guardam informações de 8 bits, ou um byte, em cada posição.

Olhando os 4 bancos de memória (*do banco 0 ao banco 3*) podemos perceber que existem registradores que estão em um banco, outras que estão mais de um banco. Devido à posição dos registradores nos bancos, durante um programa, às vezes, precisamos mudar de banco para acessar determinado registrador.
Um PIC normalmente trabalha no banco 0 (zero), caso ele precise ler ou escrever algum bit ou flag, de algum registrador em algum outro banco ele deve ir ler ou escrever e voltar para o banco 0 (zero). Mas, ele não faz isto sozinho. Você que é o programador, é quem faz.
Neste PIC temos 96 bytes de memória no banco 0 (zero), para usarmos com nossas variáveis.
Temos também, mais 80 bytes no banco 1 e 48 bytes no banco 2.
No total teremos 224 bytes de memória para nossas variáveis.
Eu aconselho, pelo menos ao começar a programar, utilizar somente a memória do banco 0 (zero).
Lembre-se que um byte é composto de 8 bits.
A memória é dividida em bancos devido à construção interna do PIC.

Capítulo 18
MEMÓRIA DE PROGRAMA DO PIC 16F628

A memória de programa do PIC 16F628 é de 2K Words ou 2.048 posições de 14 bits. O PIC 16F84 tem uma capacidade de 1K ou 1.024 posições de 14 bits. Podemos perceber que com o 16F628 podemos fazer programas maiores.

Toda esta memória é FLASH ou EE-FLASH.

Programas maiores acontecerão com certeza quando você ler o meu próximo livro sobre PICs. Lá você aprenderá a programar em C e verá que um programa em C, normalmente ocupa mais espaço. Mas, primeiro vamos escovar os bits em Assembler...

Lembre-se que a memória FLASH não perde os dados quando a alimentação é cortada.

Perceba que com o 16F628 você terá 2.048 posições para colocar uma combinação de números em binário, 0 e 1. Perceba também que você terá 2 elevado a 14 combinações diferentes em binário, ou seja, 16.384 possibilidades de 0 e 1 para colocar em 2.048 endereços.

Capítulo 19

UM “GERALZÃO” SOBRE MEMÓRIA DE PROGRAMA

Basicamente memória de programa é o local onde você estará gravando o seu programa. Por programa você pode chamar o conjunto de instruções em Assembler ou seu algoritmo como algumas pessoas gostam de dizer.
Observe que seu programa não consistirá apenas das instruções, mas, também de variáveis, dados e acessos a registradores. Se você não ficar atento, poderá nem perceber isto tudo, mas, é assim que funciona.

Capítulo 20
UMA “GERALZÃO” SOBRE MEMÓRIA DE DADOS

Como já deu para perceber o “geralzão” não é tão grande assim, é apenas uma forma de enfatizar alguns conceitos.

Vamos lá: Na memória de dados você colocará suas variáveis. Estas variáveis normalmente receberão nomes dados por você e ocuparão um determinado endereço. Este recurso facilitará muito a construção de um programa, pois toda vez que você precisar daquele variável é só chamá-la pelo nome.
Na memória de dados também ficam os registradores de controle.

Capítulo 21
REGISTRADORES ESPECIAIS E MAIS UTILIZADOS NO PIC 16F84 E PIC 16F628

21.1. Registrador STATUS:

Este registrador está diretamente associado à unidade lógica Aritmética, ou a já famosa “ULA”. Através dele conseguimos ver o estado da ULA. Gostaria de relembrar que um registrador é um endereço na memória de dados, composta por 8 bits e que cada bit deste recebe o nome carinhoso de flag.

Perceba meu amigo leitor que de 7 até 0 teremos 8 posições, portanto 8 bits.

R/W quer dizer Read e Write - Leitura e escrita que tem suas versões em português para L/E.

Então:

L/E quer dizer leitura e escrita. Um bit indicado por R/W ou L/E permite que você faça a leitura de seu valor ou que escreva um valor nesta posição. Lembre-se que você só poderá escrever 0 (zero) ou 1 (um).

Um bit ou flag indicado por R permite apenas a leitura, mas não a escrita. Com a análise destes bits podemos fazer nossos programas ficarem mais rápidos e eficientes.

Podemos ler ou escrever em um registrador bit a bit ou tudo de uma vez. Em minha opinião a análise separada de cada bit é mais eficaz.

Vamos ver o que cada bit representa:

Bit 7 - IRP –> este flag seleciona o banco de dados usado para endereçamento indireto (calma...).
Se o seu valor for 0 (zero) estaremos usando os bancos 0 (zero) e 1 (um).
Se o seu valor for 1 estaremos usando os bancos 2 e 3.
Como o 16F84 só tem os bancos 0 e 1, devemos manter este bit sempre em 0 (zero).
Permite a leitura e escrita.
Bit 7 = 0 –> bancos 0 e 1.
Bit 7 = 1 –> bancos 2 e 3.

Bit 6 – RP1 –> este tem a função de selecionar os bancos no endereçamento direto.
Podemos perceber que ele trabalha junto com o Bit 5:

Bit 5 – RP0 –> seleciona os bancos no endereçamento direto.

Veja:

RP1	RP0:	
0	0	= banco 0
0	1	= banco 1
1	0	= banco 2
1	1	= banco 3

podemos perceber que temos 4 combinações em binário o que nos permite selecionar 4 bancos. Como no 16F84 só temos dois bancos, o 0 (zero) e o 1 (um), devemos manter RP1 sempre em 0 (zero), desta forma nossa tabela fica assim:

RP1	RP0	
0	0	= banco 0
0	1	= banco 1
0	0	= banco 0
0	1	= banco 1

Só temos duas combinações e podemos acessar os dois bancos do 16F84, o banco 0 e o banco 1.
Estes dois Flags são R/W, ou seja, permitem a escrita ou leitura.

Bit 4 - /T0 ou T0\ –> este bit informa que ocorreu um *time-out*.
Mas, o que é isto?
Isto significa que houve um estouro no tempo de contagem do *Watch-dog* ou que o *Watch-dog* já contou até onde consegue e resetou o PIC. Lembre-se que o *Watch-dog* conta separadamente de tudo e que se você não quer que o tempo de contagem estoure e o seu PIC seja resetado, deve apagar sempre o valor na *Watch-dog*. Para que isto não aconteça, use a instrução ***Clrwdt*** em seu programa, principalmente dentro de Loops.

Seu estado será 0 (zero) quando acorrer um estouro do *Watch-dog*.

Seu estado será 1 quando você ligar o PIC, mandar o PIC fazer a instrução *Clrwdt* ou Sleep.

/T0 = 0 –> estouro de *Watch-dog* seu programa foi resetado e voltará ao começo (muitas vezes, por causa disto, um programa não funcionará).
/T = 1 –> você usou as instruções Sleep, Clrwdt ou ligou o circuito. Ligar o circuito também é chamado de ***Power-up***.
Este flag é R, ou seja, só permite que seja feita sua leitura.

Bit 3 - /PD ou PD\ –> este bit se chama ***Power-down***, mas, não quer dizer que você desligou o PIC. Mas ele serve para ver se você executou uma instrução Sleep ou Clrwdt.

/PD = 1 –> você executou uma instrução Clrwdt.
/PD = 0 –> você mandou o PIC dormir executando uma instrução Sleep.

Este bit só permite a leitura, então ele é R.

Bit 2 – Z –> a função deste flag é sinalizar o 0 (zero). Com este bit podemos simplificar nossos programas e conferir resultados de instruções.

Z = 0 –> mostra que o resultado da última operação matemática ou lógica não foi igual a 0 (zero).
Z = 1 –> demonstra que o resultado da última operação lógica ou aritmética foi igual a 0 (zero).

Durante um programa, ou seja, quando ele estiver rodando, este flag assumirá valores de 1 e 0 muitas vezes, dependendo do programa é claro.

2 + 2 é uma operação matemática ou aritmética.

0 and 1 é uma operação lógica.

Este bit e R/W, ou seja, permite escrita e leitura.

Bit 1 – DC –> o nome deste bit é **Digit Carry**/**Borrow**. Traduzindo quer dizer que seu valor se altera quando ocorreu a passagem de um bit da posição 3 para a posição 4.

DC = 0 –> não houve um carry-out.
DC = 1 –> houve carry-out de 3º para 4º bit ou de P3 para P4.

Este Flag é R/W.

Bit 0 – C –> este bit se chama **Carry**/**Borrow** e ele indica que ocorreu um carry-out da posição P7 o do bit 7 para a posição P8. veja que não há posição P8. Dizemos, quando isto acontece, que houve um estouro, pois só temos as posições de P0 a P7, ou seja, 8 bits e esta situação ultrapassa este 8 bits

C = 0 –> não houve um carry-out e está tudo normal.
C = 1 –> houve um carry-out do bit 7 para o bit 8 (nona posição). Veja a *figura 21.1* anterior.

Este flag permite a leitura e a escrita, portanto, é R/W.

Depois de um reset o registrador STATUS estará assim:

- Ajustado para banco 0 de memória.

- T0\= 1;

- PD\ = 1.

Outros bits estarão em estado desconhecido.

Este registrador é muito importante e tem seu lugar garantido no cabeçalho do programa. Mais para frente, você verá que poderá utilizar um cabeçalho padrão para fazer os seus programas (com poucas modificações, às vezes).
Você pode usar este registro fora do cabeçalho do seu programa, mas, deve chamá-lo sempre por seu nome, ou seja, STATUS.
A mesma regra se aplica para os outros registradores especiais, salvo raras exceções.

21.2. Registrador OPTION ou OPTION-REG:

Como o nome diz, este registrador permite se escolher uma série de opções do microcontrolador. É através dele que configuramos o prescaler, o TMR0, como será aceita uma interrupção externa, como ficarão as pull-ups do Portb entre outras coisas.
Ele normalmente é chamado de “Option-Reg", pois existem PICs com uma instrução chamada Option e se colocássemos só o nome Option, nosso programa poderia apresentar problemas.

R/W é igual a L/E o quer dizer que estes flags permitem leitura e escrita.

Você verá que no cabeçalho de qualquer programa irá ter que configurar este registrador.

Vamos ver qual a função de cada flag ou bit deste registrador.

Bit 7 – RBPU\ –> este Flag define como estão os resistores de pull-ups do Portb.

Bit 7 = 0 = pull-ups habilitados.
Bit 7 = 1 = pull-ups desabilitados.

Mas, o que vem a ser ***pull-ups***?

O ***pull-up*** é um resistor que é colocado entre um pino do portb e o Vcc internamente.

Mas, como esta ligação é interna não podemos vê-la, mas, podemos configurá-los.

Fica assim então:

Bit 7 = 0 = resistores ligados ao Vcc – configurados como entrdas.
Bit 7 = 1 = resistores dos ligados da Vcc – configurados como saídas.

Ele aceita leitura e escrita.

Bit 6 – INTEDG -> Este flag define como será aceita uma interrupção externa. Se quando o nível subir ou se quando o nível descer no pino Rb0/INT.

Este flag aceita leitura e escrita.

Bit 6 = 0 = a interrupção é entendida quando o pino está em nível alto (Vcc) e desce (terra).
Bit 6 = 1 = a interrupção é entendida quando o pino está em nível baixo (terra) e vai para nível alto (Vcc).

Bit 5 – TOCS -> Este bit ou Flag define a fonte de clock do timer0.

“Você verá alguns softwares neste livro e na Internet, que demonstram o funcionamento do timer 0”.

Bit 5 = 0 = o timer 0 conta através do clock interno. Normalmente usamos esta opção.
Bit 5 = 1 = o timer 0 conta através das mudanças de nível ou clock no pino Ra4/TOCKI.

Bit 4 – TOSE -> Define se o timer0 será incrementado na subida do sinal ou na descida do sinal aplicado em Ra4/TOCKI.

Bit 4 = 0 = conta quando o sinal passa de 0 para Vcc, ou seja, sobe.
Bit 4 = 1 = conta quando o sinal passa de Vcc para 0, ou seja, desce.

Bit 3 – PSA -> Este Flag define com quem o prescaler estará ligado, internamente, no PIC.

Mas, o que é um Prescaler?

Se você já trabalhou com RF deve conhecer o famoso PLL, se não conhece, acesse o site *www.luizbertini.net/dowload.html* e "abaixe" a apostila de PLL, e saberá que, muitas vezes ele trabalha em conjunto com um divisor que é o famoso Prescaler.

Um Prescaler é um divisor que pode ser fixo ou não. Normalmente um Prescaler é utilizado para se dividir uma freqüência e a forma de onda desta freqüência deve ser quadrada.

Nos PICs o valor de divisão dos Prescaler podem ser alterados, mas, tudo dentro de certos padrões.

Os flags responsáveis pela configuração do Prescaler são o PS2, PS1 e PS0. E eles são os bits ou flags bit2, bit1 e bit0.

Todos eles permitem leitura e escrita.

Veja seus nomes e posições:

Bit 2 – PS2 – bit mais significativo.
Bit 1 – PS1 – bit "do meio".
Bit 0 – PS0 – bit menos significativo.

Com os três podemos manter uma tabela com oito possibilidades.

Teoricamente eles seguem a divisão de acordo com esta tabela. Perceba que o timer 0 começa dividindo por 2.
Depois de um *reset* este registrador estará todo em 1 (1 1 1 1 1 1 1 1), na seguinte configuração:

RBPU\=1 = pull-ups desabilitados.

INTEDG = 1 = interrupção na subida.

TOCS = 1 = clock pelo Ra4.

TOSE = 1 = incrementa o clock na descida.

PSA = 1 = prescaler com o Watch-dog.

PS2 = PS1 = PS0 = 1 = divisão de Watch-dog por 128.

Veja que é importante saber o estado após o reset e reconfigurá-lo se necessário.

21.3. Registrador INTCON:

A função básica deste registrador é controlar as interrupções. Uma interrupção é um comando elétrico que pode ser externo ou interno e que obriga o microcontrolador a ir para um determinado

endereço da memória. Em programas simples poucas vezes usamos interrupções, mas, em programas elaborados elas são fundamentais.

O registrador tem 8 bits ou 8 flags.
R/W = L/E = que quer dizer que eles permitem leitura e escrita.

As configurações destes registradores permitirão definir como o microcontrolador trabalhará com as interrupções.

Conhecendo flag a flag o registrador INTCON.

Bit 7 – GIE –> este bit habilita ou desabilita todas as interrupções.

Caso a chave Ch1 esteja aberta, não importa a posição das chaves Ch2 a Ch6, que não circulará corrente (I) entre as pontas (A) e (B). É esta a função do GIE, mais ou menos.

Bit 7 = 0 = todas as interrupções desabilitadas, nenhuma interrupção externa ou interna atua sobre o PIC.
Bit 7 = 1 = permite que as interrupções funcionem de acordo com sua “setagem” ou programação.

Bit 6 – EEIE -> gera uma interrupção no final da escrita do EEPROM ou E2PROM interna.
Bit 6 = 0 = não tem interrupção após acabar a escrita.
Bit 6 = 1 = tem interrupção após acabar a escrita.

Fique esperto com este flag, pois você poderá não conseguir fazer um programa que grave no E2PROM funcionar, devido ao “detalhe” de se esquecer que, caso haja uma interrupção o PIC vai para outro endereço da memória. No PIC 16F628, ele recebe o nome de PEIE e monitora todos os periféricos (atenção).

Bit 5 – TOIE –> interrupção gerada por estouro ou overflow do TMRO ou timer0. Esta é uma interrupção, ou no caso do nosso circuito, uma chave individual.

Bit 5 = 0 = interrupção habilitada, tem um estouro no timer0, tem uma interrupção.
Bit 5 = 1 = interrupção desabilitada.

Bit 4 – INTE –> controle ou flag de controle da interrupção externa no pino Rb0/INT.

Bit 4 = 0 = está desabilitada.
Bit 4 = 1 = está habilitada.

Bit 3 – RBIE –> controla interrupções por mudanças no Portb.

Se estiver habilitada qualquer mudança de estado no portb será interpretada como uma interrupção. Só da Rb4 ao Rb7.

Bit 3 = 0 = não entende mudança no portb como interrupção.
Bit 3 = 1 = entende mudança no portb como interrupção.

Bit 2 – TOIF –> este Flag indica que houve uma interrupção no timer0 por estouro ou overflow.

Bit 2 = 0 = não ocorreu estouro e por isto não houve interrupção.
Bit 2 = 1 = ocorreu estouro e por isto ocorreu o sinal de interrupção, mas, ele só será reconhecido se o bit 5 deixar.

Quando ficar em 1, você deve zerar este flag através de seu software, caso contrário, ele sempre ficará igual a 1.

Bit 1 – INTF –> indica que ocorreu uma interrupção externa através do pino Rb0/INT.

Bit 1 = 0 = não existe nenhum pedido e interrupção.
Bit 1 = 1 = ocorreu pedido e interrupção.

Este flag também deve ser “zerado” pelo seu software.

Bit 0 – RBIF –> indica mudanças do portb e interrupção do portb da Rb4 ao Rb7.

Bit 0 = 0 = não houve mudança de estado em nenhum pino do portb do Rb4 ao Rb7.
Bit 0 = 1 = houve mudanças de nível de tensão em algum pino da portb do pino Rb4 ao Rb7.

Este bit também deve ser “zerado” pelo seu software.

Dá para perceber que este registrador só cuida das interrupções. Quando acontece um reset ele fica assim:

Bit 7:							Bit 0
0	0	0	0	0	0	0	X

Todas as interrupções desabilitadas e o bit 0 não é afetado, mantém o estado que tinha antes do reset. Se era 1 fica 1 e se era 0 fica 0.

Existem muitos outros registradores, mas os conheceremos depois. Calma gente, somos só aprendizes de feiticeiros...

Capítulo 22
REGISTRADORES ESPECIAIS E UTILIZADOS NO PIC 16F628

Além do registrador STATUS e do OPTION ou OPTION_REG que são iguais ao da 16F84, vamos estar vendo aqui outros registradores não usados no 16F84.

Lembre-se, cada PIC pode ter registradores diferentes, o importante é saber o conceito. Releia o capítulo de registradores.

Vamos ver primeiro o INTCON.

Ele é responsável pelas interrupções, mas, como o PIC 16F628 tem EEPROM interna, comparadores internos e uma porta USART de comunicação serial o bit 6, que no 16F84, era associada à EEPROM, neste registrador está associado há todos os dispositivos citados (é como uma chave geral) e teremos mais dois registradores para controlar as interrupções originadas deles. São o PIE1 e o PIR1.

GIE: Habilitação geral das interrupções.
0 = Nenhuma interrupção será tratada.
1 = As interrupções habilitadas individualmente serão tratadas.

PEIE: Habilitação das interrupções de periféricos.
0 = As interrupções de periféricos não serão tratadas.
1 = As interrupções de periféricos habilitadas individualmente serão tratadas.

T0IE: Habilitação da interrupção de estouro de TMR0.
0 = Interrupção de TMR0 desabilitada.
1 = Interrupção de TMR0 habilitada.

INTE: Habilitação de interrupção externa no pino Rb0.
0 = Interrupção externa desabilitada.
1 = Interrupção externa habilitada.

RBIE: Habilitação da interrupção por mudança de estado nos pinos Rb4 a Rb7.
0 = Interrupção por mudança de estado desabilitada.
1 = Interrupção por mudança de estado habilitada.

T0IF: Identificação de estouro do TMR0:
0 = Não ocorreu estouro do TMR0.
1 = Ocorreu estouro do TMR0 (este bit deve ser limpo por você via software).

RBIF: Identificação da interrupção por mudança de estado nos pinos Rb4 a Rb7.
0 = Não ocorreu evento da interrupção.
1 = Ocorreu evento da interrupção (este bit também deve ser limpo por voce).

Agora vamos nos concentrar no bit 6 que é o PEIE e que a faz a diferença entre o PIC16F84 e o PIC 16F628.

Bit 6 – PEIE –> responsável pelas interrupções dos periféricos.
Por periféricos chamamos a E2PROM, os comparadores a USART e os timers.
Este flag é a “chave geral” mais teremos mais dois registradores para dividir entre eles as interrupções dos periféricos.

O registrador PIE1 permite a habilitação e desabilitação das interrupções dos periféricos.

O registrador PIR1 é o sinalizador das interrupções dos periféricos. Ele sempre sinalizará se houver uma interrupção.

Tenha claro isto:

PIE1 –> habilitação.
PIR1 –> sinalização.

Vamos ver o PIE1:

Bit 7 – EEIE –> interrupção do final da escrita EEPROM.
Bit 7 = 0 = não habilitada.
Bit 7 = 1 = habilitada.

Bit 6 – CCIE –> interrupção dos comparadores.
Bit 6 = 0 = não habilitada.
Bit 6 = 1 = habilitada.

Bit 5 – RCIE –> interrupção da USART.

“USART” quer dizer: Universal Síncrono Assíncrono "Rx e TX” – Vulgo comunicação serial.

Bit 5 = 0 = não habilitada.
Bit 5 = 1 = habilitada.

Bit 4 – TXIE –> agora este flag habilita a transmissão (Tx) da USART.
Bit 4 = 0 = interrupção desabilitada.
Bit 4 = 1 = interrupção habilitada.

Bit 3 – Unused (não utilizada).

Bit 2 – CCP1IE –> este Flag habilita a interrupção do CCP (Captura/Comparação/PWM).
Bit 2 = 0 = desabilitada.
Bit 2 = 1 = habilitada.

Bit 1 – TMR2IE –> habilitação da interrupção de estouro do timer 2 (este PIC tem o timer0, o timer1 e o timer2).
Bit 1 = 0 = não habilitada.
Bit 1 = 1 = habilitada.

Bit 0 – TMR1IE –> habilitação da interrupção do estouro do timer 1 do PIC 16F628X.
Bit 0 = 0 = não habilitada.
Bit 0 = 1 = habilitada.

Após o reset fica:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0

Agora o PIR1:

L/E = permitem a leitura e escrita. Mas, o Bit 5 e o Bit 4 precisam para isto, de outros SFRs (registradores). Este registrador indica o que está acontecendo com os periféricos do PIC 16F628/PIC 16F627 e permite a você acessá-lo com seu software, e fazer um programa ou “algoritmo” mais “legal” ou objetivo.

Sempre a indicação é feita pelo dígito 1.

Bit 7 – EEIF –> sinaliza o término de escrita na EEPROM (você pode usá-lo para comandar/controlar uma escrita na EEPROM quando fizer um programa), lembre-se disto.

Bit 7 = 0 = escrita não terminou ou nem começou.
Bit 7 = 1 = escrita terminou.

Bit 6 – CMIF –> sinaliza mudança de estado nas saídas dos comparadores analógicos do PIC 16F62X.

Bit 6 = 0 = não houve mudança na saída.
Bit 6 = 1 = teve mudança de nível na saída.

Bit 5 – RCIF –> indica a recepção de caractere na USART.

Bit 5 = 0 = sem recepção.
Bit 5 = 1 = com recepção.

Bit 4 – TXIF –> sinaliza transmissão da USART.

Bit 4 = 0 = sem transmissão.
Bit 4 = 1 = com transmissão.

Bit 3 – Unused (não utilizada)

Bit 2 – CCP1IF –> sinalização de captura ou de comparação.

Bit 2 = 0 = sem captura ou comparação.
Bit 2 = 1 = com captura ou comparação.

Bit 1 – TMR2IF –> indica estouro no timer2.

Bit 1 = 0 = sem estouro.
Bit 1 = 1 = com estouro “bum”.

Bit 0 – TMR1IF –> indica estouro no timer 1.

Bit 0 = 0 = não houve estouro.
Bit 0 = 1 = houve estouro.

Vamos ver, de novo, o STATUS:

IRP: Seletor de banco de memória de dados usado para endereçamento indireto:
0 = Banco 0 e 1 (00h – FFh).
1 = Banco 2 e 3 (100h – 1FFh).

RP1 e
RP0 Seletor de banco de memória de dados usado para endereçamento direto:
00 = Banco 0 (00h – 7Fh).
01 = banco 1 (80h – FFh).
10 = Banco 2 (100h – 17Fh).
11 = banco 3 (180h – 1FFh).

/TO: Indicação de Time-out:
0 = indica que ocorreu um estouro do Watch-dog Timer (WDT).
1 = indica que ocorreu um power-up ou foram executadas as instruções CLRWDT ou SLEEP.

/PD: Indicação/Power-down:
0 = indica que a instrução SLEEP foi executada.
1 = indica que ocorreu um power-up ou foi executada a instrução CLRWDT.

Z: Indicação de zero:
0 = indica que o resultado da última operação (lógica ou aritmética) não resultou em zero.
1 = indica que p resultado da última operação (lógica ou aritmética) resultou em zero.

DC: Digit Carry/Borrow:
0 = A última operação da ULA não ocasionou um estouro de dígito.

1 = A última operação da ULA ocasionou um estouro (carry) entre o bit 3 e 4, isto é, o resultado ultrapassou os 4 bits menos significativos. Utilizado quando se trabalha com números de 4 bits.

C: Carry/Borrow:
0 = A última operação da ULA não ocasionou um estouro (carry).
1 = A última operação da ULA ocasionou um estouro (carry) no bit mais significativo, isto é, o resultado ultrapassou os 8 bits disponíveis.

Vamos ver, de novo, o OPTION:

/RBPU: Habilita *pull-ups* internos para o PORTB:
0 = *Pull-ups* habilitados para todos os pinos do PORTB configurados como saída.
1 = *Pull-ups* desabilitados.

INTEDG: Configuração da borda que gerará a interrupção externa no RB0:
0 = A interrupção ocorrerá na borda de descida.
1 = A interrupção ocorrerá na borda de subida.

T0CS: Configuração do incremento para o TMR0:
0 = TMR0 será incrementado internamente pelo *clock* da máquina.
1 = TMR0 será incrementado externamente pela mudança no pino RA4/T0CK1.

T0SE: Configuração da borda que incrementará o TMR0 no pino RA4/T0CK1, quando T0Cs = 1:
0 = O incremento ocorrerá na borda de subida de RA4/T0CK1.
1 = O incremento ocorrerá na borda de descida de RA4/T0CK1.

PSA: Configuração de aplicação do prescaler:
0 = O *prescaler* será aplicado ao TMR0.
1 = O *prescaler* será aplicado ao *WDT*.

PS2, PS1 e PS0: Configuração de valor de *prescaler*:

PS2/1/0	TMR0	EDT
0 0 0	1:2	1:1
0 0 1	1:4	1:2
0 1 0	1:8	1:4
0 1 1	1:16	1:8
1 0 0	1:32	1:16
1 0 1	1:64	1:32
1 1 0	1:128	1:64
1 1 1	1:256	1:128

Agora o Pcon que indica a queda de energia, o tipo de oscilador interno e a falta de energia.
PCON:

OSCF: Freqüência do oscilador interno:
0 = 4 MHz.
1 = 37 KHz.

/POR: Indicação de *Power-On Reset* (energização):
0 = Ocorreu um Power-On Reset.
1 = Não ocorreu um Power-On Reset.

/BOR: Indicação de Brown-Out Reset (queda de energia):

0 = Não ocorreu um Brown-Out Reset.
1 = Ocorreu um Brown-Out Reset.

Vamos ver o Pcl e o Pclath que trabalham diretamente com o PC Counter.
PCL e PCLATH:

Para endereçamentos indiretos precisamos deles.
FSR e o INDF:

Comandando o Porta.
PORTA e TRISA:

Comandando o Portb.
PORTB e TRISB:

Verificando o timer0.
TMR0:

Trabalhando com o prescaler, clock interno ou externo, e timer1.
T1CON, TMR1L e TMR1H:

T1CKPS1
T1CKPS0:

Ajuste do prescaler do Timer1:
00 = prescaler de 1:1.
01 = prescaler de 1:2.
10 = prescaler de 1:4.
11 = prescaler de 1:8.

T1OSCEN: Habilitação do sistema de oscilação externa para os pinos T1OSO e T1OSI:
0 = Oscilador desabilitado. Caso exista um cristal externo, o sistema é desligado.
1 = Habilita oscilador externo.

/T1SYNC: Controle de sincronismo interno. Quando TMR1CS = 0 este bit é ignorado:
0 = Sistema de sincronismo ligado.
1 = Sistema de sincronismo desligado (modo assíncrono).

TMR1CS: Seleção da origem do clock para Timer 1:
0 = Clock interno (Fosc/4).
1 = Clock externo no pino T1OSO/T1CK1.

TMR1ON: Habilitação do Timer 1:
0 = Timer1 desabilitado. Paralisa o contador do Timer1.
1 = Timer1 habilitado.

Trabalhando com o timer2, com o prescaler e com o postcaler.
T2CON, TMR2 e PR2:

TOUTPS3

TOUTPS2
TOUTPS1
TOUTPS0: Ajuste do postscale:

0000 = postscaler de 1:1.	1000 = postscaler de 1:9.
0001 = postscaler de 1:2.	1001 = postscaler de 1:10.
0010 = postscaler de 1:3.	1010 = postscaler de 1:11.
0011 = postscaler de 1:4.	1011 = postscaler de 1:12.
0100 = postscaler de 1:5.	1100 = postscaler de 1:13.
0101 = postscaler de 1:6.	1101 = postscaler de 1:14.
0110 = postscaler de 1:7.	1110 = postscaler de 1:15.
0111 = postscaler de 1:8.	1111 = postscaler de 1:16.

TMR2ON: Habilitação do Timer2:
0 = Timer2 desabilitado. Paralisa o contador do Timer2.
1 = Timer2 habilitado.

T2CKPS1
T2CKPS0: Ajuste do prescaler:
00 = prescaler de 1:1.
01 = prescaler de 1:4.
10 = prescaler de 1:16.
11 = prescaler de 1:16

Trabalhando com os comparadores internos.
CMCON:

C2OUT: Valor da saída do comparador 2:
Normal (C2INV=0):
0 = C2 V_{IN+} < C2 V_{IN-}
1 = C2 V_{IN+} > C2 V_{IN-}
Inversa (C2INV=1):
0 = C2 V_{IN+} > C2 V_{IN-}
1 = C2 V_{IN+} < C2 V_{IN-}

CIOUT: Valor da saída do comparador 1:
Normal (C1INV=0):
0 = C1 V_{IN+} < C1 V_{IN-}
1 = C1 V_{IN+} > C1 V_{IN-}
Inversa (1INV=1) :
0 = C1 V_{IN+} > C1 V_{IN-}
1 = C1 V_{IN+} < C2 V_{IN-}

C2INV : Tipo de saída do comparador 2 :
0 = Normal.
1 = Inversa.

C1INV: Tipo de saída do comparador 1:
0 = Normal.
1 = Inversa.
CIS: Chave seletora de entrada do comparador:
Quando CM2:CM0 = 001
0 = RA0 conectado a C1 V_{IN-}
1 = RA3 conectado a C1 V_{IN+}
Quando CM2:CM0 = 010

0 = RA0 conectado a C1 V_{IN-}
RA1 conectado a C2 V_{IN+}
1 = RA3 conectado a C1 V_{IN-}
RA2 conectado a C2 V_{IN+}

CM2, CM1
CM0: Configura a pinagem dos comparadores (modo de operação):

Trabalhando com as tensões de referencia.
VRCON:

VREN: Energização do sistema de tensão de referência:
0 = Circuito de V_{REF} desenergizado.
1 = Circuito de V_{REF} energizado.

VRON: Habilitação da saída de V_{REF}:
0 = Tensão de referência desligada.
1 = Tensão de referência ligada ao pino RA2.

VRR: Seleção do range de operação do sistema de V_{REF}:
0 = Range baixo.
1 = Range alto.

VR3.VR0: Seleção do valor da tensão de V_{REF}:
Se VRR = 1:
$V_{REF} = (VR/24) * V_{DD}$
Se VRR = 0:
$V_{REF} = ¼ * V_{DD} + (VR/32) * V_{DD}$

Capture, compare e PWM.
CCP1CON, CCPR1L e CCPR1H :

CCP1X
CCP1Y: Parte baixa do PWM de 10 bits. A parte alta fica em CCPR1L. Válido somente quando em PWM.

CCP1M3
CCP1M2
CCP1M1
CCP1M0: Seleção do modo CCP1 – Compare/Capture/PWW:
0000 = Modo desligado.
0100 = Capture ligado para borda de descida com prescaler de 1:1.
0101 = Capture ligado para borda de subida com prescaler de 1:1.
0110 = Capture ligado para borda de subida com prescaler de 1:4.
0111 = Capture ligado para borda de subida com prescaler de 1:16.
1000 = Compare ligado. Pino de saída (RB3) será setado (1) quando o compare ocorrer.
1001 = Compare ligado. Pino de saída (RB3) será zerado (0) quando o compare ocorrer.
1010 = Compare ligado. Pino de saída (RB3) não será afetado.
1011 = Compare ligado. Pino de saída (RB3) não será afetado. TMR1 será resetado.
1100 = PWM ligado.
1101 = PWM ligado.

1110 = PWM ligado.
1111 = PWM ligado.

Falando da eeprom interna.
EECON1, EECON2, EEADR e EEDATA:

WRERR: Identificação de erro durante a escrita na EEPROM:
0 = Não ocorreu erro, a escrita foi completada.
1 = Em erro ocorreu por uma escrita não terminada (um reset pode ter ocorrido).

WREN: Habilitação de escrita na EEPROM (bit de segurança):
0 = Não habilita a escrita na EEPROM.
1 = Habilita a escrita na EEPROM.

WR: Ciclo de escrita na EEPROM:
0 = Este bit será zerado pelo hardware quando o ciclo de escrita terminar (não pode ser zerado por software).
1 = Inicia o ciclo de escrita (deve ser setado por software).

RD: Ciclo de leitura da EEPROM:
0 = Este bit será zerado pelo hardware quando o ciclo de leitura terminar (não pode ser zerado por software).
1 = Inicia o ciclo de leitura (deve ser setado por software).

Com eles conseguimos nos comunicar.
TXSTA e RCSTA:

CSRC: Seleção entre Máster/Slave (somente modo síncrono):
0 = Slave.
1 = Máster.

TX9: Habilitação da comunicação em 9 bits para a transmissão:
0 = Transmissão em 8 bits.
1 = Transmissão em 9 bits.

TXEN: Habilitação da transmissão:
0 = Transmissão desabilitada.
1 = Transmissão habilitada. No modo síncrono, a recepção tem prioridade sobre este bit.

SYNC: Seleção entre modo assíncrono/síncrono:
0 = Assíncrono.
1 = Síncrono.

BRGH: Seleção para Baud Rate (somente modo assíncrono):
0 = Baud Rate baixo.
1 = baud Rate alto.

TRMT: Seleção do registrador interno de transmissão (TSR):
0 = TSR cheio.
1 = TSR vazio.

TX9D: Valor a ser transmitido como 9º bit. Pode ser usado como paridade ou endereçamento.

SPEN: Habilitação da USART:
0 = USART desabilitada.
1 = USART habilitada.

RX9: Habilitação da comunicação em 9 bits para a recepção:
0 = Recepção em 8 bits.
1 = recepção em 9 bits.

SREN: Habilitação da recepção unitária (somente para modo síncrono em Máster):
0 = Recepção unitária desabilitada.
1 = Recepção unitária habilitada. Depois de receber um dado, desliga-se automaticamente.

CREN: Habilitação da recepção contínua:
0 = Recepção contínua desabilitada.
1 = Recepção contínua habilitada.

ADDEN: Habilitação do sistema de endereçamento (somente modo assíncrono de 9 bits):
0 = Desabilita sistema de endereçamento.
1 = Habilita sistema de endereçamento.

FERR: Erro de Stop bit (somente modo assíncrono):
0 = Não ocorreu erro. Stop bit = 1
1 = Ocorreu um erro. Stop bit = 0 (deve ser atualizado lendo o registrador RCREG e recebendo o próximo dado valido).

OERR: Erro de muitos bytes recebidos sem nenhuma leitura:
0 = Não houve problemas de estouro do limite.
1 = Estouro do limite de 3 bytes recebidos antes da leitura de RCREG (para limpar deve-se zerar o bit CREN).

RX9D: Valor recebido no 9º bit. Pode ser usado Omo paridade ou endereçamento.

Ainda nos comunicando.
TXREG e RCREG.

Ajustando a velocidade de comunicação.
SPBRG.

CAPÍTULO 24
TEORIA DA GRAVAÇÃO - MANUAL EXSTO

A Exsto cedeu gentilmente, uma placa Pratic628, para as simulações dos softwares escritos e usados neste livro, por isto citamos aqui o manual desta placa, esperamos que você, adquirindo-a, tenha o seu estudo e aprendizado muito mais completo.

Introdução:
Parabéns! Você acaba de adquirir um produto de alta qualidade e tecnologia de ponta. O **Pratic 628** será o grande auxílio no aprendizado e desenvolvimento com microcontroladores da linha PICMicro da Microchip.
A Exsto Tecnologia é uma empresa situada em Santa Rita do Sapucaí, Minas Gerais, conhecida como “Vale da Eletrônica” por seu destaque na indústria eletroeletrônica e pela excelência de suas

instituições de ensino. Nossa missão é sempre fornecer as melhores ferramentas para o desenvolvimento e aprendizado em eletrônica e, em especial, microcontroladores. Visite nosso site ***www.exsto.com.br*** para conhecer outras soluções e produtos oferecidos.
Este documento contém as principais características do Sistema de desenvolvimento **Pratic 628** e visa ser o guia de instalação e utilização desse sistema.
O **Pratic 628** é um ambiente de desenvolvimento composto por um hardware e software visa facilitar o aprendizado e desenvolvimento com microcontroladores das linhas PIC 16Xxxx utilizando especialmente o PIC16F628, além de maximizar as possibilidades de experimentos.
O software (em português) realiza basicamente as seguintes funções:
® Permite a edição de programas;
® Compila/Monta os programas;
® Grava os programas compilados no microcontrolador presente na placa principal.
O hardware do **Pratic 628** foi desenvolvido procurando disponibilizar o máximo de recursos possíveis ao PIC16F628. Neste sentido, o kit contém diversos circuitos que são ligados ao microcontroladores através de jumpers, permitindo assim realizar um grande número de montagem mesmo com um microcontrolador de poucas entradas e saídas.
A escolha do PIC16F628 se deve a popularidade desse componente, que agrega periféricos avançados sem deixar de ser simples. Além disso, ele é compatível com o **clássico PIC16F84.**

1. Componentes Suportados:
O **Pratic 628**, incluindo funcionalidades de hardware e software, suporta trabalhar com os microcontroladores listados abaixo, todos eles compatíveis pino a pino com o PIC16F628A.

PIC16F627	PIC16LF627A
PIC16F627A	PIC16LF627A
PIC16F628	PIC16LF628
PIC16F628A	PIC16LF628A
PIC16F648A	PIC16LF648A

Além desses componentes, a Exsto Tecnologia está em constante trabalho de atualização do software, criando novas versões para trabalhar com os novos lançamentos em microcontroladores PIC suportados pela placa. Visite periodicamente o site ***www.exsto.com.br*** onde estão disponíveis as atualizações.

2. Software Pratic 628:
1) Componentes da janela principal:
Ao se abrir o programa tem-se a janela principal, como mostrada abaixo com os seguintes componentes:

a) Menus: Arquivo, Editar, Ferramentas, Janelas:
Através dos diversos menus é possível ter acesso ao todas as funcionalidades do sistema.
b) Barra de ferramentas:
As funções principais localizam-se nesta barra, agilizando os seus acessos.
c) Barra de status:
i. Indica se as teclas, CAPS LOCK, INSERT e NUM estão acionadas ou não;
ii. Informa a linha do cursor na janela de edição;
iii. Informa o processador que está sendo utilizado;
iv. Informa o compilador utilizado;
v. Mostra mensagens relativas a vários comandos;
vi. Mostra a hora atual do sistema.

2) Como criar um novo arquivo:
Para criar um novo arquivo siga os passos abaixo:
a) No menu escolha *Arquivo-Novo*, ou através da barra ferramentas;
b) Escolha um diretório onde deseja salvar o projeto;

c) Digite o nome do arquivo;
d) No menu escolha *Arquivo – Salvar Como*, *Salve* o arquivo no diretório desejado;
e) Em intervalos regulares de tempo, salve o projeto através do menu Projeto – Salvar, ou através da barra de ferramentas 🖫.

3) Abrindo um arquivo existente:
Para abrir um arquivo criado anteriormente, siga os passos abaixo:
a) No menu escolha *Arquivo – Abrir*, ou através da barra de ferramentas 🗁.
b) Navegue até a pasta onde se encontra o arquivo e abra o arquivo desejado.

4) Montando/Compilando arquivos:
Para montar um arquivo em Assembly ou compilar um arquivo em C, siga os passos abaixo:
a) Clique na janela de edição do arquivo que deverá ser compilado;
b) No menu escolha *Ferramentas – Compilar* ou através da barra de ferramentas🗏, ou ainda através da tecla de atalho *F5*;
c) Uma janela de status do compilador aparecerá indicando o andamento do processo de compilação do arquivo;
d) Ao final da compilação, aparecerá a janela *Resultados da Compilação*, listando todos os erros, mensagens e avisos gerados pelo compilador.

5) Componentes da janela Resultados da Compilação:

6) Resultados da Compilação:
a) Mostra a qualidade erros, mensagens e avisos contidos no arquivo fonte compilando;
b) Lista todos os erros, mensagens e avisos da compilação, indicando:
i. o número;
ii. a descrição;
iii. a linha onde se localiza no arquivo fonte;
iv. qual o arquivo a que se refere.
Obs.: A partir deste ponto os erros, mensagens e aviso gerados pelo compilador, serão chamados somente de mensagens, ficando entendido que se trata dos 3 tipos.
c) Comentário do erro, mensagem ou aviso selecionado com o mouse;

7) Interpretando e localizando os resultados da compilação:
Para localizar a linha que se refere à mensagem gerada na janela *Resultados de Compilação*, siga os passos abaixo:
a) Clique na mensagem que desejar localizar;
b) A linha correspondente à mensagem na caixa de listagem ficará em azul, indicando a sua seleção;
c) Na janela de edição, o cursor irá até a linha referente à mensagem e irá selecioná-la inteiramente;
d) Na caixa de comentário da mensagem da janela *Resultados de Compilação*, aparecerá um pequeno texto explicativo referente à mensagem selecionada.

8) Alterando as configurações de montagem/compilação:
Algumas configurações podem ser alteradas ou adicionadas na linha de compilação executada pelo compilador, para alterá-las, siga os passos abaixo:
a) No menu escolha *Ferramentas – Configurações de Compilador*, ou através da tecla de atalho *F6*;
b) O software pode trabalhar com duas linguagens de programação. Uma delas é o Assembly, utilizando o montador MPASM da Microchip fornecido juntamente no CD. A outra é o C-CCS, utilizando o compilador C do fabricante CCS, que não é fornecido. A seleção de qual compilador é feita na opção Compilador da janela de configuração do compilador. Para cada compilador há opções diferentes, conforme abaixo:
    a) Montador MPASM:

i. Escolha de processador;
ii. Formato do arquivo ".hex" a ser gerado;
iii. Diferenciação de letras maiúsculas;
iv. Permite o uso de Macros;
v. Base numérica padrão: decimal (DEC), hexadecimal (HEX) ou octal (OCT);
vi. Configuração de tabulação.

A linha que será executada com as opções selecionadas é mostrada logo abaixo na janela.
b) Compilador CCS;
i. Indica o local onde se espera que esteja instalado o compilador;
ii. Permite checar se a versão do compilador é compatível com o gravador;
iii. Mostra a linha de comando que será utilizada para a execução do compilador.

9) Trabalhando com placa de gravação:

Para trabalhar o sistema de gravação, siga os passos:

a) No menu escolha *Ferramentas – Programador*, ou através da barra de ferramentas, ou ainda através da tecla de atalho *F9*;
b) A janela *Gravador* aparecerá.

Obs.: *Para usuários do WinNT, Win2000 ou WinXP, tenha a certeza de que o software foi instalado para usuários que possuam as permissões necessárias para acessar a porta paralela do PC*.

c) A janela *Gravador* possui os seguintes componentes:
i. Configurações de processador;
ii. Bits de configuração dos fusíveis;
iii. Comandos de gravação;
iv. Status de andamento do processo.

10) Executando comandos de gravação:
a) Apagando:
Lê toda a memória de programa, verificando se está apagada.

b) Programar:
i. Programa a memória de programa do dispositivo, de acordo com os dados contidos na janela *Memória de programa*;
ii. Programa a memória de dados do dispositivo, de acordo com os dados contidos na janela *Memória de dados*. Se esta janela não estiver aberta, a memória de dados do dispositivo permanecerá inalterada;
iii. Programa dos bits de fusíveis de acordo com a configuração escolhida.

Obs.: Caso se esteja utilizando os componentes PIC16F628/627 fica disponível a opção MCLR. Caso essa opção seja selecionada "como I/O" o chip deve ser apagado antes de realizar uma regravação.

c) Verificar:
i. Lê a memória de programa do dispositivo, comparando com os dados contidos na janela *Memória de programa*;
ii. Lê a memória de dados do dispositivo, comparando com os dados contidos na janela *Memória de dados*, caso esta janela esteja aberta.

d) Ler:
i. Lê memória de programa do dispositivo, colocando os dados nos endereços correspondentes na janela *Memória de programa;*
ii. Lê memória de dados do dispositivo, colocando os dados nos endereços correspondentes na janela *Memória de dados*, caso esta janela esteja aberta.

iii. Lê os bits de fusíveis e reconfigura às opções mostradas na janela *Gravador* seção *Bits de configuração*, de acordo com os bits lidos do dispositivo;

iii. Lê o Device ID.

e) Apagar:
Apaga todos os dados do dispositivo, incluindo, memória de programa e dados, bits de configuração e Device ID.

11) Trabalhando com arquivos ".hex" separadamente de um projeto:
É possível trabalhar abrindo e salvando arquivos ".hex" separadamente. Por exemplo para abrir um arquivo ".hex" de memória de programa, siga os seguintes passos:
a) No menu escolha *Arquivo – Abrir programa* (**.hex*);
b) Escolha o arquivo desejado;
c) A janela *Memória de programa* aparecerá, com o conteúdo do arquivo.
Siga os mesmos passos para abrir um arquivo de memória de dados, neste caso aparecerá à janela *Memória de dados* contendo os dados do arquivo que foi aberto.
Estando os dados nas janelas de memória de programa ou de dados, pode-se gravar no dispositivo estes dados.

12) Abrindo o manual do componente:
É possível abrir o manual do microcontrolador que estiver sendo usado. Para isso é necessário que o Acrobat Reader esteja adequadamente instalado no computador. A instalação desse programa está no CD. Para abrir o manual do arquivo basta clicar no ícone.

3. Hardware

A figura abaixo a placa didática Pratic 628 baseada no PIC16F628.

Na figura acima são enumeradas as principais aplicações de hardware, listadas abaixo:

1. PIC16F628;
2. Circuito de gravação in-circuit;
3. Conector de acesso ao portal B;
4. Conector de acesso ao portal A;
5. Chave de reset ou uso geral;
6. Chave de interrupção ou de uso geral;
7. Chave de contador ou uso geral;
8. Dois displays de sete segmentos multiplexados;
9. Oito Leds;
10. Dip Switch de oito chaves;
11. LDR (resistor variável com luminosidade) com ajuste;
12. NTC (resistor variável com temperatura) com ajuste;
13. Buzzer;
14. Lâmpada incandescente;
15. Um relé;
16. Conectores para acesso aos terminais do relé;
17. Jumpers para configuração das funções do port B;
18. Jumpers para configuração das funções do port A;
19. Trimpot;
20. Conector paralelo DB25F (para gravação);
21. Circuito de alimentação;
22. Conector para fonte de alimentação externa;
23. TRIAC;
24. Detector de cruzamento por zero;
25. Chave "Programar/Executar".

3.1. Descrição do hardware:

O sistema é baseado no PIC16F628A, que apresenta as características:

® 2048 palavras de 14 bits de memória de programa FLASH;
® 224 bytes de memória de dados RAM;
® 128 bytes de memória de dados EEPROM;
® 3 Temporizadores/Contadores;
® 2 Comparadores analógicos;
® Módulo CCP (Captura/Comparação/PWM);
® Comunicação serial síncrona e assíncrona (USART);
® Tensão de referência interna programável;
® Encapsulamento DIP de 18 terminais;
® 16 terminais configuráveis como entrada ou saída independente;
® 10 interrupções;
® Vários modos de oscilador (XT, LP, HS, RC interno e externo).

A gravação do componente é feita *in-circuit*, por um circuito já presente na placa. Isso significa que o componente não necessita ser retirado da placa para gravação; basta apenas mudar a posição da chave RUN/PROG para "Programar" (PROG) e gravar, mudar a chave para "Executar" (RUN) e rodar o programa gravado, conforme mostrado na figura abaixo. Dessa forma é aumentado consideravelmente o tempo de vida do componente, evitando o desgaste/quebra dos terminais, decorrente das retirada constante do microcontrolador da placa para gravação.

Executar (RUN)

Programar (PROG)

Quanto aos aplicativos de hardware presentes na placa, foi elaborado um conjunto de circuitos que permitisse uma grande gama de experimentos. Esses circuitos são ligados ao microcontrolador por *jumpers*, o que torna o sistema extremamente maleável. Além disso, os terminais dos portais do microcontrolador estarão acessíveis em dois conectores, de forma que o usuário pode interligar a placa aplicações desenvolvidas por ele mesmo.

O sistema possuirá chaves ligadas a terminais com funções especiais (reset, interrupção, contador). Tais terminais também podem ser configurados como entradas normais. Para entrada de dados há também um *Dip Switch* (conjunto de oito chaves). Abaixo dessas chaves está escrito na serigrafia da placa os pesos binários de cada bit do port A, ou seja, a chave "1" se refere ao bit 0, a chave "128" se refere ao bit 7, e assim por diante. Conta também com dois displays de sete segmentos para apresentação de valores numéricos. Existem também oito LEDs que podem apresentar valores de oito bits ou serem acionados individualmente. Os LEDs possuem valores na serigrafia que indicam em qual pino do port B eles estão ligados, de forma análoga ao *Dip Switch*.

PORT A **PORT B**

Dip Switch **LEDs**

Foram incluídos também um *buzzer* (buzina) e uma lâmpada incandescente. Ambos podem ser acionados através de uma saída normal ou utilizando o módulo de PWM.

Está presente também um relé para ser acionado pelo microcontrolador. Há um conector de tipo KRE que permite o acesso aos terminais do relé, conforme mostrado a seguir.

A placa conta também com um circuito de controle de carga AC com TRIAC, com o circuito de detecção de cruzamento por zero já incorporado.

Todo o circuito AC é opto-acoplado, ou seja, isolado da placa.

Para fazer uso dos módulos comparadores analógicos existem um Trimpot de Referência, um LDR (resistor controlado por luminosidade) e um NTC (resistor controlado por temperatura). Tanto o LDR como o NTC tem trimpots para ajustes.

A placa suporta ainda a conexão de um teclado e um display tipo telefônico, fornecidos separadamente.

® O esquema elétrico da placa encontra-se no CD na pasta “Esquemas”.

3.2. Jumpers:
A tabela abaixo apresenta as funções selecionáveis através dos jumpers. A seqüência dos jumpers é a mesma da placa.

® Para a utilização de teclado e display todos os jumpers devem ser retirados, com exceção de JP3.
A tabela a seguir traz os sinais associados a cada um dos pinos do microcontrolador. No caso dos pinos ligados a jumpers, a coluna “jumper” trás essas indicações. Segue abaixo uma legenda da tabela:
SEG_X: São os segmentos do DISPLAY de 7 Seg;
CL_X: São as colunas do Teclado Matricial;
L_X: São as linhas do Teclado Matricial.

• Para uso de LCD e de teclado todos os jumpers, com exceção do JP3 devem ser retirados.

4. Instalação
4.1. Instalação do Software:
Para instalar o software:
® Utilizando o Explorer, clicar no drive de CD-ROM (D: ou E);
® Entrar na pasta *Instalação*;
® Executar o aplicativo *Setup*, através de um clique duplo;
® Seguir as orientações do software de instalação;
® Para usuário de WinNT, WinXP ou Win2000, após a instalação do software do Pratic 628, executar o arquivo port95nt.exe em “\instalação” do CD-ROM (caso não faça isto o software não rodará).

Para utilizar os códigos de exemplo:
® Utilizando o Explorer, clicar no drive de CD-ROM (D: ou E:);
® Copiar a pasta exemplos para o HD do computador;
® Antes de utilizar um código fonte, verificar em suas propriedades se ele não está como “Somente leitura”.

Para verificar as propriedades de um arquivo:
® Entrar na pasta que contém e clicar sobre o arquivo com o botão direito do mouse;
® Escolher propriedades (última opção do menu);
® No campo *Atributos* verificar se a opção *Somente Leitura* está selecionada.

➡ Para retirar o atributo de somente leitura do arquivo basta desmarcar a opção *Somente Leitura*.

4.2. Instalação do Hardware:
Para a instalação do hardware devem ser seguidos os seguintes passos:
® Conectar um extremidade de cabo DB25M-DB25M (paralelo) à porta LPT1 (porta de impressora) do computador;
® Conectar outra extremidade do cabo DB25M-DB25M ao conector DB25 da placa principal do Pratic 628;
® Pó último conectar-se a fonte à placa principal pelo conector.

➡ Atenção para tensão da tomada! A chave 110/220 da fonte deve estar corretamente selecionada.

Para a conexão dos módulos de display e teclado devem ser observados os seguintes cuidados:
® Ao conectarem-se os cabos aos módulos e à placa principal deve-se estar atento para a indicação do sentido do conector;

® **Recomenda-se fazer a conexão dos módulos antes de se ligar a fonte à placa principal.**

5. Resolvendo problemas

1 – Quando eu clico em "programar" o gravador realiza a gravação e mostra a mensagem que a gravação foi concluída, mas quando em "verificar" aparece um aviso de erro na verificação.

Há duas possibilidades:
1ª) A opção "code protect" (proteção e código) não está desligada, portanto o programa gravado está protegido contra leitura;
2ª) Por algum outro motivo o microcontrolador não está sendo gravado. *Veja a questão 2*:

2 – O microcontrolador não está sendo gravado:
1ª) A fonte sai de fábrica ajustada para 220V. Se ligada em 110V o único sintoma será a não gravação. Selecione corretamente a tensão da fonte;
2ª) A chave "run/prog" deve estar em posição "prog" durante o processo de gravação;
3ª) O microcontrolador que está sendo utilizado deve estar devidamente especificado no campo "Processador" da janela "gravador", **há diferença entre s modelos com "final A" e os comuns no que se refere ao processo de gravação.**
4ª) A janela "memória de programa" deve estar aberta e contendo o programa a ser gravado, durante o processo de gravação. Quando for compilar/montar um programa, mantenha a janela ("Gravador" aberta). Quando o arquivo for compilado será aberta à janela "Memória do Programa" (e a janela "Memória de dados", se estiver sendo usada). Não feche essas janelas;
5ª) O jumper JP5 deve estar conectado em qualquer posição (1 ou 2). Haverá erro na gravação se ele for retirado.

3. Não consigo ler um pic. Quando clico em "ler" aparece:
a) todos os endereços 0000.
O microcontrolador está protegido contra leitura. Não é possível lê-lo nessas condições.
b) todos em 3FFF.
1º) o microcontrolador está apagado;
2º) tendo certeza de que ele está gravado, pode haver algum problema na gravação. Veja a solução da Questão 2:

4. Quando vou compilar/montar um arquivo de exemplo, aparece à mensagem "Erro número 5".
O arquivo em questão tem propriedade somente leitura. Veja a seção "Instalação do Software" nesse manual como resolver esse problema.

5.1. Suporte Técnico:

A Exsto tecnologia oferece suporte técnico gratuito para questões de utilização de seus produtos através do e-mail ***suporte@exsto.com.br*** ou do telefone (35) 3471-6898.

A placa da Exsto vêm, de fábrica, sem o cristal de 4Mhz e os capacitores de 15pF. Isto permite que você use as I/Os Ra6 e Ra7 do PIC, mas deverá usar o oscilador interno do mesmo. Em alguns exemplos deste livro usaremos o cristal no hardaware e o oscilador XT no software, para isto você deverá soldar um cristal e dois capacitores de 15pF no local indicado. Eu aconselho que você solde terminais torneados nestas posições, de forma a conseguir apenas encaixar, quando necessário, o cristal e os capacitores. Aviso você, leitor, sobre isto, para que possa fazer o máximo de experiências possíveis usando os códigos fontes contidos neste livro e na homepage da Exsto (http://www.exsto.com.br)ou na minha página sobre microcontroladores
http://www.luizbertini.net/microcontroladores/microcontroladores.html

Antes de cada código fonte haverá um alerta se você deve ou não usar o cristal externo de 4 MHz.
Você também poderá trabalhar com um cristal de 20Mhz, contanto que use um PIC 16F628A – 20 que oscila em até 20Mhz.
Lembre-se que, se usar um cristal de 20Mhz deverá setar o oscilador para HS.

Capítulo 25
RESUMO DAS INSTRUÇÕES

25.1. Algumas Dicas sobre as Funções:

Tudo o que está escrito abaixo é baseado na prática:

Instruções muito usadas em Loopings:
goto
call
return
decfsz
clrwdt
incfsz
nop
movwf
movlw

Instruções usadas em leitura de chaves:
btfsc
btfss
nop

Instruções usadas no cabeçalho:
movlw
movwf
bcf
bsf
clrw
retfie

Mas, e estas “letrinhas” f, k, W, d, R, b o que significam?

f –> é um registro que deve estar definido no campo do cabeçalho onde colocamos as variáveis. Deve ter o valor entre 0 a 127 decimal.

k –> é normalmente, um valor que será carregado, normalmente de novo, no registrador de trabalho W. É um número entre 0 a 255 decimal. Pode ser chamada de constante.

W –> é o registrador de trabalho, tudo ou quase tudo, que o PIC faz passar por ele.

d –> o d pode ser colocado ou não, se for colocado, o resultado será salvo nele, mas se não for colocado o resultado será salvo em f. Ele especifica o destino do resultado da instrução:
Se d = 0 o resultado é gravado em W;
Se d = 1 o resultado é gravado no registrador indicado na operação, ou seja, f.
Se não for colocado o padrão será 1, ou seja, o resultado será gravado no registrador f da operação em W.

R –> é um nome que você dará há uma rotina ou a uma parte do programa. Usado com goto e call.

b –> o b define um bit de um determinado registrador e você precisa usá-lo para que as instruções que o antecedem funcionem corretamente. Deve se um número entre 0 a 7 decimal.

Observações:
Os números estão indicados em decimal, mas, podem ser em binário ou hexadecimal, basta estarem dentro dos valores apropriados.
Depois de cada instrução, devemos usar o ponto e vírgula (;) para separá-lo dos comentários. E vamos começar isto logo. Os comentários ajudam.

Capítulo 26
COMO FUNCIONAM AS INSTRUÇÕES

Vamos estudar as instruções, uma a uma, desta forma:
- Instrução.
- Comentários.
- Significado.
- Explicação.
- Exemplo teórico / Prático.
- Flags afetados nos registros.

Todos em ordem alfabética, para ficar mais fácil de encontrá-los.

Instrução:
addlw k ; comentários

Significado:
- Soma a constante k (valor numérico entre 0 a 255) a W.

Explicação:
- O valor de k será somado a W e o valor resultante será no próprio W. É importante que o valor não seja maior do que 255, pois temos 8 bits para trabalhar e 2^8 nos proporciona 256, posições, ou seja, de 0 a 255.

Exemplo:
```
addlw 45            ; o resultado desta instrução colocará
                    ; o valor de W = 45.
```

Flags afetados:
C, Z, DC –> todos do registrador STATUS.

Instrução:
addwf f, d ; comentários

Significado:
- Soma o valor de W com f.

Explicação:
- O valor de W será somado com o valor que existir em f.
Se d = 0 o resultado será salvo em W.

Se d = 1 o resultado será salvo em f.
O padrão é d = 1 e, na maioria dos programas é assim que será, e o valor será salvo em f.

Para salvar em W use a instrução assim:

Addwf f, 0 ;

Exemplo:

W = 10 e f = 7

```
addwf  f        ; somará 10 que é o valor de W
                ; com 7 que é o valor de f
                ; e o resultado será f = 17
```

Mas, se a instrução for assim:

```
addwf f, 0      ; somará 10que é o valor de W
                ; com 7 que é o valor de f
                ; e o resultado será W = 17.
```

Veja a diferença de se colocar d = 0 (não se coloca d = 1 pois, já é o padrão).

Flags afetados:
C, Z, DC –> todos do registrador STATUS.

Instrução:
andlw k ; comentários

Significado:
- Executa a função lógica AND ou E entre W e o valor numérico k.

Explicação:
- Será efetuada uma operação and bit a bit e para visualizarmos isto é mais fácil ver em binário. O resultado será salvo em W.

Exemplo:
Se W = 1 1 1 1 0 0 0 0 e k = 0 0 0 0 1 1 1 1

```
andlw k ; iremos fazer a operação E bit a bit
                ; o que resultará no seguinte:
                ;       W = 1 1 1 1 0 0 0 0
                ;       k = 0 0 0 0 1 1 1 1
                ;           0 0 0 0 0 0 0 0
                ; o valor salvo em W será igual
                ; a 0 0 0 0 0 0 0 0 ou zero. Portanto:
                ; W = 0
```

Flags afetados:
Z –> do registrado STATUS. Neste caso Z será igual a 1 pois o resultado da operação foi 0.
Se fosse diferente de Z seria igual a 0.

Instrução:
andwf f, d ; comentários

Significado:
- Executa a função lógica “E” entre W e F.

Explicação:
- Será feita uma operação “E”, bit a bit, entre W e f.

Exemplo:
Se W = 0 0 0 0 1 1 1 1 efF = 1 1 1 1 0 0 0 1

```
andwf  f; operação E bit a a bit
                    ; e visto resultará em:
                    ;   W = 0 0 0 0 1 1 1 1
                    ;   f = 1 1 1 1 0 0 0 1
                    ;         0 0 0 0 0 0 0 1
                    ; o valor salvo em f será:
                    ; igual a 1.
```

Mas, se a instrução for assim:

```
andwf f, 0      ; o valor agora será salvo em
                ; W, isto devido ao o depois
                ; do f.
```

Flags afetados:
Z –>registrador STATUS. Neste caso Z = 0 pois o resultado da operação foi diferente de 0.

Instrução:
bcf f, b ; comentários

Significado:
- Faz o bit b do registrador f igual a 0.

Explicação:
- O bit indica por b (que deve ser um número de 0 a 7) será zerado no registrador f.

Exemplo:
Se f = 1 1 1 1 1 1 1 1 = portb

```
bcf portb, 7    ; coloca zero no bit 7 do porb
                ; o que faz o valor do portb ficar
                ; igual a 0 1 1 1 1 1 1 1
```

Flags afetados:
- Nenhum.

Instrução:
bsf f, b ; comentários

Significado:
- Faz o portb do registrador f igual a 1.

Explicação:
- O bit indicado por b, que deve ser um bit de 0 a 7, será colocado em nível lógico 1 (5Vpp neste PIC).

Exemplo:
Se F = 0 0 0 0 0 0 0 0 = portb

```
bsf portb, 0      ; fará com que o bit 0 do portb
                  ; seja 1
                  ; veja: 0 0 0 0 0 0 0 1
```

Flags afetados:
- Nenhum

Instrução:
btfsc f, b ; comentários

Significado:
- Testa o bit b do registrador f e pula a instrução seguinte se b = 0.

Explicações:
- O bit b do registrador f será testada, se for 0 a próxima instrução será pulada se for 1 a próxima instrução será lida. Esta instrução é muito utilizada em testes condicionais associadas a chaves.

Exemplo:

Se o portb = 0 0 0 0 0 0 0 1

```
btfsc portb, 0    ; testa o bit 0 da portb
nop               ; como ele é 1 a próxima instrução
goto  A           ; é lida
                  ; como a instrução seguinte é um NOP
                  ; o PIC esperará um ciclo de máquina.
```

Se o portb = 1 1 1 1 1 1 1 0

```
btfsc    portb, 0       ; testa o bit 0 da portb
nop                     ; como ele é 0 pula o NOP
goto  A                 ; e vai para a instrução GOTO
```

Flags afetados:
- Nenhum

Instrução:
btfss f, b ; comentários

Significado:
- Testa o bit b do registrador f e pula a próxima instrução se b = 1

Explicação:
- Supondo que o bit b seja uma entrada do portb podemos usar esta instrução para testar se esta, que está ligada com uma chave, está ligada no Vcc ou no terra.
Se estiver no Vcc (1) a próxima instrução será pulada, se estiver no terra (0) será lida.

Exemplo:
Se portb = 1 0 0 0 0 0 0 0

```
btfss portb, 7    ; testa o bit 7 do portb
goto  X           ;como ele é 1 a próxima instrução
goto  Y           ; será pulada e o programa não irá
                  ; para X mas, sim para Y
```

Mas se fosse assim:
Se portb = 0 0 0 0 0 0 0 0

```
btfss  portb, 7    ; testa o bit 7 da portb
goto   X           ; como ele é 0 lê a instrução
goto   Y           ; seguinte e vai para X
```

Dá para perceber que com um simples apertar de botão o seu programa pode fazer uma coisa ou outra.

Flags afetados:
- Nenhum.

Instrução:
call k ; comentários

Significado:
- Chama uma sub-rotina

Explicação:
- Esta instrução chama uma sub-rotina e pula um pedaço do programa, mas, para saber como voltar, entra em cena o **"Famoso C ou Program Counter".** Quando você usa esta instrução, e digamos que ele esteja na linha 40, o PC salva o endereço 40 + 1 ou 41 e na hora de voltar da sub-rotina, sabe que deve voltar para a linha 41. Simples não.

Exemplo:

```
call tempo       ; desvia o programa para a
linha depois     ; sub-rotina tempo
                 ; quando voltar da sub-rotina voltará
                 ; para a linha imediatamente depois
                 ; da CALL tempo
```

Flags afetados:
- Nenhum

Instrução:
clrf f ; comentários

Significado:
- Coloca 0 no registro f

Explicação:
- O registro f terá o valor 0, ou 0 0 0 0 0 0 0 0 b ou 00H. Esta instrução também fará com que o Flag Z fique em 1, ou seja, "setado".

Exemplo:
Se zeca = 1 1 1 1   1 1 1 1

```
clrf zeca        ; zero zeca
                 ; zeca fica igual a 0 0 0 0   0 0 0 0
```

Flags afetados:
Z –> do registrador STATUS, pois como o registrador fica todo em zero ele, o Z, vai para 1.
Z = 1.

Instrução:
clrw ; zera o registrador de trabalho W

Significado:
- O W terá o valor 0 e o flag Z passará a ser 1. Se você quiser ter certeza que W virou 0, teste o flag Z e veja se ele virou 1, pense como.

Exemplo:
Se W = 254

```
clrw    ; W era igual a 254, mas, após
        ; esta instrução é igual a 0
```

Flags afetados:
Z –> do STATUS Z = 1

Instrução:
```
clrwdt  ; zera o watch dog impedindo o
                ; reset do PIC
                ; esta função tem que estar
                ; dentro das loopings. Caso contrário,
                ; o watch dag estoura, reseta o PIC e
                ; seu programa não funciona.
```

Significado:
- reseta o, watch dog

Explicação:
- O watch dog, que é um timer independente de tudo, será zerado, impedindo o reset do PIC. Se o prescaler estiver direcionado ao watch dog este também será zerado.

Exemplo:
O watch dog conta.

```
clrwdt          ; reseta o watch dog ele zera e
                ; começa a contar novamente
```

Flags afetados:
TO\ e PD\ –> que estão no registrador STATUS são “setados” e virarão 1.

Instrução:
comf f, d ; comentários

Significado:
- faz o complemento dos bits do registrador f, ou seja, o inverte. Se eram:
0 0 0 0 1 1 1 1

Ficarão:
1 1 1 1 0 0 0 0

Explicação:
- Bit a bit será invertido e você pode usar esta função para fazer um seqüencial ou um painel que escreva duas mensagens, vá pensando.

Exemplo:

Se f = 0 0 0 0 0 0 0 0

```
comf   f          ; inverte/complementa f
                  ; que fica com o valor de 1 1 1 1   1 1 1 1
                  ; e este valor é salvo, fica nele mesmo
```
Mas.

```
comf   f, 0       ; inverte/complementa f
                  ; agora o valor de 1 1 1 1  1 1 1 1 será
                  ; salvo em W e F continua
                  ; com 0 0 0 0  0 0 0 0
                  ; veja a importância dof, 0
```

Flags afetados:
Z –> do STATUS
Se o resultado for 0, Z será igual a 1
Se o resultado for 1, Z será igual a 0
Só para variar.

Instrução:
decf f, d ; comentários

Significado:
- Diminui em uma unidade o valor armazenado em f, que é um registrador.

Explicação:
- O número que está em f será diminuído em uma unidade, se era 10 depois da instrução virou 9. Você pode usar esta instrução para fazer um detonador, brincadeirinha...

Exemplo:
```
movlw 45                ; carrega W com o valor 45
movwf      tempo1       ; carrega tempo1 com o valor 45
decf       tempo1       ; diminui 1 de tempo1 e agora
                        ; o valor dele é igual a 44
                        ; e W continua com 45
```

Mas,

```
movlw 45                ; já sabe né
movwf   tempo1          ; carrega tempo1 com 45

decf     tempo1, 0      ; decrementa 1 de tempo1
                        ; o valor 44 será salvo em W
                        ; o valor de tempo 1 será 45.
```

Flags afetados:
Z –> do STATUS
Se o resultado for ≠ de 0 –> Z = 0
Se o resultado for = a 0 –> Z = 1

Instrução:
decfsz f, d ; comentários

Significado:
- Diminui uma unidade o valor que está no registrador f e pula a próxima instrução se o resultado for zero.

Explicação:
- O valor de f (que deve ser entre 0 a 255) será diminuído em uma unidade (f – 1) e caso o resultado seja 0 a instrução seguinte será pulada. Esta instrução é muito utilizada em Loops de tempo.

Exemplo:
Se tempo2 = f = 100

```
inicio:                 ; porta para "chegar" Looping

decfsz   tempo 2        ; diminui 1 de tempo 2
goto inicio             ; caso o resultado seja 0
bsf    portb, 0         ; pula a instrução seguinte e não vai
                        ; para inicio. Mas, como resultado é
                        ; 99 lê a próxima instrução vai para
                        ; inicio e fuga dano voltinhas/Loopings
                        ; até zerar e acende o Led que esta na
                        ; portb, 0 o resultado da "decrementação"
                        ; é salvo no próprio tempo2.
```

Mas,

```
decfsz  tempo2, 0       ; diminui de tempo2 e salva em W
                        ; a maneira de usar em Loops de
                        ; tempo é a anterior
```

Flags afetados:
- Nenhum

Instrução:
goto k ; comentários

Significado:
- manda o programa para o endereço/nome simbolizado por k.

Explicação:
- Este desvio é incondicional, ou seja, não precisa de nenhuma condição ou teste. Ele diz vai e o programa vai.
- Alguns dizem que usar goto é falta de conhecimento, falta de conhecimento é a arrogância.
- Use com critério e use também outras ferramentas que ainda veremos

Exemplo:

```
goto  livro     ; vai para o livro
nop             ; e pula os nops
nop             ;

livro:          ;
```

Flags afetados:
- Nenhum

Instrução:
incf f, d; comentários

Significado:

- Soma 1 ao valor do registrador f (nunca pode passar de 255).

Explicação:
- Você somará 1 ao valor de f se não existir o d, o que corresponderá que ele é igual a 1, o resultado será salvo no próprio f. Se no lugar de d existir um 0 (f, 0) o resultado será salvo em W.

Exemplo:
```
movlw 55        ; carrega W com 55
movwf teste     ; carrega teste com 55 mas W
incf   teste    ; continua com 55
                ; diminui 1 de teste (55-1 = 54) e teste
                ; agora tem o valor de 54 pois d = 1
                ; e o resultado foi salvo em teste
```

Flags afetados:
Z –> do registrador STATUS

Instrução:
```
incfsz f, d     ; comentários
```

Significado:
- Soma 1 ao valor do registrador f e pula a próxima instrução se o resultado for 0.

Explicação:
- Soma 1 ao valor do registrador f e pula a instrução seguinte se o resultado for 0 (quando falamos "pula" a linha seguinte sempre estamos nos referindo à instrução seguinte). Agora você me pergunta: Mas, quando que eu vou somar 1 a alguma coisa e vai dar 0? Calma.

Exemplo:
```
Se f = dica e dica = 0
movlw 0         ; carrega W com 0
movwf dica      ; carrega dica com 0
incfsz dica, 0  ; soma 1 ao valor de dica (1 + 0 = 1)
                ; mas, salva em W e dica continua com 0
                ; portanto a próxima instrução é pulada
bsf portb, 1    ; esta instrução é pulada
```

Viu agora o "poder de d"? Ele faz esta instrução mais útil e funcional.

Flags afetados:
- Nenhum.

Instrução:
```
iorlw k   ; comentários
```

Significado:
- Executa um "OR" ou "OU" entre W e k.

Explicação:

- Um "OU" é uma operação lógica básica e vamos usar uma tabela da verdade e a simbologia usada em "Técnicas Digitais.

Quando a entrada A "OU" a entrada B, forem iguais a 1 a saída será 1. "Se tem 1 em uma entrada tem 1 na saída, não importa quantas entradas".
O resultado será salvo em W.

Exemplo:
Se W = 0 0 0 0 1 1 1 1 e k = 1 1 1 1 0 0 0 0
Lembre-se que k é um nome, vamos chamá-lo de livro.

```
iorlw livro     ; faz um OU entre W e livro
                ;   W = 0 0 0 0  1 1 1 1
                ; livro = 1 1 1 1   0 0 0 0
                ;           1 1 1 1   1 1 1 1
                ; e salva o resultado em W mas e se
                ; você quiser o resultado em livro veja
                ; as instruções seguintes
movwf livro     ; carrega livro com o valor de W
                ; está bem?
```

Flags afetados:
Z –> do registrador STATUS

Instrução:
iorwf f, d ; comentários

Significado:
- Executa um OU entre W e f.

Explicação:
- Como já vimos o OU é uma operação da lógica Booleana ou dos "Técnicas Digitais".
Esta operação irá fazer um OU em o valor que está em W e o valor que está em f o resultado será salvo, como sempre, dependendo de d. Se d = 0 o valor será salvo em W. Se d = 1 o valor será salvo em f. Lembre-se que a default, ou valor padrão para d, é sempre 1. Pois mais óbvio que pareça, se você não colocar 0 será 1.

Exemplo:

```
movlw B 0 0 0 0  0 0 1 0        ; carrega W com 2
movwf  regfteste                ; carrega regfteste com 2
movlw B 0 0 0 0  0 0 0 1        ; carrega W com 1. É 1 sim,
                                ; só que está em binário e
                                ; B 0 0 0 0  0 0 0 1 é o k.
iorwf    regfteste              ; Faz o OU de W com regfteste
                                ; e salva em regfteste. Veja que
                                ; eu até coloquei o 1 (mas não é
                                ; preciso)
                                ;          W = 0 0 0 0  0 0 0 1
                                ; regfteste = 0 0 0 0  0 0 1 0
                                ;             0 0 0 0  0 0 1 1
                                ; o resultado é 3 em regfteste
```

Flags afetados:
Z –> do STATUS e como a operação é diferente de 0 ele é igual a 0.

Instrução:
movlw k ; comentários

Significado:
- carrega W com o valor representado por k.

Explicação:
- W terá o valor de k e este X pode ser escrito em decimal, binário, hexadecimal.

em decimal 10 ou .10 (mais para frente)
em binário B 0 0 0 0 0 0 0 0 (mais para frente mais explicações)
em hexadecimal 10H (mais explicações à frente).

Exemplo:
Se W = 0 ou qualquer outro número

```
movlw 17        ; o valor de W é 17 agora
                ; simples e extremamente útil
                ; W tinha que ser um "trabalhador"
```

Flags afetados:
- Nenhum.

Instrução:
```
movf f, d       ; comentários
```

Significado:
- Move f para d

Explicação:
- Até que enfim d vai pegar pesado.
Vamos ver.
Se d = 0 ou d = W o valor é armazenado em W.
Se d = 1 o valor é armazenado em f.
Mas, que nada? É apenas uma forma de copar um valor de um registrador para W e manter este valor no registro. Mas, tem mais.

Exemplo:
Se f = spock = 15 e W = 0 ou outro número.

```
movf spock, W       ; move o valor de spock para W
                    ; e W fica igual à spock e W = 15
```
Poderia ser assim:

```
movf spock, 0       ; moveria spock para W também
                    ; sempre alteramos W
```

Mas também pode ser assim, e isto é que é importante, mas, assim como?
Vamos supor que você quer testar o Flag Z que indica se uma operação resultou em 0, mas, sempre mexer no valor que existe registrado W.
Veja:

```
movf spock      ; desta forma o valor de spock é copiado para o
                ; próprio spock e se spock for 0 então Z será
                ; 1 spock = 0 –> Z = 1 e se spock for ≠ 0 então Z será 0
                ; spock ≠ 0 –> Z = 0 sem colocar o registrador de
                ; trabalho W na jogada. Dia de folga dele.
```

Flags afetados:
Z –> do registrador STATUS. Este Flag só para relembrar, é o indicador de zeros.

Instrução:
movwf f ; comentários

Significado:
- O valor que está em W vai para f.

Explicação:
f é um registrador que tem um nome que você escolheu e definiu em uma área específica do cabeçalho de programa. Alguns chamam de variáveis. O importante é que este nome, associado ao f (minúsculo) tem que estar lá, senão o software dará erro na compilação e não funcionará.

Exemplo:
Se W = 217

```
movwf  jaca     ; o valor de jaca será 217, pois a instrução
                ; mandou fazer isto, esta instrução será
                ; muito usada em qualquer programa.
                ; Normalmente junto com MOVLW e são
                ; fáceis e úteis.
```

Flags afetados:
- Nenhum.

Agora que vimos um monte de **movxxs**, vamos fazer uma pausa para reflexão...

Quando você quer saber se um registrador está com o valor igual a 0 como você faz?

Ou você carrega este registrador no W e carrega o W em algum lugar que permita visualizar isto. Como ver se algum Led se acende na portb, mas, usa a instrução **movf**, não muda o valor de W e sabe se o registrador está com o valor 0 lendo o Flag Z.

Para isto serve **movf**.

Vamos ver exemplos destes dois modos:

Se o registrador dream = 0 e W = 251.

```
movlw dream     ; carrega W com o valor de dream que é 0 e o valor
                ; de W, que era 251, vai para o espaço.
movwf portb     ; o valor de W vai para o registrador portb que como
                ; tem o valor 0, não acenderá nenhum Led.
                ; Veja a seguir:
                ;
                ;          Fig. 26.3
                ;
                ;
                ;
                ; o registrador portb é 0 mas, desta forma mudamos
                ; o valor original de W.
```

Para não mudar.

Se dream = 0 e W = 251

```
movf dream      ; esta instrução vai carregar em dream o próprio valor de dream e
                ; como em um sonho, não afetará o valor de W. Você está vendo
                ; W junto às instruções? Não. Então o valor de W continua igual
                ; a 251 (W = 251), para saber se dream é igual à zero (0) basta
                ; verificar o Flag Z se ele for igual a 1 significa que dream é igual
                ; a 0. Z = 1 –> dream = 0 mas, este teste deve ser feito logo após
                ; a instrução MOVF.
```

Como fazer isto?

```
movf dream          ; carrega dream em dream
nop                 ; dá um tempo para ter certeza da operação
btfsc STATUS, 2     ; testa o bit2 (Z) do STATUS como ele é 1
                    ; lê a linha seguinte e
bsf portb, 0        ; acende o Led 0 na portb, sem mexer com W.
```

Esta é uma das formas. Vamos aprender a ler os valores dos registradores para ajudar nos programas.

Chega de reflexão.

Instrução:
nop ; comentários

Significado:
- Não faz nenhuma operação

Explicação:
- Durante o **nop** o microcontrolador fica um ciclo de máquina sem fazer nenhuma operação na CPU. O "tempo perdido" dependerá do clock do PIC. Você pode usar o **nop** para "arredondar" um Loop de tempo, para esperar entre uma instrução e outra (às vezes isto é útil e você verá nos programas, exemplos).

Exemplo:
```
nop     ; espera um ciclo de máquina
```

Flags afetados:
- Nenhum

Instrução:
retfie ; comentários

Significado:
- A função desta instrução é terminar com a interrupção em andamento.

Explicação:
Ela busca o endereço de retorno na pilha (STACK) e seta o flag GIE, que é o responsável pelas interrupções gerais (GIE = 1).
Embora não tenhamos falado nada semelhante até agora, esta instrução demora dois ciclos de máquina. Conversaremos mais disto depois.

Exemplo:

```
call  interrupção; chama a rotina de interrupção
                  ;
 - - - - - - -    ; tracinhos são linhas de programa
 - - - - - - -    ;
 - - - - - - -    ;
                  ;
                  ;
                  ;
                  ;
interrupção       ;
- - - - - - - -    ;
- - - - - - - -    ;
- - - - - - - -    ;
                  ;
retfie            ; volta para a linha seguinte
                  ; ao call e faz GIE = 1
```

Flags afetados:
- Já que GIE é um flag ele é afetado e ele está na INTCON.

Instrução:
retlw k ; comentários

Significado:
- Volta de uma sub-rotina com o valor de W = k.

Explicação:
- Além desta instrução chamar a de sub-rotina ela faz o valor de W igual ao valor de k. Para saber em qual linha voltar ela busca o endereço na pilha ou stack. Esta instrução gasta dois ciclos de máquina.

Exemplo:

```
1 – movlw 10          ; carrega W com 10
2 – nop               ; espera um ciclo de máquina
3 – call  saudade     ; vai para a sub-rotina
                      ; saudade
4 – bsf portb, 0  ; seta o bit zero da portb
                      ;
end                   ; indica o termino do programa
                      ;
5 – saudade:          ; indicador para a sub-rotina
6 – bsf portb, 1  ; seta o bit 1 da portb
7 – retlw 2           ; retorna da instrução mas, muda
                      ; o valor de W de 10 para 2.
```

Mas para que tenta coisa?
Para você ver que a instrução troca o valor de W e para você começar a ver as chamadas de sub-rotinas e as sub-rotinas.
O programa funcionará assim:
Linha 1 –> W é carregada com o valor 10.
Linha 2 –> espera um ciclo de máquina.
Linha 3 –>a instrução call manda o programa para linha 5.
Linha 5 –> ter o nome da sub-rotina.
Linha 6 –> acende o Led que está na portb, 1.

Linha 7 –> muda o valor de W para 2 e volta para a linha imediatamente após o call, que é a linha 4.
Linha 4 –> acende o Led que está na portb, 0.

Flags afetados:
- Nenhum.

Instrução:
return ; comentários

Significado:
Retorna da sub-rotina

Explicação:
- Com esta instrução você volta da sub-rotina, volta para a linha imediatamente após a chamada de sub-rotina, feita com a instrução **call.**
Esta instrução também demora 2 ciclos de máquina. É importante saber o quanto demora cada instrução quando fazemos temporizadores de precisão bem como quando fazemos geradores de vídeo ou de pulsos. Estamos falando do stack de novo.

Exemplo:

```
call lembrança  ; vai para a sub-rotina lembrança
nop             ; a volta da sub-rotina é nesta linha
                ; (linha com o nop)
                ;
lembrança       ; indica o início da sub-rotina lembrança
movlw 100       ; carrega W com 100
return          ; volta para linha imediatamente
                ; após o call que é a linha com o nop
```

Flags afetados:
- Nenhum.

Instrução:
rlf f, d ; comentários

Significado:
Caminha com os números para a esquerda.

Explicação:
- Para entendermos isto o melhor é ver tudo em binário. Vamos ver:
Antes da instrução você tem –> 0 0 0 0 0 0 0 0 e C = 1 instrução.
Depois da instrução você tem –> 0 0 0 0 0 0 0 1 e C = 0 instrução.
Depois da instrução você tem –> 0 0 0 0 0 0 1 0 e C = 0 instrução.
Depois da instrução você tem –> 0 0 0 0 0 1 0 0 e C = 0.
Você vêm caminhando com o bit 1 da direita para esquerda e passa pelo C.
Esquerda 0 ← 1 direita
Se o valor de d for igual a 0, como já sabemos o valor, será salvo em W, usando o default o valor será salvo em F. Esta instrução é útil para se fazer leds ou lâmpadas correrem de um lado para outro.
O flag C do registrador STATUS é importante nesta função. Para quem não se lembra o Carry/Borrow indica quando aconteceu um carry-out. Em outras palavras, quando um número 1

passa do bit 7 para o bit 8 (nona posição) que de uma forma simples, poderíamos dizer que é o flag C do registrador STATUS. Veja:
Antes da instrução –> 0 1 1 1 1 1 1 1 e C = 0 instrução.
Depois da instrução –> 1 1 1 1 1 1 1 0 e C = 0 instrução.
Depois a instrução –> 1 1 1 1 1 1 0 0 e C = 1 houve um carry-out.
Você pode aproveitar a leitura do flag C, que é o bit 0 do registrador STATUS para resetar o registrador ou fazer através da instrução **rrf**, os "1" voltarem.

Exemplo:

Se C = 0

```
movlw 1         ; carrega W com o valor 1
movwf anda      ; carrega o registrador anda o valor 1
rlf  anda       ; rotaciona ou anda com um 1 da direita para
                ; esquerda ← direita
                ; agora o valor de anda é igual a 10 que
                ; corresponde a 2.
rlf   anda      ; adiciona mais um 1, agora anda fica
                ; igual a 100 que corresponde a 4.
```

Flags afetados:
C – Carry/Borrow do registrador STATUS. Perceba que o bit 7 vai sendo "empurrado" para o flag C.

Instrução:
rrf   f, d ; comentários

Significado:
Caminha com os números para a direita.

Explicação:
Vamos ver em binário para entender melhor:
Antes da instrução –> 0 0 0 0 0 0 0 0 e C = 1 instrução
Depois da instrução –> 1 0 0 0 0 0 0 0 e C = 0 instrução
Depois da instrução –> 0 1 0 0 0 0 0 0 e C = 0
Você caminha da esquerda para a direita e passa pelo C.
Esquerda –> direita

Exemplo:

Se C = 0

```
movlw B' 1 1 1 1  1 1 1 1'      ; W assume o valor 1 1 1 1  1 1 1 1
movwf anda1                     ; carrega anda1 com 1 1 1 1  1 1 1 1 = 255
rrf anda1                       ; caminha com os números para a
                                ; direita passando pelo flag C
                                ; o valor de anda1 será agora igual a
                                ; 0 1 1 1  1 1 1 1 = 127 e C será igual a 1 (C = 1)
rrf anda1                       ; rotaciona para a direita um digito o valor
                                ; de anda1 será agora igual a 1 0 1 1  1 1 1 1 = 191
                                ; e C será igual a 1 (C = 1)
```

Flags afetados:

C – bit0 do registrador STATUS, perceba que o flag C (o valor que está nele) vai sendo "empurrado" para o bit7.

Outra pausa para meditação.

Estas duas instruções, muitas vezes não são usadas pois, o programador não as entende direito (isto é uma observação minha, feita por pura observação, na prática). Vamos mostrar elas de uma outra forma e descrever seus segredos.

Observação:

- Elas não caminham apenas com o número 1.
- Elas vão rotacionando ou trocando de lugar, um bit por vez, os bits de um registrador.
- Podemos usar o flag C para ajudar nos programas onde os usamos.

Vamos ver a estrutura da coisa.

rlf f, d –> rotaciona para a esquerda.

Quando aplicamos a instrução **rlf** no registrador da *Figura 26.5*, acontecerá o seguinte:
- O bit C, que tem o valor 0, vai enviar este valor para o bit 0 e assim todos serão empurrados para a esquerda e o bit C terá o valor 1, que era o número que estava no bit 7. Se C = 1 ocorreu um carry-out.

Vamos ver a estrutura da outra coisa.

rrf f, d –> rotaciona para a direita.

Quando aplicamos a instrução **rrf** no registrador da *Figura 26.6*, acontecerá o seguinte:
- O bit C, que tem o valor 1, vai enviar este valor para o bit 7 e assim todos serão empurrados para direita. O bit 7 assumirá o valor 1, que era de bit C, e o bit C assumirá o valor 0 que era de bit 0.

Chega de meditação, vamos continuar de onde paramos.

"Antes que eu me esqueça, estas são as instruções mais mal educadas do PIC, ficam empurrando os bits pra lá e pra cá".

Instrução:
sleep ; comentários

Significado:
- Vai dormir PIC e economizar energia.

Explicação:
- Nesta condição o conteúdo da memória fica salvo e o processamento para, fazendo o PIC ficar mais econômico. O oscilador também para e o Watch-dog e o prescaler são zerados.

Exemplo:

```
sleep  ; o PIC "dorme"
            ; para sair do modo SLEEP você pode fazer
            ; algumas coisas (usar um despertador):
            ; - reset pelo pino MCLR ou Reset
```

```
                ; - estouro do Watch-dog, mas, ele precisa
                ; estar habilitado
                ; - interrupção da EEPROM (Final de escrita)
                ; - interrupção no Rb0/INT
                ; interrupção através do Rb4 ao Rb7 da portb
                ; você pode perguntar como o Watch-dog consegue
                ; "estourar" se o oscilador está parado. É que, seu
                ; esquecidinho, o Watch-dog tem seu próprio oscilador.
                ; Se você resetar ele, através do pino MCLR ou Reset,
                ; o PIC começará a rodar do endereço 0 ou 00H em
                ; hexadecimal ou 0 0 0 0  0 0 0 0 em binário que é o
                ; início do programa.
```

Flags afetados:
- TO\e PD\ –> bits4 e 3 do STATUS
- TO\ será igual a 1 (TO\=1)
- PD\ será igual a 0 (PD\ = 0)

Instrução:

```
sublw   k       ; comentários
```

Significado:
- Faz uma subtração entre W e k (tem gente que chama k de Literal).

Explicação:
- O valor do registrador de trabalho W é subtraído do valor de k e o valor é salvo em W. Perceba que subtrair W de k é diferente que subtrair k de W.

- Subtrair W de k é igual a: k – W
- Subtrair k de W é igual a: W – k

É a primeira situação que está em ação no uso desta função.

Dá tudo no seguinte: Após a instrução W = k – W

k tem que ter um valor entre 0 a 255.

Podemos perceber que às vezes, podem acontecer resultados negativos e aí?

E aí, que quando o resultado der negativo devemos somar 256 ao valor de W para saber o seu valor real.

Exemplo:

```
movlw      10           ; carrega W com o valor 10
movwf  santicfy         ; carrega o registrador santicfy com 10
movlw       5           ; carrega W com 5
sublw    santicfy       ; faz a subtração de Santicfy e W
                        ; W = santicfy – W –> W = 10 - 5 = 5
                        ; o valor de W é igual a 5
                        ; isto faz com que o flag C fique igual
                        ; a 1 (C = 1)
```

Mas podia ser assim:

```
movlw   25              ; carrega W com 25 “em decimal”
movlw  yourself         ; carrega o registrador yourself com 25
movlw   27              ; carrega W com o valor 27
sublw    yourself       ; W = yourself – W –> W = 25 – 27 = -2
                        ; o valor de C é igual a 0 (C = 0) pois, o
                        ; resultado foi negativo.
                        ;Z = 0 pois, o resultado não foi zero.
                        ; o valor de W é: W = 256 + (-2) –> W = 254
```

Flags afetados:
- C, DC e Z –> do registrador STATUS

Instrução:

```
subwf  f, d         ; comentários
```

Significado:
- Subtrai o valor de W de f e salva o resultado em W se d = 0 ou em f se d = 1.

Explicação:
- O valor de W é subtraído do valor de f. Na instrução anterior o valor só podia ser salvo em W, mas, nesta, devido ao d, ele pode ser salvo em W ou f por isto escrevemos assim:

d = f – W –> pois d é quem define onde o valor será salvo.

Também podemos ter resultados negativos.
Quando o resultado for negativo C = 0.
Quando o resultado for positivo C = 1.

Caso o resultado seja negativo, basta somar 256 decimal ao resultado e você terá o valor real.
Se Z = 0 o resultado não foi zero.
Se Z = 1 o resultado foi zero.

Exemplo:

```
movlw  100      ; carrega W com 100
movwF ano       ; carrega o registrador ano com 100
                ; mas W continua com 100
movlw  100      ; carrega W com 101
subwf ano, 0    ; subtrai W de ano e salva em W (,0)
                ; como d = 1 fica:
                ; W = ano – W –> W = 100 – 101 = -1
                ; Z = 0 pois o resultado não foi zero
                ; C = 0 pois o resultado foi negativo
                ; o valor de W é: W = 256 + (-1) = 255
```

Mas, podia ser assim:

```
movlw   100     ; carrega W com 100
movlw   novo    ; carrega o registrador novo com 100
                ; mas, W continua com 100
movlw   99      ; carrega W com 99
subwf  novo     ; subtrai W de novo e salva em novo
                ; novo = novo – W –> novo  100 – 99 = 1
                ; temos então: C = 1 pois, o resultado foi
                ; positivo e Z = 0 pois, o resultado não foi zero.
```

Flags afetados:
- C, DC, Z –> do registrador STATUS

Instrução:
swapf f, d ; comentários

Significado:
- Inverte os bits de 0 a 3 com os de 4 a 7.

Explicação:
- Esta função troca nibles, ou seja, troca 4 bits.

Vamos ver isto em binário:

Antes da instrução –> 0 0 0 0 1 1 1 1 instrução
Depois da instrução –> 1 1 1 1 0 0 0 0

O destino onde será salvo este valor depende de d, só para variar.

Exemplo:

```
movlw  B' 0 0 0 0  1 1 1 1'      ; carrega W com 0 0 0 0 1 1 1 1
                                 ; que é um valor em binário
movwf  troca                     ; carrega o valor 0 0 0 0  1 1 1 1 em troca
swapf troca                      ; inverte os nibles, o valor de troca fica
                                 ; 1 1 1 1  0 0 0 0 o valor será salvo erm troca.
```

Mas, pode ser assim:

```
movlw  15                        ; carrega W com 15 que é um valor em decimal
movwf  muda                      ; carrega o valor 15 no registrador muda
swapf   muda                     ; inverte ao nibles, muda fica com o valor de
                                 ; 240 em decimal.
```

Só por curiosidade, 0 0 0 0 1 1 1 1 = 15 e 1 1 1 1 0 0 0 0 = 240
Flags afetados:
- Nenhum

Instrução:
xorwf f, d ; comentários

Significado:
- executa a função OU exclusivo entre W e f.

Explicação:
- A função OU exclusivo é uma função lógica, ou uma função da aritmética Booleana. Bonito não é mesmo. Mas, na prática trata-se da comparação de níveis lógicos na entrada de um dispositivo (normalmente chamada de porta lógica) e o resultado em sua saída. Algo similar à função E ou OU.

Só quando as duas entradas forem diferentes é que a saída será igual a 1. Este negocio de "exclusividade" é coisa de...

O resultado será salvo em um local definido por d.Para ver tudo isto melhor é útil recorrer ao binário. Esta função é feita bit a bit.

Exemplo:

```
movlw  B' 0 0 1 1 0 0 1 1'      ; carrega W com o valor 0 0 1 1 0 0 1 1 em binário
movwf   radio                   ; carrega radio com o valor de W
movlw  B' 1 1 0 0 1 1 0 0       ; carrega W com o valor 1 1 0 0 1 1 0 0
xorwf  radio                    ; faz a função lógica OU exclusivo entre W e radio
                                ;
                                ;    W = 1 1 0 0 1 1 0 0
                                ; radio = 0 0 1 1 0 0 1 1
                                ;          1 1 1 1 1 1 1 1
                                ; o resultado será tudo 1 (1 1 1 1 1 1 1 1)
                                ; Z = 0
```

Flags afetados:
- Z –> do registrado STATUS

Instrução:

```
xorlw   k          ; comentários
```

Significado:
- Faz um OU exclusivo entre W e k.

Explicação:

- Esta função faz uma operação bit a bit entre W e k.
A operação lógica é um OU exclusivo ou XOR.
- Bits diferentes têm como resultado 1.
- Bits iguais têm como resultado 0.

O valor será salvo em W. E se você usar esta instrução com o próprio W o resultado é zero. Você pode usá-la para comparar a combinação de uma entrada/porta com o valor de W e quando der 0 acionar alguma coisa. A função anterior também é útil para isto. “Combinação de cofre”.
Vamos usar o binário.

Exemplo:

```
movlw  B' 1 0 1 0 1 0 1 0'      ; carrega W com 170 (é o mesmo valor em decimal)
xorlw  W                        ; Faz um ou exclusivo de W com W.
                                ; W = 1 0 1 0 1 0 1 0
                                ; W = 1 0 1 0 1 0 1 0
                                ;       0 0 0 0 0 0 0 0
                                ; o valor será salvo em W.
```

Mas, pode ser assim:

```
movlw B' 1 1 1 1 0 0 0 0'       ; carrega W com 1 1 1 1 0 0 0 0
movf  uva                       ; carrega uva com 1 1 1 1 0 0 0 0
movlw B' 0 1 1 1 0 0 0 1'       ; carrega W com 0 1 1 1 0 0 0 1
xorlw  uva                      ; Faz o ou exclusivo com uva e salva em W
                                ;  W = 0 1 1 1 0 0 0 1
                                ; uva = 1 1 1 1 0 0 0 0
```

```
;          1 0 0 0 0 0 0 1
; o valor de W será 1 0 0 0 0 0 0 1
```

Flags afetados:
- Z –> do registrador STATUS

Existem mais duas instruções que podem não funcionarem com todos os microcontroladores PIC e por isto não serão comentadas aqui. O próprio fabricante dos PICs, a Microchip™ não recomenda o uso delas.

Eles são:

OPTION -> faz registro OPTION = W;
TRIS –> faz o registro TRIS = W.

26.1. Operadores do MPASM:

O MPASM é o software compilador que está incluído no MPLAB. Compilar em linguagem simples, que dizer transformar os mennônicos do assembly, ou instruções do assembly, no famoso código de máquina ou no português claro, um monte de zeros e uns.

Os operadores são comandos reconhecidos pelo MPASM e que podem ser usados nos programas, um muito usado neste livro é o $.

Vamos falar dele, especialmente, mas, antes vamos mostrar outros.

Operadores matemáticos:
+ –>adição
- –> subtração
/ –> divisão
* –> multiplicação
% –> restante de uma divisão

Operadores relacionais:
= –> igual
! –> diferente de
> –> maior
< –> menor
> = –> maior ou igual
< = –> menor ou igual

Operadores lógicos:
~ –> complemento
<< –> rotação para a esquerda
>> –> rotação para a direita
&& –> and (função lógica é óbvio)
| –> OU bit a bit
^ –> ou exclusivo
& –> E ou and, bit a bit
|| –> OU ou or

Operadores de atribuição:
= –> igual

Poderíamos falar mais sobre isto, mas, vamos ver como usar o $ e deixar que você aprenda a usá-lo e faça “experiências”. Não se esqueça da homepage que pode ajudá-lo.

Para aprender a usar o $ o ideal é ver exemplos e depois defini-lo. Daí você compara uma coisa com a outra e tem suas conclusões.
Normalmente usamos o operador $ junto com a instrução goto, com um número e com os operadores matemáticos + ou -.

Exemplo:

```
btfsc portb, 0    ; testa se o bit 0 da portb é igual a 0.
                  ; Se for zero pula a próxima linha
goto $ - 1        ; a instrução goto junto com os operadores
                  ; S e (-) e o número 1 acontece o seguinte:
                  ; se o bit 0 da portb não for zero a linha de
                  ; goto será linha e o goto irá mandar o programa
                  ; para $ - 1. Mas onde é $ - 1 você me pergunta e
                  ; eu respondo: $ - 1 é a posição atual do PC
                  ; (program counter) menos (-) uma (1) linha,
                  ; ou seja, na linha que tem a instrução btFsc.
                  ; Enquanto o bit 0 da portb não for zero o
                  ; programa ficará em "Looping" entre estas duas linhas.
```

Traduzindo, o operador $ manda o programa para a posição atual do PC (program counter ou contador das linhas do programa). Se colocarmos junto um número mais um sinal de (+) ou (-) acontecerá o seguinte:

goto $ -1, volta para a linha anterior;
goto $ - 2, volta duas linhas;
goto $ - 3, volta três linhas e assim adiante;
goto $ + 1, vai uma linha para frente ou visualmente falando, uma linha para baixo;
goto $ + 2, vai duas linhas para frente e assim adiante.

Com este operador podemos enviar um programa para uma determinada posição sem indicadores. O $ faz o que faz, pois, pega a posição atual do PC, que é a linha em que ele está, e subtrai ou soma linhas de acordo com os operadores e números que estão junto com ele. Teste os outros operadores.

Diretivas ou Diretrizes do MPASM:

Com estes "comandos" que não são instruções dos PICs, mas, que são entendidas pelo MPASM você conseguirá fazer um programa mais eficaz e mais rapidamente. Muitos destas diretivas são utilizados nos cabeçalhos dos programas.

Vamos estudá-los em ordem alfabética:

Diretiva:
_BADRAM –> define endereços não válidos na memória RAM do PIC.

Uso:

```
_badram D' 10' – D' 30' ; os endereços de 10 a 30 são inválidos.
                        ; O D maiúsculo indica que o endereço
                        ; está em decimal.
```

Explicação:

Na verdade esta diretiva nem precisa ser usada pois, o PIC já tem definidos os endereços inválidos (veja mapa ou banco de memória), mas, caso você "queira" usar, antes dela deve ver outra diretiva. Daí fica assim:

```
_maxram D' 100'              ; "máximo endereçamento"
_badram D' 10' – D' 30' ;
```

Diretiva:
BANKISEL –> gera um código de acesso que permite o acesso indireto ao banco de memória. Nos PICs que estamos estudando o responsável por isto é o flag IRP do STATUS.

Diretiva:
BANKSEL –> gera um código para acesso indireto aos bancos de memória. Com esta diretriz ajustamos automaticamente os bits RP0 e RP1 do STATUS para ir para o banco de memória onde está o registrador que buscamos.

Uso:

```
banksel  intcon      ; ajusta RP0 e RP1 para irmos para o banco
                     ; de memória onde está o intcon.
```

Diretiva:
CBLOCK –> esta diretriz define um bloco de variáveis e/ou constantes. Ela é muito usada nos programas mostrados neste livro e facilita bastante à confecção do cabeçalho do programa. Vem sempre acompanhada da diretriz ENDC.

Uso:

```
cblock 0 x 20        ; define endereço 20 em hexadecimal como
                     ; início dos variáveis
                     ;
tempo                ;
tempo1               ;
tempo bom            ; variáveis
beleza               ;
legal                ;
endc                 ; define o final dos variáveis
```

Diretiva:
CODE –> permite que partes de programas sejam ligadas a partes de outros programas para formar uma "coisa" maior.

Diretiva:
Duas under lines
__CONFIG –> define os fusíveis dos PICs ou os bits de configuração. Esta diretiva é muito utilizada (eu aconselho você a usá-la em todo programa). Ela fica no cabeçalho. Caso você não coloque esta linha de configuração, deverá selecionar estas opções no gravador do PIC. No nosso caso no gravador da Exsto, isto é configurado no software de gravação.

Veja as definições dos PICs usados neste livro:

_boden_on –> se ligado o PIC resetará se a tensão de alimentação cair para menos do que 4V durante 100µs.

_boden_off –> se desligado o PIC não reseta se a tensão cair para menos de 4V. Mas, você deve saber se o seu projeto permite isto.

_cp_all –> protege toda a memória evitando-se cópia. Dá para regravar mas não copiar ou ler o que está gravado.

_cp_75 –> protege parte da memória.

_cp_50 –> protege parte da memória.

_cp_off –> código de proteção desligado. Alguém pode copiar o seu programa.

_cp_on –> protege a memória toda contra cópia.

_pwrte_off –> desabilita o reset interno do PIC. Use esta opção se estiver usando um circuito de reset externo.

_pwrte_on –> habilita o reset interno do PIC, ou seja, o PIC só começará a funcionar depois de 72ms, após você colocar Vcc no MCLR. Você pode ligar o Vcc com o MCLR.

_wdt_on –> liga o watch-dog. Se esta opção estiver ligada você deve resetar o watch-dog durante o programa.

_wdt_off –> desliga o watch-dog. Não use esta opção se o seu projeto inclui a segurança física das pessoas.

_lvp_on –> permite a programação em baixa tensão (5V). Deve ficar desabilitada com a maioria das placas de gravação. Usada em ICSP.

_lvp_off –> a gravação deve ser feita com tensão de 13V.

_mclre_on –> permite o reset externo através da colocação de 0 volts no pino MCLR. Deve ficar habilitado.

_mclre_off –> não permite o reset ou “clear” externo.

_er_osc_clkout –> usado quando o oscilador externo é composto por uma constante RC e temos a saída de clock no pino Ra6.

_er_osc_noclkout –> oscilador RC sem saída de clock.

_intrc_osc_clkout –> oscilador interno de 4 MHz com saída de clock pelo Ra6 ou porta,6 (pode trabalhar em 37 kHz).

_intrc_osc_noclkout –> oscilador interno de 4 MHz sem oscilador externo. Ra6 vira uma I/O (pode trabalhar em 37 kHz).

_extclk_osc –> usado com clock externo, entrando no pino porta,7 (Ra7) e sem saída na porta,6. O Ra6 será uma I/O (entrada/saída).

_lp_osc –> usada para oscilar o cristal de baixa freqüência. Cristais abaixo de 200 KHz. Você sabia que existem cristais de 15 kHz?

_xt_osc –> para cristal ou ressonador entre 100 kHz a 4 MHz.

_hs_osc –> para oscilador com ressonador ou cristal acima de 4 MHz.

Normalmente estas definições são escritas uma após a outra, com o uso do operador &, e colocadas no cabeçalho do programa.

Usando osciladores internos podemos trabalhar em 4 MHz ou 37 KHz. Selecionamos isto no registrador PCON através do flag OSCF.

Se OSCF = 0 –> clock de 4 MHz.
Se OSCF = 1 –> clock de 37 KHz.

Estes valores são aproximados e tem uma tolerância.

Exemplo de linha de __config:

__config _boden_on & cp_on & pwrte_on & wdt_off & lvp_off & mclre_on & _xt_osc

Diretiva:
CONSTANT –> define um valor constante para uma palavra.

Uso:
Constante meia dúzia = 6

Diretiva:
#DEFINE –> define uma palavra ou pequeno texto que será associado há uma função em assembler. Desta forma colocamos no programa o texto ou palavra ao invés da instrução, mas, toda vez que a palavra for encontrada no programa, as instruções associadas serão executadas.

Uso:

```
#define  botao  porta, 2                ; toda vez que botao for encontrado
                                        ; estaremos falando do porta, 2

#define led_aceso bsf portb, 1          ; toda vez que encontrarmos
                                        ; led_aceso, a instrução bsf
                                        ; será executada e acenderá o
                                        ; led na portb, 1
```

Explicação:
Não tenho muito que explicar, mas, posso garantir que esta diretriz é muito útil.

Diretiva:
DA –> armazena conectores no rom ou Flash.

Diretiva:
DATA –> preenche a memória com um valor definida por um texto ou palavra.

Diretiva:
DB –> vai encher a memória de programa byte após byte.

Diretiva:
DE –> inicializa os dados da EEPROM interna durante a gravação.

Diretiva:
DT –> cria uma tabela na memória de programa.

Diretiva:
DW –> coloca expressões na memória de programa uma após a outra.

Diretiva:
ELSE –> usada junto com IF.

Diretiva:
END –> esta diretiva deve estar no final de todo programa para fazer com que o compilador pare de “montar” o programa.

Uso:
```
movlw   10      ; carrega W com 10
                ;
end             ; termina o programa
```

ou

```
goto y          ; vai para y
                ;
x:              ; índice para instrução goto
                ;
end             ; final do programa
                ;
                ;
y:              ; índice para goto
goto x          ; vai para x
```

O final do programa não está sempre no final do que está escrito. Fique atento a isto.

Diretiva:
ENDC –> é usada junto com CBLOCK. Define o final de um bloco de constantes.

Uso:
```
cblock          ; início dos constantes
                ;
                ;
tempo           ;
tempo 1         ;
tempo bom       ;
tempo 3         ;
                ;
endc            ; final das constantes
```

Diretiva:
ENDIF –> indica o final de um teste condicional. Deve ficar no final dos testes. Usados com IF.

Diretiva:
ENDM –> finaliza uma macro.

Diretiva:
ENDW –> é usado para definir o final de um bloco de repetição que também pode ser chamado de Loop. Normalmente usamos ela junto com a diretiva WHILE.

Diretiva:
EQU –> associa um nome há um endereço na memória.

Uso:
```
tempo1  equ   0X0D      ; o valor de tempo1 será salvo no
```

; endereço 0X0D em hexadecimal

Diretiva:
ERROR –> cria uma mensagem de erro para o usuário programador.

Diretiva:
ERRORLEVEL –>.9 define o tipo de mensagem de erro.

Capítulo 27
POR QUE USAR O MPLAB 5.7.40?

Vamos utilizar esta versão do MPLAB, pois ela é mais leve e roda em máquinas mais simples mais rapidamente. Esta é o principal motivo para usar esta versão deste ambiente de desenvolvimento integrado. Mas, veja se ele tem o PIC 16F84 e os PICs 16F627, 16F627A, 16F628, 16F628A e também 16F877 e 16F877A (nosso futuro).
Nada impede que você use e ou até aconselho, outras versões da MPLAB. Você pode instalar mais de uma, basta na instalação escolher um caminho diferente para cada uma.
Eu tenho em minha máquina, duas versões.

Capítulo 28
SUB-ROTINAS DE TEMPO (A VERDADE POR TRÁS DAS LOOPINGS)

Existem algumas maneiras de se calcular o tempo de loopings, algumas mais precisas ou menos precisas, mas, tudo depende da sua aplicação.
Uma das formas mais simples e imprecisas, mas, que lhe dará uma noção rápida do tempo de um looping, de um programa pronto é pegar o número de µs gasto no looping principal, multiplicar pelo número deste looping (N) e multiplicar este valor pelo número dos outros loopings (N1, N2, etc). Observe que em uma sub-rotina de tempo os loopings estão “um dentro de outro” e interagem um com o outro.

Baixe a tabela sobre delay

Este capítulo começou a ser escrito em 1998, mas, teve uma “parte” em especial no ano de 2004, quando eu e meu ex-supervisor, o Ivon Luiz, quase saímos no braço para chegarmos à conclusão sobre os loops de tempo. Finalmente eu me dei por vencido e adotei algumas idéias dele.
“Faz bem aprender com os outros”.
Independente disto, o MPLAB vai oferecer ferramentas para você criar uma rotina de tempo super preciso, como o stopwatch.

Observações:
- O clock usado como referência neste capítulo é de 4 MHz.
- As instruções têm a duração de 1 ou 2 ciclos de máquinas.
- 1 ciclo de máquina = 1µs.
- 2 ciclos de memória = 2µs.
- Se usar clock diferente, os períodos das instruções mudam e você precisa levar isto em consideração nas equações.
- Isto foi o mais próximo do loop perfeito que chegamos.

Capítulo 29
EEPROM/E2PROM É O BICHO

Agora vamos ver como escrever e ler a memória EEPROM que está dentro do PIC. Usamos esta memória para, por exemplo, gravarmos uma senha e salvá-la e mesmo desligando o PIC, ser capaz de utilizá-la novamente, ou seja, ela não se perderá.
Usando a EEPROM podemos entrar com a senha que temos e gravar uma senha nova e nunca perdê-la. Sem a EEPROM isto é muito difícil...

Primeiro com o 16F84:

Escrevendo na EEPROM interna:

```
bcf  status, rp0        ; seleciona o banco 0 de memória
movlw     5             ; carrega W com 5
movwf   eeadr           ; carrega o registrador EEADR com o número 5.
                        ; Ele está no banco 0 por isto fomos para lá.
movlw   25              ; carrega W com o número 25, este será o valor
                        ; gravado e 5 será o endereço onde é gravado
                        ; este valor.
movwf  eedata           ; carrega o registrador EEDATA com o valor 25
                        ;
bcf  intcon, gie        ; desabilita as interrupções para que o PIC não
                        ; seja resetado devido a EEPROM
bss status, rp0         ; volta para o banco 1 da memória, para
                        ; poder ir para o registrador EECON1 que
                        ; permitirá gravar na EEPROM
bsf eecon1, wren        ; habilita escrita na EEPROM
movlw   85              ; carrega 85, em decimal, em W *
movwf  eecon2 x         ; carrega eecon2 com 85 *
movlw   170             ; carrega 170, em decimal, em W *
movwf  eecon2           ; carrega eecon2 com 170 *
bsf  eecon1, wrc        ; coloca 1 no flag wr do registrador eecon1
                        ; e começa a gravar.
btfsc  eecon1, wr       ; testa se ofFlag wr está em 0. Se estiver
                        ; acabou a gravação e pula a linha seguinte.
                        ; Se estiver em 1 não acabou e lê a linha
                        ; seguinte.
goto $ - 1              ; volta para a linha anterior e fica neste loop
                        ;até que wr seja zero e acabou a escrita.
bcf  eecon1, eeif       ; zera o bit EEIF evitando interrupção.
                        ;
bcf  eecon1, wren       ; desabilita a escrita na EEPROM
bcf  status, rp0        ; vai para o banco 0 da memória
bsf  intcon, gie        ; habilita as interrupções.
```

* Estas quatro linhas e/ou quatro instruções são uma seqüência usada como segurança, de forma que não ocorram gravações por acaso na EEPROM. É necessário se carregar o registrador EECON2 com 85 em decimal ou 55H em hexadecimal depois 170 em decimal ou AAH em hexadecimal antes de qualquer tentativa de escrita.

Este recurso foi desenvolvido pela própria Microchip, que é o fabricante, de forma a evitar gravações não desejadas. Toda vez que você precisar gravar na EEPROM interna precisa usar esta seqüência que funciona então como um tipo de código ou senha de permissão para gravar.
Você pode perguntar como podem acontecer gravações indesejadas e eu lhe respondo com um exemplo prático de diversos equipamentos que usam EEPROM. A resposta é: picos de tensão na alimentação.

Lendo a EEPROM interna:

```
bcf  status, rp0          ; seleciona o banco 0 para poder
                          ; "achar" o registrador EEADR
movlw  5                  ; carrega W com 5, que irá ser
                          ; o endereço
movwf  eeadr              ; carrega o registrador EEADR
                          ; com o número 5 (endereço 5)
bsf  status, rp0          ; volta para a banco 1 para "achar"
                          ; o registrador EECON1
bsf  eecon1, rd           ; lê o registrador. Isto é feito colocando-se
                          ; 1 no flag rd do registrador EECON1
bcf  status, rp0          ; volta para o banco 0 para poder "ver"
                          ; o registrador EEDATA
movf  eedata, w (ou 0)    ; faz o valor de W igual a EEDATA que
                          ; corresponde a 25 em nosso exemplo.
movwf  dadoseeprom        ; salva o valor de W, que é o valor do
                          ; endereço 5 da EEPROM no registrador
                          ; dadoseeprom
```

Agora o 16F628/16F628A:

Escrevendo na E2PROM interna:

```
bsf  status, rp0          ; seleciona o banco 1. Neste PIC temos
                          ; todos os registradores que precisamos
                          ; (espelhados).
movlw   5                 ; carrega W com 5
movwf  eeadr              ; carrega o endereço 5 no registrador EEADR
movlw  25                 ; carrega W com 25
movwf  eedata             ; carrega a data 25 no registrador EEDATA
bcf intcon, gie           ; desliga as interrupções
bsf  eecon1, wren         ; permite escrita no EEPROM
movlw  85                 ; carrega W com 85
movwf  eecon2             ; carrega eecon2 com 85
movlw  170                ; carrega W com 170
movwf  eecon2             ; carrega eecon2 com 170. Estas 4 últimas
                          ; instruções são uma "senha" de proteção
                          ; contra gravações indesejadas
bsf  eecon1, wrv          ; começa a escrita
btfsc  eecon1, wr         ; enquanto wr for 1 aguarda a escrita
goto $ - 1                ; faz o loop enquanto não termina a escrita
bcf  eecon1, wren         ; não permite mais a escrita
bsf intcon, gie           ; liga as interrupções
bcf  status, rp0          ; volta para o banco 0 que é o "banco de
                          ; trabalho" do PIC
```

Lendo o 16F628/16F628A:

```
bsf status, rp0            ; banco 1
movlw 5                    ; W = 5
movwf  eeadr               ; vai para endereço 5
bsf eecon1, rd             ; lê valor no endereço 5
movf  eedata, w (ou 0)     ; W = eedata = dados = 25
bcf status, rp0            ; volta banco 0
movlw  dadoseeprom         ; salva leitura em dadoseeprom
```

32.1 Registradores usados com a EEPROM:

São quatro os registradores usados com a EEPROM interna dos PICs.

Eles são:

EEADR – responsável pelo endereçamento da memória EEPROM. No PIC 16F84 teremos apenas 64 posições de memória ou de 0 a 63, em decimal. No PIC 16F62X teremos 128 posições ou bytes de memória ou de 0 a 127, em decimal.

EEDATA – é através deste registrador que trabalhamos com os dados que queremos gravar ou ler da EEPROM. O dado é colocado neste registrador e o endereço é colocado na EEADR.

EECON1 – este registrador é conhecido como registrador de controle 1. No PIC 16F84 este registrador tem cinco flags ou bits usados e no PIC 16F62X são apenas quatro flags ou bits.

EECON2 – este registrador não tem nenhum ajuste que passa ser feito pelo programador e/ou técnico. Ele é utilizado para evitar que ocorram escritas não desejadas. Vimos o seu uso nos exemplos anteriores.

L/E = permite leitura e escrita (R/W)

Os bits 7, bit 6 e bit 5 “não existem”, e devem ser lidos como 0 (zero).

Bit 4 – EEIF –> determina a interrupção do final da escrita.
Bit 4 = 0 = ainda não acabou a escrita.
Bit 4 = 1 = já acabou de escrever e é preciso resetar ele(tornar 0) pelo software.

Bit 3 – WRERR –> este bit indica erro na escrita da EEPROM.
Bit 3 = 0 = a escrita foi terminada com sucesso.
Bit 3 = 1 = a escrita não foi feita corretamente.

Bit 2 – WREN –> este flag habilita a escrita na EEPROM.
Bit 2 = 0 = escrita na EEPROM não é permitida.
Bit 2 = 1 = permite que se escreva na EEPROM.

Bit 1 – WR –> este flag define o início da escrita. Ele aciona a escrita.
Bit 1 = 0 = indica que a escrita terminou.
Bit 1 = 1 = colocando-o em um (1) a escrita começa e fica em 1 até que ela termina. Quando a escrita terminar o PIC coloca este bit ou flag em 0 e podemos usar isto para confirmar a escrita. Só dá para você colocar 1 neste flag, o 0 é por conta do PIC.

Bit 0 – RD –> este bit define o início da leitura da EEPROM.
Bit 0 = 0 = não começa a ler a EEPROM.

Bit 0 = 1 = inicia a leitura da EEPROM e fica em 1 até terminar a leitura. Quando terminar a leitura o PIC o coloca em 0 e isto pode ser usado para se saber quando aconteceu o final da leitura. Só dá para colocar 1 neste flag o 0 é "função" do PIC.

32.2. A EECON1 do 16F62X (16F627, 16F628, 18F628A):

L/R –> permite escrita e leitura pelo usuário (R/W).

Todos os flags funcionam como o da 16F84, menos o bit 4, que não existe neste registrador. Observe os software dos dois tipos de PICs e veja as diferenças.

Podemos perceber que os registradores EEADR, EEDATA, EECON2 são praticamente, "transparentes" para você. Apenas o EECON1 permite mais "interação".

Informações adicionais 1
PIC COM "A" E SEM "A"

Até onde eu sei a diferença entre um PIC com final A e sem final A está na estrutura interna. Os dois têm as mesmas funções e um pode substituir o outro. Ao menos foi isto que verifiquei na prática e que constatei na conversa com amigos experts em PIC.
É como uma UPGRADE no PIC. Que me perdoem os catedráticos mas, é assim que enxergo.
Mas, isto não quer dizer que você não precise setar o seu gravador, são necessárias algumas "coisinhas" para que um funcione no local do outro.
1º –> Colocar o arquivo include correspondente ao PIC usado;
2º –> Mudar o PIC no programa gravador;
3º –> Ver se o seu gravador (hardware-plaquinha) é capaz de gravar os PICs com ou sem A no final.

Informações adicionais 2
VARREDURA EM DISPLAYS DE 7 SEGMENTOS - DICAS

-Faça uma tabela de variáveis onde você associe as letras que serão escritas ao código binário que acenderá cada segmento dos displays.

- Crie uma rotina de tempo, com freqüência entre 30 a 100 Hz. Esta rotina será usada para fazer a varredura dos displays.

- Supondo que você esteja usando dois displays, tenha claro o seguinte: para cada letra que acrescentar nos displays deverá fazer com que ela seja impressa umas 20 a 30 vezes. Com isto quero dizer o seguinte:

Se você escrever CA nos displays, isto deverá sempre ser impresso pela rotina de varredura dos displays entre 20 a 30 vezes. Chamo isto de rotina de deslocamento.
Supondo que a palavra completa seja CASA o próximo passo é deslocado uma letra, assim o que aparecerá nos displays será AS.

Para fazer isto associe ao final dos 20 a 30 impressões os códigos binários nas saídas para os displays, que deverão ser trocados, entrar na rotina de varredura, repetidamente 20 a 30 vezes, assim sucessivamente.

Para que o efeito de varredura fique bom é necessário que a cada duas letras exista um loop que chame a varredura umas 20 a 30 vezes.

Exemplo:

Palavra usada = ELA

Início:

```
        movlw E         ; escreve E registrador W
        movwf display   ; escreve E no display
        call se         ; chama rotina se (varredura de 30 a 100 Hz)
        movlw L         ; carrega L no registrador W
        movwf display   ; escreve L no display
        call se         ; chama rotina se (varredura de 30 a 100 Hz)
        decfsz xxx      ; decrementa o valor entre 20 a 30 vezes
        goto início     ; volta para o início
        call see        ; retorna do deslocamento (rotina de 20 a 30 vezes)

        . . . próximas letras.
```

A rotina de varredura se chama **"se"** e faz os displays ficarem piscando de 30 a 100 vezes por segundo (30 a 100 Hz).
A rotina de deslocamento se chama **"see"** e faz com que as letras sejam impressas 20 a 30 vezes.

Informações adicionais 1
DISPLAYS LCD BASEADOS NO HD 44780

Baixe o arquivo picdisp

O circuito integrado HD 44780 é muito comum no uso em displays de cristal liquido que trabalham com linhas e caracteres, por exemplo, 1 x 16 ou 2 x 16. Este números querem dizer o seguinte:

1 x 16 -> 1 linha de 16 caracteres.
2 x 16 -> 2 linhas de 16 caracteres.

A função deste CI é fazer a comunicação entre o microcontrolador e os outros CIs usados no display. Antigamente este CI era um VLSI montado e soldado do lado de trás do display e você podia ver o código dele, mais antigamente ainda ele tinha o formato de um CI tipo DIL. Hoje em dia ele vem encapsulado em uma resina e não é possível a leitura de código nenhum. São os avançados tecnológicos.

Pinagem destes displays:

Pino 1 –> terra.
Pino 2 –> Vcc (5 Volts).
Pino 3 –> ajuste de contraste (Vee).
Pino 4 –> RS (entrada) 0 = entrada de inscrição 1 = entrada de dados.
Pino 5 –> R/W (entrada) 0 = escreve no display, 1 = lê o display.
Pino 6 –> E (entrada) habilita leitura, deve estar em 0 e depois de colocado os dados, passar para 1 e voltar para 0. Após isto temporizar 1ms.
Pino 7 –> DB 0 - I/O.
Pino 8 –> DB 1 – I/O.
Pino 9 –> DB 2 – I/O.
Pino 10 –> DB 3 – I/O.
Pino 11 –> DB 4 – I/O.
Pino 12 –> DB 5 – I/O.
Pino 13 –> DB 6 – I/O.
Pino 14 –> DB 7 – I/O.

Por I/O, entenda pinos de entrada ou saída. Isto mesmo. Normalmente entramos com informações em um display mas também podemos ler as informações que estão escritas neles.
Quando um display LCD possui back-light ou luz de fundo, quer dizer que é possível acender uma luz no fundo do display.
Normalmente os “pontos” que acionam esta luz são indicadas por “A” e “K” ou “A” e “C”.

“A” de anodo ou positivo.
“K” ou “C” de catodo ou negativo.

Para que passamos usar um display é necessário fazermos a inicialização dele. Fazer a inicialização é enviar uma série de comandos (bits de 0 e 1) para que o display comece a entender o microcontrolador. Esta inicialização pode ser feita com 8 ou 4 bits e é um padrão para todos os displays baseados no HD 44780 (Este CI virou quase um padrão para displays).
Eu costumo trabalhar com inicialização em 8 bits (na contramão da história), mas, teremos acesso a inicialização com 4 bits graças ao amigo do peito chamado Derli.

Inicialização em 8 Bits:

Para inicializarmos o display precisamos jogar uma seqüência de códigos binários no mesmo. Para que isto possa acontecer, o pino RS (pino 4) e o pino R/W (pino 5) devem estar em nível 0 (entrada de instrução e escrita no display).

Observação:
Instrução é um comando para o display funcionar.
Escrita é uma informação que aparecerá no display.

Condições Inicias:

RS = 0 para mandar as instruções.
R/W = (R = read = leitura e W = write = escrita) a 0.
E = varia entre 0 e 1 de acordo com as informações anteriores.

Seqüência Binária:

Liga o circuito e espera 45ms (tempo mínimo de 15ms)

Pino 6 (E) = 1

Espera 1ms (pode ser 1µs que funciona)
Pino 6 (E) = 0
Espera 45ms (tempo mínimo 4,1ms)

Pino 6 (E) = 1
Espera 1ms (pode ser 1µs que funciona)
Pino 6 (E) = 0
Espera 45ms (tempo mínimo 100µs)

Pino 6 (E) = 1
Espera 1ms (pode ser 1µs que funciona)
Pino 6 (E) = 0
Espera 45ms (tempo mínimo 4,1ms)

Pino 6 (E) = 1
Espera 1ms (pode ser 1µs que funciona)
Pino 6 (E) = 0
Espera 45ms (tempo mínimo 40µs)

N = 1 se o display for de duas linhas.
N = 0 se o display for de uma linha.

DL = 1 se a inicialização for com 8 bits, este caso.
DL = 0 se a inicialização for com 4 bits.

F = 1 se cada caractere for de 5 x 11 pontos.
F = 0 se cada caractere for de 5 x 8 pontos.

Pino 6 (E) = 1
Espera 1ms (pode ser 1µs que funciona)
Pino 6 (E) = 0
Espera 45ms (tempo mínimo 40µs)

D = 0 o display fica desligado.
D = 1 o display fica ligado. É interessante e mais didático deixar D = 1.

C = 0 o cursor fica desligado.
C = 1 o cursor fica ligado. Aparecerá o cursor na tela, como se fosse uma “underline” ou risquinho).

B = 0 o cursor não pisca.
B = 1 o cursor pisca. Eu acho melhor deixar o cursor piscando.

Pino 6 (E) = 1
Espera 1ms (pode ser 1µs que funciona)
Pino 6 (E) = 0
Espera 45ms (tempo mínimo 1,64ms)

Esta linha limpa o display.

Pino 6 (E) = 1
Espera 1ms (pode ser 1µs que funciona)
Pino 6 (E) = 0
Espera 45ms

ID = 1 o display escreve para a direita.

S = 0 o display não shift (não se desloca).
S = 1 o display shift (se desloca).

Término da inicialização com 8 bits.

Podemos perceber que a inicialização é uma seqüência binária com 8 dígitos.
Também percebemos que entre uma linha e outra, deve haver um pulso 1 de enable (pino 6 = E) e depois uma temporização.
Todas estas temporizações são conseguidas com rotinas de tempo ou instruções nops.
Determinadas letras devem ser substituídas por 0 ou 1 de acordo com o que desejamos.
Eu aconselho o seguinte:
D = 1
DL = 1
C = 1
B = 1
ID = 1
S = 1
F = 0 –> na prática, pelo menos na minha, a maioria dos displays tem cada caractere formado por 5 “quadradinhos” na horizontal por 8 “quadradinhos” na vertical e portanto são 5 x 8.

Mas, escrever no display que é bom nada...

Por enquanto estamos entendendo o display e depois vamos dar recursos para você brincar com ele.
Mas, a teoria ainda não acabou.
Cada linha em um display de 16 caracteres é dividida em duas partes. Cada parte possui 8 caracteres. Para escrever após o oitavo caractere, é necessário enviar uma instrução para o display para que ele enderece o seu banco de memória RAM em C0 que corresponde a 192 em decimal e a 1 1 0 0 0 0 0 0 em binário.
Tudo há seu tempo...

Rotina para escrever Instruções:

```
bcf porta, 3          ; coloca entrada RS em 1 e permite a entrada de instruções no display
movlw b ‘instrução’   ; colocar a instrução em binário em W, repetir isto quantas instruções
                      ; forem necessárias
call  comando         ; chama sub-rotina para escrever instruções, o nome desta sub-rotina
                      ; é comando.
```

fomando:

```
movwf  portb          ; move o valor de W para a portb que está ligada com o display
nop                   ; deixa passar 1µs (devido ao clock de 4 MHz)
bcf  porta, 2         ; coloca 0 em E (pino 6 do display)
nop                   ; escreve 1µs (poderia ser 1ms mas daí precisaria de outra
                      ; sub-rotina e assim funciona)
bsf  porta, 2         ; coloca 1 em E
```

```
nop                     ; espera 1µs segundo
bcf  porta, 2           ; coloca 0 em E
                        ; cuidado se mudar os pinos do PIC

; --- . ----- . ---- . ---- . ---- sub-rotina de ± 45ms --- . --- . ---- . ----

m45:
movlw 250               ; coloca o valor de 250 em W
movwF  tempo            ; coloca o valor de 250 na variável tempo

m451:
movlw  60               ; coloca o valor 60 em W
movwf  tempo1           ; coloca o valor 60 em tempo1

m452:
decfsz  tempo 1         ; decrementa 1 do valor na variável tempo1
goto m452               ; vai para m452
decfsz  tempo           ; diminui 1da variável tempo
goto  m451              ; vai para m451
return                  ; volta para a linha abaixo do comando call
end                     ; final dos comandos


; --- . ----- . ---- . ---- . ---- final da sub-rotina --- . --- . ---- . ----
```

Rotina para Escrever uma Letra ou Dado:

```
bsf  porta, 3           ; coloca entrada RS em 1 e permite escrita
                        ; de um dado no display
movlw b 'letra'         ; colocar a letra em binário no W
call  escrever          ; chamar a sub-rotina escrever para ter os
                        ; comandos necessários
                        ;

escrever:

movwf  portb            ; move o valor de W para a portb
nop                     ; aguarda um 1µs (para clock de 4 MHz)
bcf  porta, 2           ; coloca 0 em E (pino 6 do display)
nop                     ;aguarda 1µs
bsf  porta, 2           ; coloca 1 no pino E do display
nop                     ; aguarda 1µs
bcf  porta, 2           ; coloca zero (0) no pino E do display
                        ; os comandos acima, habilitam a entrada
                        ; da letra no display

ms1:
movlw  . 250            ; carrega W com 250. O ponto indica decimal
movwf xxxx              ; carrega xxxx com 250

ms1a:
nop                     ; perde 1µs para ajudar a conseguir a
                        ; sub-rotina de tempo correta
decfsz xxxx             ; diminui 1 de xxxx e pula a inscrição seguinte
                        ; quando o valor de xxxx for igual a (0) zero.
goto ms1a               ; vai para ms1a
```

```
return                  ; volta da rotina
end                     ; final dos comandos
                        ; --- . ----- . ---- . ---- . ---- final da sub-rotina --- . --- . ---- . ----
```

É importante perceber que as sub-rotinas **comando:** e **escrever:** devem ficar no final do programa e a chamada para ir até uma ou outra deve ficar na posição desejada do programa.

Veja:

rotina para escrever instruções

```
Assembler                       em um ponto
                                do programa
assembler

call  comando

comando:

assembler
                                no final do programa

assembler
```

Andando com o Texto:

Lembre-se de fazer o bit RS do display ficar igual a 0, veja:

```
bcf  porta, 3           ; o bit RS será igual à zero
```

Lembre-se também que será necessário se criar uma rotina de temporização para definir a velocidade de deslocamento.

Um exemplo ficaria assim:

```
goto deslocamento       ; vai para a rotina de deslocamento

;-------------------------rotina de deslocamento----------------------------------------------

deslocamento:

bcf porta, 3            ; limpa RS e escreve comando
movlw B ' 0 0 0 0 1 1 1 0'  ; carrega W para mover o texto
                        ; para a direita
movwf  portb            ; carrega o portb com o valor de W
bcf  porta, 2           ; coloca 0 em E
nop                     ; aguarda 1μs
bsf  porta, 2           ; coloca 1 em E
nop                     ; aguarda 1 μs
bcf  porta, 2           ; coloca 0 em E, e finaliza as instruções
                        ; para o deslocamento


; --- . --- . --- . --- rotina do 1 a 2 segundos-----------------------------------------
```

```
1a2s:
movlw 4                    ; carrega W em 4
movwf tempo                ; carrega tempo com o valor de W

1a2sa:
movlw  40                  ; carrega W com 40
movwf  tempo1              ; carrega tempo1 com 40

1a2sb:
movlw 250                  ; carrega W com 250
movwf tempo2               ; carrega tempo2 com 250

1a2sc:
nop                        ; perde 1µs (4 MHz)
decfsz tempo2              ; decrementa tempo2        4µs x 250 = 1ms
goto 1a2sc                 ; vai para 1a2sc

decfsz tempo1              ; decrementa tempo1        3µs x 40 = 120µs
goto 1a2sb                 ; vai para 1 a 2sb

decfsz tempo               ; decrementa tempo         3µs x 4 = 12
goto 1a2sa                 ; vai para 1 a 2sa
return                     ; volta à chamada
end                        ; final dos comandos
```

Observe estes tempos para ver de uma forma simples, como funciona um loop de tempo.
O loop chamado de (1) dará 250 “voltas” e perderá 4µs em cada uma;

1µs na instrução nop;
1µs na instrução decFsz;
2µs na instrução goto.

Sendo assim 250 x 4µs = 1ms.

O loop chamado de (2) dará 40 “voltas” e perderá 3µs em cada uma:

1µs na instrução decFsz;
2µs na instrução goto.

Sendo assim 40 x 3µs = 120µs.

O loop chamado de (3) dará 4 “voltas” e perderá 3µs em cada uma:

Sendo assim temos 4 x 3µs = 12µs.

Para encontrarmos o tempo total gosto para que este loop termine, basta multiplicar estes três tempos (isto é uma forma simples e não com 100% de precisão, na verdade a tempo real será sempre maior que o calculado assim).
Veja:

1ms x 120µs x 12µs = 1.000 x 120 x 12 =1.440ms = 1,44s.

O tempo será de aproximadamente, 1,44s para o deslocamento do texto. Este tempo será o necessário para o texto se deslocar de uma posição no display para outra, ou seja, se o display tiver 16 posições o tempo total será de 1,44s x 16.
Um display de 1 x 16 é formado por duas linhas, uma após a outra de 8 caracteres. Para escrever após o oitavo caractere é necessário endereçar a memória RAM do display para C0H ou 192 decimal ou 11000000B. Para fazer isto podemos usar a seqüência em assembler seguinte:

```
bsf porta, 2          ;
bcf porta, 3          ;
movlw B '11000000'    ; ou 192 em decimal ou C0 em hexadecimal
movwf portb           ;
nop                   ;
bcf porta, 2          ;
nop                   ;
bsf porta, 2          ;  reset, seta e reseta o bit E do display
nop                   ;
bcf porta, 2          ;
movlw  250            ;
movwf  tempo          ;
x:
movlw  60             ;
movwf  tempo1 ;
x1:
decfsz  tempo1 ;
goto  x1              ;
decfsz  tempo         ;
goto  x               ;
bsf  porta, 3         ;
```

Para displays de 2 x 16 o procedimento é o mesmo, só que deve ser feito após se escrever o 16 caracter.
Em alguns displays devemos mudar o endereço para 40H para mudar de linha.

A numeração dentro do display está em hexadecimal e em um padrão que começa em 80H, mas, normalmente podemos chamar este endereço subtraindo 80H daí ficaria assim:

Sem somar 80H (lembre-se, você pode ir para o endereço que quiser do display somando ou não 80H ou 128 decimal), você tem a seguir os endereços iniciais e finais de cada linha de diversos displays.

20 x 1:
Linha 1 –> 00H às 13H

20 x 2:
Linha 1 –> 00H às 13H
Linha 2 –> C0H a D3H

20 x 4:
Linha 1 –> 00H às 13H
Linha 2 –> 40H às 53H
Linha 3 –> 14H às 27H

Linha 4 –> 54H às 67H

40 x 2:
Linha 1 –> 00H às 27H
Linha 2 –> 40H às 67H

Lembre-se de chamar o endereço certo.

Observação:

As letras podem ser escritas assim:
movlw 'A' ; carrega W com a letra A
movwf portb ; escreve A no display

Mas, isto é uma característica da MPLAB e poderá não funcionar com outro compilador.
Lembre-se sempre de terminar ma escrita em um display com um loop ou um loop sem fim, principalmente se você quer fazer apenas um teste inicializando e escrevendo alguma palavra.

Informações adicionais 3
EXEMPLOS DE SUB-ROTINAS DE TEMPO

- Rotina de ± 15ms V – 15.000µs (supondo 4 MHz de Clock).

```
ms:             ; aqui se perde o tempo da instrução Call
movlw  230      ; + 1µs
movwf  tempo    ; + 1µs

ms15:           ;
movlw  20       ; + 1µs
movwf  tempo1   ; + 1µs

ms1sa:
decfsz  tempo1 ; + 1µs
goto  ms15a     ; + 2µs
decfsz  tempo   ; + 3µs
goto  ms15      ;
return          ; + 2µs
```

- Rotina de ± 4,1ms – 4.100µs (para 4 MHz de Clock):

```
ms:             ;
movlw  180      ;
movwf  tempo2   ;

ms4,1:
movlw  6        ;
movwf  tempo3 ;

ms4,1a:         ;
decfsz  tempo 3 ;
goto  ms4,1a    ;
decfsz  tempo2 ;
```

```
goto  ms4,1          ;
return               ;
```

- Rotina de ± 40µs (para clock de 4 MHz):

```
ms:                  ; a instrução Call chama ms (+ 2µs)
movlw  12            ; carrega W com o valor 12 (+ 1µs)
movwf  tempo5        ; carrega o variável tempos com o
                     ; valor do registrador W (+ 1µs)

ms40:                ; nome para "ajudar" o goto
decfsz  tempo5n      ; diminui 1 de tempo 5 e perde 1µs quando
                     ; tempo 5 for = 0 pula a instrução seguinte (+ 1µs)
goto  ms40           ; vai para ms40 (+2µs)
return               ; volta para a linha após a instrução Call (+ 2µs)
```

- Rotina de ± 100µs (para um cristal de 4 MHz):

```
ms:                  ;
movlw  33            ;
movwf  tempo4        ;

ms 100:              ;
decfsz  tempo4 ;
goto  ms100          ;
return               ;
```

- Rotina 1,64ms – 1.640µs (para um xtal de 4 MHz):

```
ms:                  ;
movlw  82            ;
movwf  tempo6 ;

ms1,64:              ;
movlw  5             ;
movwf  tempo7        ;

ms1,64a:             ;
decfsz  tempo7       ;
goto  ms1,64a        ;
decfsz  tempo6       ;
goto  ms1,64         ;
return               ;
```

- Rotina de ± 60s (para xtal de 4 MHz):

```
ms250:               ;
movlw  240           ; W = 240
movwf  y             ; Y = W = 240

ms250c:              ;
movLw  250           ; W = 250
```

```
movwF  tempo        ; tempo = W = 250

ms250a:             ;
movlw  250          ; W = 250
movwf  tempo1       ; W = tempo1 = 250

ms250b:             ;
nop                 ; espera 1μs
decfsz  tempo1      ; diminui 1 de tempo1 e pula a próxima
                    ; linha se tempo 1 = 0
goto ms250b         ; vai para ms250b
decfsz tempo        ;
goto ms250a         ;
decfsz  y           ;
goto  ms250c        ;
return              ;
end                 ;
```

veja que se a sub-rotina for a última “coisa” do programa, mesmo depois da instrução return é “interessante” se colocar a instrução end.

- Rotina de tempo de ± 250ms (4 MHz):

```
ms250:              ; uma instrução call chama ms250 e isto
                    ; demora 2 ciclos de memória (2μs)
movlw  50           ; o registrador de trabalho W será carregado
                    ; com o valor 250
movwf  tempo        ; a variável tempo será carregada com o valor
                    ; do registrador W

ms250A:             ; nome para se conseguir usar a instrução
                    ; goto e fazer Loop
movlw  250          ; carrega W com 250 em decimal
                    ;
movwf  tempo 1      ; carrega tempo 1 com o valor de W

ms250B:             ; nome para se chamar com a instrução goto
nop                 ; perde 1 ciclo de máquina
                    ;
decfsz  tempo 1     ; diminui 1 de tempo 1 quando o tempo 1 for
goto ms250B         ; igual a 0 pula a linha seguinte, que é esta.
                    ; A linha acima vai para ms250B.
decfsz  tempo       ; Decrementa ou diminui 1 de tempo
goto  ms250A        ; manda o programa para o nome ms250A
return              ; volta para uma linha após a instrução Call
end                 ; última linha do programa
```

Perceba que para se fazer uma sub-rotina de tempo, precisamos de nomes:

- Uma para chamar a rotina, neste caso ms250;
- Outros para fazer os loopings, neste caso ms250A e ms250B.

Também precisamos de variáveis, neste caso tempo e tempo1.

Informações adicionais 4
INICIALIZANDO UM DISPLAY COM 4 BITS

O bom de inicializar um display com 4 bits é que você "economiza" quatro fios, ou quatro pinos do PIC que poderão ser usados para outras funções. **"Os bits não usados (DB3 a DB0) devem ter o valor zero (0)".**

Veja a inicialização:

1 – loop de 15ms

2 – RS R/W DB7 DB6 DB5 DB4
0 0 0 0 1 1

3 – loop de 100µs

4 – RS R/W DB7 DB6 DB5 DB4
0 0 0 0 1 1

5 – loop de 100µs

6 – RS R/W DB7 DB6 DB5 DB4
0 0 0 0 1 1

7 – loop de 4,1ms

8 – RS R/W DB7 DB6 DB5 DB4
0 0 0 0 1 0 –> esta linha define inicialização em 4 bits

9 – loop de 40µs

10 – RS R/W DB7 DB6 DB5 DB4
0 0 0 0 X 0
0 0 1 F X X –> o valor de F define a matriz do display
se F = 1 display de matriz 5 x 11
se F = 0 a matriz do display é de 5 x 8
X = não importa o valor

11 – Loop de 40µs

12 – RS R/W DB7 DB6 DB5 DB4
0 0 0 0 0 0
0 0 1 D C B –> D = 0 display desligado, D = 1 display ligado
C = 0 cursor desligado, C = 1 cursor ligado
B = 0 cursor sem piscar, B = 1 cursor piscando

13 – Loop de 40µs

14 – RS R/W DB7 DB6 DB5 DB4
0 0 0 0 0 0
0 0 0 0 0 1

15 – Loop de 1,64ms

16 – RS R/W DB7 DB6 DB5 DB4
0 0 0 0 0 0
0 0 0 1 ID S –> ID = 1 cursor para direita, ID = 0 cursor para esquerda
S = 0 escrita não se move, S = 1 escrita se move

Terminou a inicialização de 4 bits.

Lembre-se que entre cada seqüência de bits de instruções, após os bits, você deve fazer o seguinte:

Pino 6 (E) = 1
Espera 1ms
Pino 6 (E) = 0

Dúvidas? Leva novamente a inicialização com 8 bits.

Não há necessidade de colocar algum tempo entre a primeira linha de bits e a segunda linha de bits dos comandos 10, 12, 14 e 16.

Para enviar os comandos ou o que vai ser escrito no display, faça da seguinte forma:
Envie primeiro os 4 bits mais significativos e depois os 4 bits menos significativos. Veja:

RS R/W DB7 DB6 DB5 DB4
0 0 1 1 1 1

depois,

RS R/W DB3 DB2 DB1 DB0
0 0 1 1 1 1

Informações adicionais 5
UMA SIRENE COM PIC E COM FET

A finalidade deste circuito é fazer com que você comece a brincar com Mosfets. Esta sirene terá dois tons, 400 Hz e 1 kHz.
Vamos ver o circuito da sirene sem o PIC.

Faça download do arquivo sirene.rar

RL = alto falante de 8Ω e 5 Watts
D1 = diodo de proteção = 1N4007
Q1 = Mosfet IRF640
C = 100nF para 1kHz e 200nF para 400 Hz
R = 2K2Ω x 1/4 Watts

A comutação entre 0 e 5 Volts na saída Rb1 do PIC fará com que o Fet ora conduza ora corte e o alto falante “berre”. O som não é muito agradável, mas, o resultado do aprendizado é bom.

Informações adicionais 6
Testando a inicialização com 8 bits

A finalidade deste circuito é testar a inicialização do display de LCD com 8 bits. Você deve saber iniciá-lo na teoria e na prática com 8 e 4 bits. Aqui estamos apresentando um circuito para inicialização e funcionamento com 8 bits. Você pode gravar o PIC na placa Pratic628 da Exsto, mas, depois o retire e o coloque aqui.
Você pode usar este circuito como um mini - placar, onde passam mensagens repetidas. Eu montei este pequeno circuito para ser usado como placar de um mini - estádio, isto há long time ago.

Baixe o circuito display_pratica.rar

O display usado é um 2 x 16 baseado no IC HD 44780.

Através de 7 exemplos que estão escritos aqui, você poderá brincar bastante com o display LCD. Veja as diferenças entre os programas, faça mudanças, aprenda na teoria e na prática.

Baixar programas LCD.rar

Informações adicionais 7
ICSP

ICSP significa (In Circuit Serial Programing) e traduzindo para um bom português, quer dizer programando o PIC com ele já montado no próprio circuito. Você desenvolve um projeto, vende milhares e, para isto monta milhares, e só programa com tudo “soldado” inclusive o PIC. Como você vendeu centenas de milhares, você pode me fazer uma doação, em cash, que eu agradeço.
Através da programação no próprio circuito ou ICSP, fica muito mais fácil você fazer um upgrade no software do seu PIC.
A gravação ICSP serve tanto para regravar toda a memória de um dispositivo Flash, bem como o que resta de memória em um dispositivo OTP.
O ideal para fazer este tipo de gravação é o gravador (hardware) Pro Mate II da Microchip ou algum equivalente real.

O ICSP usa cinco pinos para a gravação do software:

- O clock que vai ligado na RB6.
- Os dados que vão ligados no RB7.
- A tensão de programação que vai ligado no MCLR/Vpp.
- A tensão de alimentação Vcc ou Vdd.
- O terra ou Vss.

Para programar o PIC você deve elevar a tensão de programação (Vpp) para 13 Volts (entre 3,5 a 13,5 Volts).

Informações adicionais 8
O CIRCUITO INTEGRADO MT 8870

O circuito integrado MT 8870 é um detetor de DTMF. DTMF são aqueles tons emitidos por um telefone comum. Cada tom DTMF é composto de duas freqüências diferentes e com forma senoidal.
Com um telefone comum, mais um IC destes (pode ser o HT 9170 também), mas um 16F628 você pode brincar de controlar tudo a distância via telefone comum/fixo ou telefone celular.

Faça download do arquivo MT8870.rar

O pino 1 é a entrada não inversora.
O pino 2 é a entrada inversora.

Só de ver isto dá para perceber que podemos trabalhar com áudio balanceado ou desbalanceado.

O pino 3 define o ganho do IC, junto com alguns resistores externos.
O pino 4 gera uma tensão de referência e deve ficar ligado com o pino 1.
O pino 5 inibe a detecção dos tons. Para detectar ele deve estar em 0 (zero).
O pino 6 desabilita o IC e desliga e oscila. Para o circuito funcionar ele deve estar em 0 (zero).
O pino 7 é a entrada do oscilador.
O pino 8 é a saída do oscilador. Normalmente colocamos entre estes dois pinos um cristal de 3,579545 MHz, que corresponde há um cristal de cor do sistema NTSC.
O pino 9 é o terra e vai ligado em 0 Volt.
O pino 10 define se as saídas Q1 a Q4 funcionam normalmente ou ficam em TRI-STATE (alta impedância). Para funcionarem normalmente, este pino deve estar ligado no Vcc.
Os pinos 11, 12, 13 e 14 são as saídas que apresentam um código em binário para cada tom DTMF.
O pino 15 fica em Vcc quando o IC detecta um tom DTMF. Este pino pode ser usado para controle.
Os pinos 16 e 17 estão ligados há uma constante RC que determina um período para validação dos tons DTMF.
O pino 18 á o Vcc que deve ser alimentado com +5 Vcc.

Normalmente os valores de R1 e R2 são iguais a 100K, mas, são eles que definem o ganho e/ou a sensibilidade do circuito.

$$G = \frac{R1}{R2}$$

Na prática eu já usei ganho de 10, ou seja, usei R1 = 100K e R2 = 10K, principalmente se a linha que você usar for ruidosa ou a fonte de sua bancada não eliminar todo o ripple e RFI.

As saídas Q1 a Q4 e CT fornecem uma corrente baixa e caso você queira ligar leds a elas para analisar o funcionamento, você deve usar transistores como driver de corrente, veja:

Informações adicionais 9
RS 232 – USART

RS 232 quer dizer padrão recomendado 232 ou, em inglês “Recommended Standard 232”.

Este padrão foi criado há muito tempo, há algumas décadas, e hoje em dia a forma correta de chamá-lo é EIA 232, EIA quer dizer, “Eletronic Industries Associaton”.
Existem alguns nomes e sinais que são definidos por este padrão:

DTE –> terminal responsável pelo envio da informação. Normalmente um microcomputador.
DCE –> equipamento responsável pelo “recebimento” das informações e por manuseá-las.
Para explicar poderíamos citar um microcomputador “controlado” ou conectado com uma placa de programa de PICs, através da porta serial (a do mouse).

Vamos aqui apresentar os sinais usados em um cabo serial DB9 (com conector DB9) com ênfase em aplicações com microcontroladores.

Tx – transmissão de dados.
Rx – recepção de dados.
DTE – indica DTE pronto.
Terra – gnd.
DCE – indica DCE pronto.
Clear to Send – “pode enviar” – CTS.
Request to Send – “vu enviar” – RTS.

Agora o que eles fazem?

Tx – está funcionando quando estão sendo transmitidos bits do micro para o gravador (de DTE para o DCE). Quando nada é transmitido, sua tensão é negativa em relação ao terra e isto significa nível lógico 1.
Rx – está operando quando o microcomputador recebe bits do gravador (bits do DCE para o DTE). Sem uso, nível lógico 1 o que corresponde há um nível de tensão negativa.
DTE – este sinal passa para nível lógico 0, o que corresponde há um nível positivo de tensão, quando o micro quer falar com o gravador (o DTE informa o DCE que quer falar. Acabou a conversa, ele vai para nível lógico 1). Este sinal também pode ser chamado de DTR.
Terra – é o terminal comum entre DTE e DCE.
DCE – este sinal é usado quando o microcomputador é ligado com um modem e não é importante em nossas aplicações.
CTS – este sinal é habilitado pelo gravador (DCE) em nível lógico 0 (tensão positiva) e avisa ao microcomputador (DTE) que a transmissão de bits pode começar.
RTS – este sinal é habilitado em 0 (tensão positiva) para enviar o gravador (DCE) que ele deve aceitar os bits transmitidos pelo microcomputador (DTE). Quando o DCE está pronto ele avisa com o sinal CTS.

• Níveis de tensão dos sinais:

Nível lógico 1 –> tensões entre -3 Volts a -25 Volts em relação ao terra.
Nível lógico 0 –> tensões entre +3 Volts a + 25 Volts em relação ao terra.

Entre -3 a +3 Volts “tudo pode acontecer” pois estes níveis estão em uma região de transição e é muito útil que permaneçam o mínimo tempo nela. Apenas o necessário para a subida e descida dos sinais.

• Níveis de corrente:

Normalmente se mantém o nível de corrente inferior a 500mA.
Você pode alimentar circuitos simples com estes sinais, sem o uso de uma fonte de alimentação, este é o caso de alguns gravadores de baixo custo:

• Velocidades da comunicação comuns:

300bps
1200bps
2400bps
4800bps
9600bps
19200bps
33.000bps
48.000bps
56.000bps

• Pinagem do DTE (microcomputador):

1 – NC* * em nossas aplicações
2 – Rx
3 – Tx
4 – DTE
5 – terra
6 – DCE
7 – RTS
8 – CTS
9 – NC*

• Pinagem do DCE (gravador):

1 – NC* * em nossas aplicações
2 – Tx
3 – Rx
4 – DTE
5 – terra
6 – DCE
7 – CTS
8 – RTS
9 – NC*

Quando você monta um circuito para comunicação serial pode ser que esteja trabalhando com níveis TTL (0 e 5 Volts) e precisará convertê-los para o padrão EIA 232. Você pode fazer isto, usando o IC MAX 232, que gera tensões de -10 e +10 Volts a partir de uma tensão de +5 Volts.

Este tipo de ligação necessita da conexão com o cabo/conector DB9 como mostrado como Null Modem (sem modem) e é utilizado para se conectar dois DTE (ou dois dispositivos “inteligentes” como um microcontrolador com um microcomputador).
O cabo entre os dois dispositivos só será conectado através de 3 vias que são o Rx o Tx e o terra. Os pinos 8 e 7 são ligados em curto e os pinos 4,1 e 6 também são ligados em curto. Isto “engana” o microcomputador permitindo a comunicação.
Com este tipo de ligação e com os dois dispositivos trabalhando “na mesma velocidade” não precisamos ter o controle de fluxo.

Apenas para informação, podemos ter dois tipos de controle de fluxo de dados:

- Por hardware – que precisa das linhas RTS e CTS além do Tx, Rx e terra e que é conhecida por controle RTS/CTS.

- Por software - onde o controle de dados é feito através de dois caracteres/números (caracteres ASCII) e que é conhecido por XON/XOFF e só usa Tx, Rx e terra.

• Algumas considerações:

- No modo síncrono só poderemos trabalhar como half-duplex (meio-duplo), ou seja, quando há transmissão não há recepção e vice versa. Pois um pino é usado para clock;
- Os pinos de comunicação são o Rb1 e Rb2;
- Temos que configurar os registradores próprios da USART;
- O pino Rb1 é o Rx em modo assíncrono e DT (ou entrada e saída de dados) em modo síncrono;
- O pino Rb2 é Tx em modo assíncrono ou entrada/saída de clock. Quando Rb2 é a saída de clock dissemos que estamos trabalhando em modo mestre. Quando Rb2 é entrada de clock dissemos que estamos trabalhando em modo escravo;
- Quem gera o clock é o "chefe";
- Tente trabalhar em modo assíncrono, com 8 bits e cabo Null Modem.

Informações adicionais 10
ALGUMAS DICAS E ORELHADAS

- Uma forma simples de "parar" um programa é se fazer um loop sem fim.

Exemplo:

```
loop:
nop             ;
goto loop       ;
```

O loop ficará rodando e o programa ficará parado até que o PIC seja resetado.
- Cuidado para não confundir O (letra O) com 0 (zero);
- Toda variável que irá assumir um valor, deve estar indicada no espaço de constantes e RAM, exemplo:

```
x   equ         0DH   ;
```

para:

```
movlw   100     ; W = 100
movwF   x       ; X = 100
```

- Ao fazer um projeto no MPLAB o projeto e o arquivo ".asm" devem estar no mesmo diretório e este diretório só deve ser usado para eles.

- Quando definir as características do compilador coloque as seguintes em off:

- case sensitive
- cross-reference file

Coloque em on:

- INHX8M
- error File
- List File

O resto deixe em branco (salvo alguma informação específica ao contrário).

- O ideal é que as portas sejam definidas sempre pelos seus números:

Ra0 = 0
Ra1 = 1
Rb3 = 3
Etc.

- É necessário compilar o código fonte (.asm), dentro de um projeto. Caso contrário, ele não aparece na janela na hora de gravar o PIC (isto para quem usar o MPLAB 5.7.4X para tudo).

- Os valores para se escrever em um LCD podem estar em decimal de acordo com a tabela de caracteres já mencionada.

- Podemos escrever todo o programa no bloco de notas e salvar com a terminação “.asm”. Depois rodamos o compilador (MPASWIN) e passamos para “.hex”. Depois gravamos.

Se o Propic do MPLAB, começar a apresentar erros, feche-o e o inicie-o novamente.

- Nunca inicie uma chamada de sub-rotina com um número, exemplo:

```
call  1ms       ;
  “     “       ;
  “     “       ; isto está errado
  “     “       ;
1ms:            ;
```

Faça sempre com uma letra:

```
call  ms ;
  “     “       ;
  “     “       ; isto está certo
  “     “       ;
ms:             ;
```

- As versões do MPLAB acima da 5.7.40 são superiores porém só rodam com o Windows 98SE para cima com o XP, Millenium, etc) e se o seu micro for meio “lento” ela será pesada.

- A versão do MPLAB 5.7.40 é mais leve, roda em Windows 95 e 98, porém pode apresentar alguns conflitos com alguns gravadores, versões de firmawares e pcs.

- Você pode usar um software de gravação para gravar qualquer arquivo “.hex”, mesmo que tenha sido criado em outro local. Basta importá-lo e mandar gravar.

- Se acontecer persistentemente, erros ao importar um arquivo “.hex” para o MPLAB, desinstale algum “outro” compilador e instale-o novamente associando-o ao MPLAB.

- Jamais crie mais de um projeto dentro da mesma pasta, pois poderão acontecer erros na gravação do PIC.

- Se após a gravação aparecer uma lista de erros, apague a memória Flash, feche o gravador, importe o arquivo novamente e grave novamente. Isto pode acontecer, mais provavelmente, se você usa um cabo conversor USB – Serial ou USB – paralelo. Ás vezes é necessário “ajustes” nesta comunicação.

O timer0, mesmo sem o prescaler já apresenta, dividido por dois, um sinal aplicado na entrada Rb4/TOCKI.

- Em uma sub-rotina, que poderá se repetir muitas vezes, podemos usar as mesmas variáveis, porém os nomes das chamadas para o goto devem ser diferentes.

- Se as diretrizes começarem da coluna 1, aparecerão “warnings” no resultado da compilação. Deixe um espaço antes de colocar uma diretriz.

Informações adicionais 11
Tentando ajudar...

Espero que você tenha lido esta apostila inteira e tenha aprendido mais do que já sabia.
Nossa intenção ao escrevê-la, antes de tudo, e ajudar a você leitor aprender sozinho. (Deu um trabalho danado escrever).
Visite estes sites:

www.luizbertini.net/circuitos.html - *inclusive circuitos para o PIC*
www.luizbertini.net/download.html
www.luizbertini.net/eletronica.html
www.luizbertini.net/livros.html

Luiz Bertini
De Janeiro a julho de 2004

Santicfy Yourself