UNIPAC
UNIVERSIDADE PRESIDENTE ANTÔNIO CARLOS
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO E COMUNICAÇÃO SOCIAL

CURSO DE CIÊNCIA DA COMPUTAÇÃO

Pietro Diovane Keoma Bergamaschi de Assis

MICROCONTROLADOR

BARBACENA
DEZEMBRO DE 2004

PIETRO DIOVANE KEOMA BERGAMASCHI DE ASSIS

MICROCONTROLADOR

Trabalho de Conclusão de Curso apresentado à Universidade Presidente Antônio Carlos, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

ORIENTADOR: Prof. Eduardo Macedo Bhering

BARBACENA
DEZEMBRO DE 2004

Pietro Diovane Keoma Bergamaschi de Assis

MICROCONTROLADOR

Trabalho de Conclusão de Curso apresentado à Universidade Presidente Antônio Carlos, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovada em _____/_____/_____

BANCA EXAMINADORA

_____

Prof. Eduardo Macedo Bhering (Orientador)
Universidade Presidente Antônio Carlos

_____

Prof. Eliseu César Miguel
Universidade Presidente Antônio Carlos

_____

Prof. Gustavo Campos Menezes
Universidade Presidente Antônio Carlos

Dedico este trabalho a minha querida avó, Zulmira Puiatti de Assis (In Memorian).

Agradeço aos meus amigos, familiares e professores, em especial ao Fausto, Leandro e César, grandes e eternos amigos; a meus pais Francisco e Lucimar, por acreditarem em mim, a

minha irmã Patysie e ao grande mestre Eduardo Bhering.

LISTA DE FIGURAS

LISTA DE SIGLAS

A/D – Analógico / Digital

ALU – Unidade Lógica Aritmética

CI – Circuito Integrado

CISC – Conjunto Complexo de Instruções

CPU – Unidade Central de Processamento

DIP – Empacotamento em duas linhas

GPR – Registros de uso Genérico

IDE – Ambiente Integrado de Desenvolvimento

LCD – Display de Cristal Líquido

LED – Diodo Emissor de Luz

NOP – Nenhuma Operação

PC – Contador de Programa; Computador Pessoal

PIC – Controlador de Interrupções Programável

PLC – Controlador Lógico Programável

R/W – Ler / Escrever

RC – Resistência-condensador

RISC – Conjunto Reduzido de Instruções

SMD – Dispositivos de Montagem em Superfície

XT – Cristal

SUMÁRIO

3.9.1 – Port A.....37

CAPÍTULO 1 – INTRODUÇÃO

Dentro do ambiente computacional e da eletrônica digital nota-se a crescente utilização dos microcontroladores, tanto nas indústrias (circuitos para automação e gerenciamento, ferramentas de bancada como multímetros e frequêncimetros, dentre outros) quanto em residências (portões eletrônicos, alarmes, impressoras, telefones). Porém não é dada a devida atenção, ou por falta de conhecimento ou por ser tratado como um simples Circuito Integrado (CI).

Um microcontrolador é um CI capaz de efetuar processos lógicos com extrema rapidez e precisão. A grande vantagem deste CI é a sua possibilidade de programação, o que o torna adaptável à finalidade desejada, e que possibilita seu ajuste de acordo com a tarefa que deverá executar (1).

Um dos primeiros microcontroladores a ser utilizado foi o 8051, que atualmente está caindo em desuso devido ao seu custo, tamanho e funções restritas. Atualmente os mais utilizados nas bancadas dos projetistas são os da Empresa Microchip: o PIC16F84A com 18 pinos e o seu irmão mais novo e mais poderoso o PIC16F628 também com 18 pinos. As intenções da Microchip são bem claras, pois estão investindo cada vez mais no desenvolvimento de microcontroladores com mais funções, memória e portas de E/S; ou seja, em breve haverá novos microcontroladores que se tornarão cada vez mais versáteis (2).

É possível se definir o microcontrolador como sendo um pequeno componente eletrônico, que possui uma “inteligência” que pode ser programável (1). Utilizada no gerenciamento de processos lógicos, esse gerenciamento pode ser entendido como o controle de periféricos, como: sensores, relês, resistências, display de cristal líquido (LCD), display’s, diodo emissor de luz (LED’s) dentre outros; para os leigos

um microcontrolador seria como um Controlador Lógico Programável (PLC) da Eletrônica.

No âmbito científico, se encontra em crescente pesquisa e desenvolvimento, à procura de novos microcontroladores, visando atender as exigências do mercado, com isso seria de suma importância se tomar conhecimento e se familiarizar com eles o quanto antes, caso se deseje estar atualizado e preparado para atender as necessidades do mercado atual.

O objetivo deste trabalho é o estudo dos seguintes aspectos: conceitos teóricos, programação, recursos, utilização e as funcionalidades dos microcontroladores, em especial dos Controladores de Interrupções Programáveis (da família PIC _ PIC16F84), devido ao seu baixo custo de aquisição e por estar dotado de todas as funções essenciais para o desenvolvimento dos mais variados sistemas, sendo versátil, compacto e poderoso.

Levantar uma literatura suficiente para oferecer aos interessados, suporte no aprendizado da programação e dos conceitos teóricos dos microcontroladores, uma vez que a eletrônica e a computação têm evoluído de forma surpreendente, e a eletrônica digital tem ficado cada vez mais acessível aos técnicos, engenheiros e até mesmo curiosos. Por este mesmo motivo surgiram os microcontroladores, visando aumentar ainda mais a evolução e a facilidade de desenvolvimento de novas tecnologias e produtos.

Por isso cresce cada dia mais a necessidade das pessoas se atualizarem e aprenderem novas formas de projetar novos sistemas. Porém, no Brasil, não se encontra facilidade quanto à obtenção de informações e com isso, espera-se que este trabalho venha atender ao menos em parte as necessidades desses. Resumindo, deseja-se efetuar um apanhado literário e por fim implementar um circuito que venha a demonstrar algumas das funcionalidades do microcontrolador.

CAPÍTULO 2 – OS MICROCONTROLADORES

O que existe hoje, quando se trata de microcontroladores, deve-se ao desenvolvimento da tecnologia dos circuitos integrados (3). Este desenvolvimento tornou possível armazenar milhares de transistores num único chip. Isso constituiu um dos requisitos para a produção de microprocessadores e, os primeiros computadores foram construídos adicionando periféricos externos tais como memória, linhas de entrada e saída, temporizadores dentre outros. Um crescente aumento do nível de integração, permitiu o aparecimento de circuitos integrados contendo simultaneamente processador e periféricos. Foi assim que apareceu o primeiro chip contendo um microcomputador e que passou a ser designado por microcontrolador.

2.1 – Microcontroladores x Microprocessadores

Um microcontrolador se diferencia de um microprocessador em vários aspectos. O mais importante, é a sua funcionalidade (4). Quanto a um microprocessador, para que ele possa ser usado, outros componentes devem ser adicionados, tais como memória e componentes para receber e enviar dados. Em resumo, isso significa que o microprocessador é o verdadeiro coração do computador. Já o microcontrolador foi projetado para ter tudo isso num só componente. Nenhum outro componente

externo é necessário nas aplicações, uma vez que todos os periféricos necessários já estão contidos nele. Assim, tem-se um baixo custo de tempo e espaço na construção de dispositivos.

2.2 – Unidade de Memória

A memória é a parte do microcontrolador cuja função é guardar dados (2). A figura 2.1 mostra um exemplo de um modelo simplificado de uma unidade de memória. Para uma entrada específica, obtém-se a saída correspondente. A linha Ler/Escrever (R/W) determina quando estamos lendo ou escrevendo na memória.

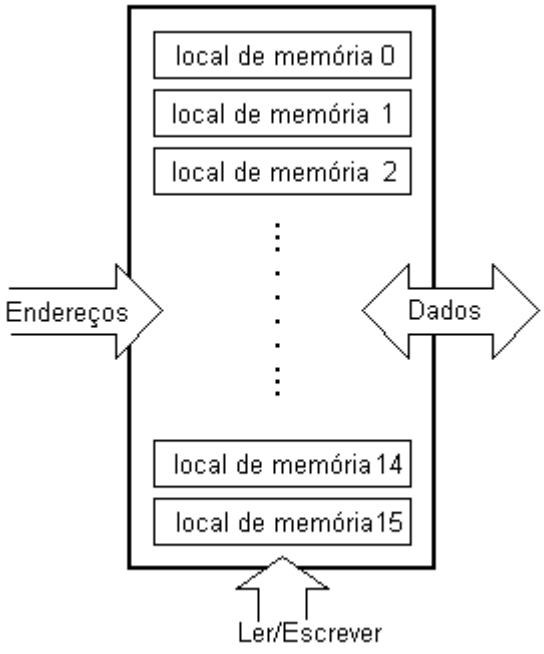


Figura 2.1 – Modelo da Unidade de Memória

A memória é composta pelo conjunto de todos os locais de memória, já o endereçamento é a seleção que se faz de um desses locais. Isto significa que é preciso selecionar o endereço desejado e esperar que o conteúdo desse endereço

seja apresentado. Além de ler de um local da memória, também é possível escrever num endereço da memória. Isto é feito utilizando uma linha adicional chamada linha de controle. Esta linha é designada por R/W (*read/write*) - ler/escrever. A linha de controle é usada do seguinte modo: se r/w=1, é executada uma operação de leitura, caso contrário é executada uma operação de escrita no endereço de memória. A memória é o primeiro elemento, mas são necessários mais alguns elementos para que o microcontrolador possa trabalhar.

2.3 – Unidade Central de Processamento

Adicionando mais 3 locais de memória a um bloco específico é possível ter as capacidades de multiplicar, dividir, subtrair e mover os seus conteúdos de um local de memória para outro. A parte que será inserida é chamada "*central processing unit*" (CPU) ou Unidade Central de Processamento. Os locais de memória nela contidos chamam-se registros. A figura 2.2 mostra um exemplo simplificado de uma unidade central de processamento (CPU) com três registros.

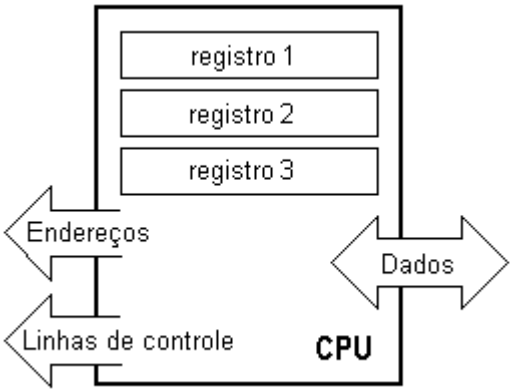


Figura 2.2 – CPU

Os registros são locais de memória cujo papel é ajudar a executar várias operações matemáticas ou quaisquer outras operações com dados, quaisquer que sejam os locais em que estes se encontrem (5).

2.4 – BUS

É o caminho que interliga a memória com a CPU. Fisicamente ele corresponde a um grupo de 8, 16 ou mais fios. Existem dois tipos de bus: o de dados e o de endereço (6). O número de linhas do primeiro depende da quantidade de memória que se deseja endereçar e o número de linhas do outro depende da largura da palavra de dados, para este caso é igual a oito. O primeiro bus serve para transmitir endereços da CPU para a memória e o segundo para ligar todos os blocos dentro do microcontrolador. A figura 2.3 mostra a ligação entre a memória e a unidade central de processamento através do BUS, para obter maior funcionalidade.

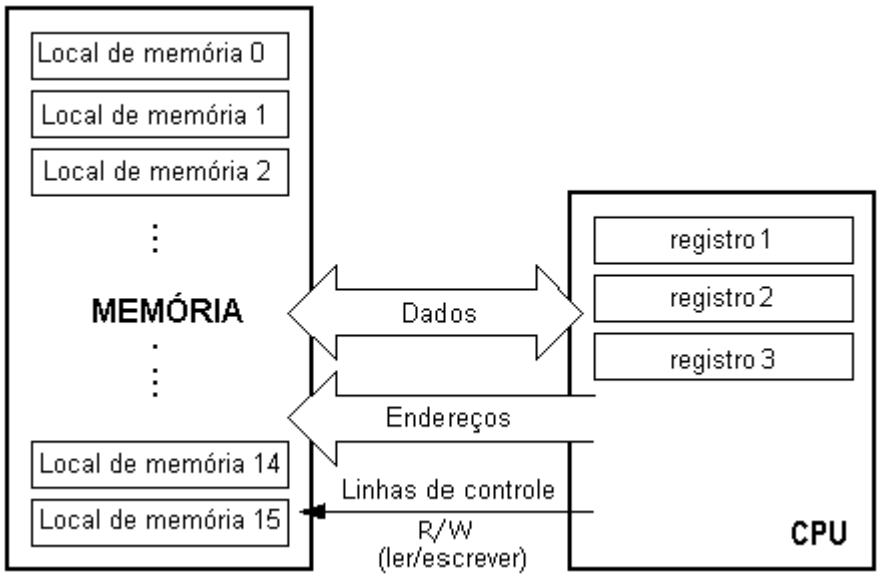


Figura 2.3 – Interligação entre Memória e CPU

2.5 – Unidade de Entrada / Saída

Para permitir o contato com o mundo exterior, foi adicionado mais um bloco que contém as localizações de memória, ligando o bus às linhas de saída do microcontrolador. Estas localizações, chamam-se "port's". Existem vários tipos de port: de entrada, de saída e de entrada/saída. Quando se trabalha com port, primeiro é necessário escolher o port com que se quer trabalhar e, em seguida, enviar ou receber dados para ou desse port. A figura 2.4 mostra um exemplo simplificado de uma unidade de entrada/saída (I/O) que fornece comunicação com o mundo exterior.

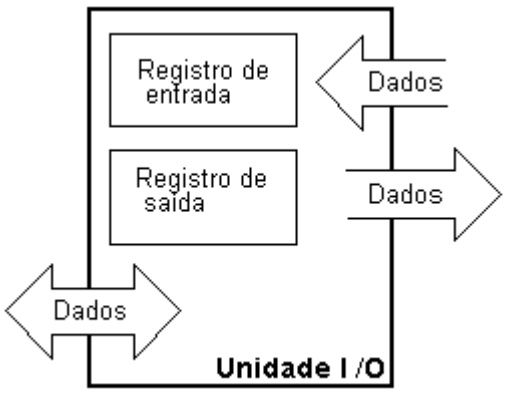


Figura 2.4 – Unidade de Entrada/Saída

2.6 – Unidade Série

É o bloco que possibilita a comunicação, uma vez que se tenham linhas separadas para o envio e recebimento, é possível receber e enviar dados (informação) simultaneamente. Ao contrário da transmissão em paralelo, aqui os dados movem-

se bit após bit em série, daí vem o nome de comunicação série. Depois de receber os dados é preciso ler e guardar-los na memória, no caso da transmissão de dados o processo é inverso. Os dados vêm da memória através do bus para o local de transmissão e dali para a unidade de recepção de acordo com o protocolo (7). A figura 2.5 mostra a unidade série usada para enviar e receber dados somente com três linhas.

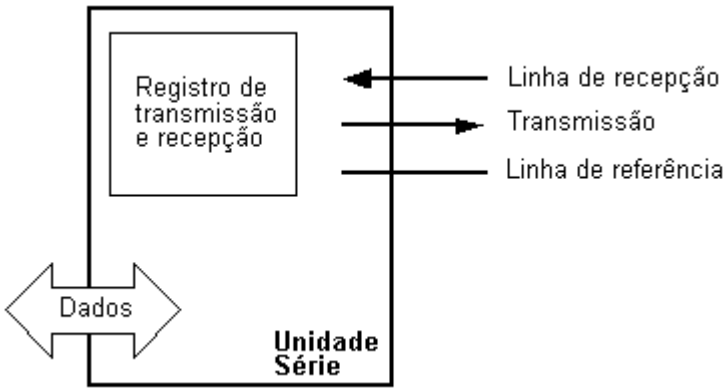


Figura 2.5 – Transmissão e Recepção

2.7 – Unidade de Temporização

É de grande interesse, pois dá informações acerca da hora, duração, protocolo, etc. A unidade básica do temporizador é um contador que é na realidade um registro cujo conteúdo aumenta uma unidade num intervalo de tempo fixo, assim, anotando o seu valor durante os instantes de tempo T1 e T2 e calculando a sua diferença, tem-se como saber o tempo decorrido. A unidade de temporização gera sinais em intervalos de tempos regulares, como é observado na figura 2.6.

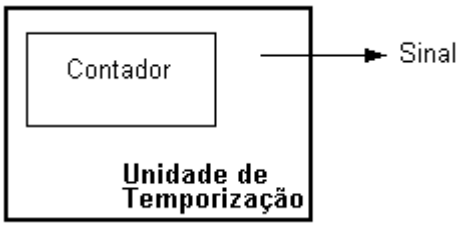


Figura 2.6 – Unidade de Temporização

2.8 – Watchdog (Cão de Guarda)

Este bloco é de fato outro contador que está sempre contando e que o programa sempre o coloca a zero quando é executado corretamente (1). No caso do programa "travar", o zero não será escrito e o contador, irá fazer o reset do microcontrolador quando alcançar o seu valor máximo (figura 2.7). Isto fará com que o programa rode novamente e desta vez corretamente. Este é um elemento importante para que qualquer programa se execute corretamente, sem precisar da intervenção do ser humano.

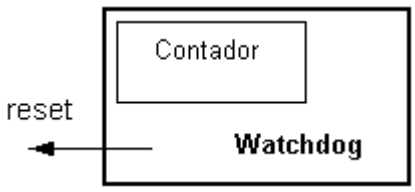


Figura 2.7 – Watchdog

2.9 – Conversor Analógico / Digital

Como os sinais dos periféricos são diferentes daqueles que o microcontrolador pode processar (zero e um), eles devem ser convertidos num formato que possa ser compreendido pelo microcontrolador (2). Esta tarefa é executada por intermédio de um bloco destinado à conversão analógica-digital ou com um conversor A/D. Este bloco vai ser responsável pela conversão de uma informação de valor analógico para um número binário e pelo seu trajeto através do bloco do CPU, de modo que seja processado de imediato. A figura 2.8 mostra o bloco para converter uma grandeza analógica numa digital.

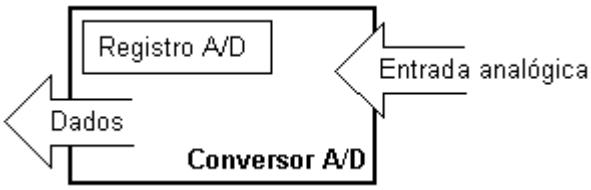


Figura 2.8 – Bloco Conversor

O gráfico da figura 2.9 representa a parte principal de um microcontrolador.

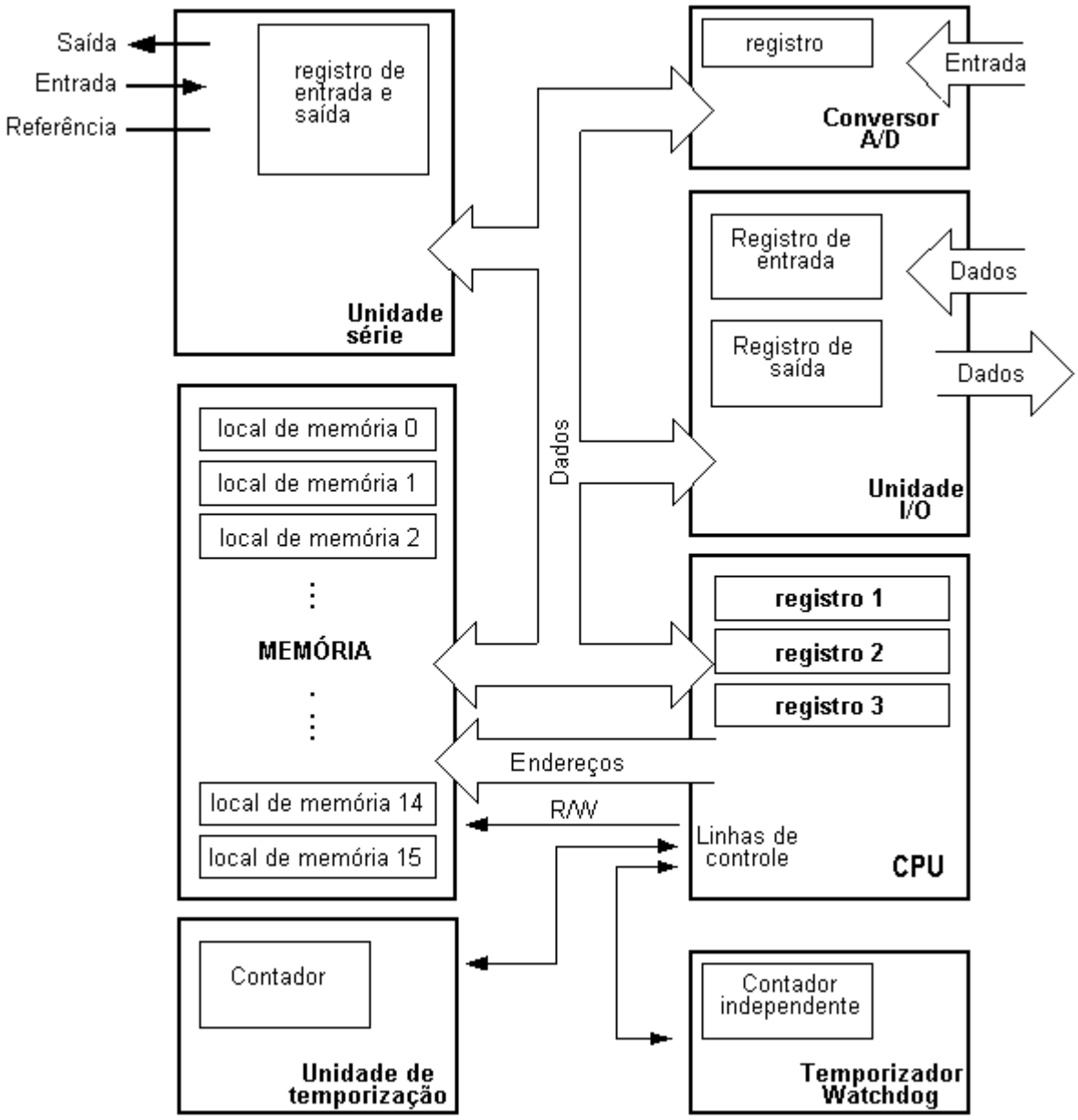


Figura 2.9 – Esquema de um microcontrolador com os seus elementos básicos e ligações internas

2.10 – Programa

A tarefa de programação pode ser executada em várias linguagens tais como o Assembler, C e Basic que são as linguagens normalmente mais usadas (5).

O Assembler pertence ao grupo das linguagens de baixo nível que implicam um trabalho de programação lento, mas que oferece os melhores resultados quando se pretende poupar espaço de memória e aumentar a velocidade de execução do programa (9). Como se trata da linguagem mais freqüentemente usada na programação de microcontroladores, ela será discutida posteriormente no capítulo 5.

Os programas na linguagem C são mais fáceis de se escrever e compreender, mas, também, são mais lentos para serem executados que os programas assembler.

Basic é a mais fácil de todas para se aprender e as suas instruções são semelhantes à maneira de um ser humano se expressar, mas como a linguagem C, é também de execução mais lenta que o assembler.

Em qualquer caso, antes de se escolher uma destas linguagens, é preciso examinar cuidadosamente os requisitos de velocidade de execução, de espaço de memória e o tempo que vai demandar para se fazer o programa em assembly (8). Depois que o programa estiver escrito, é preciso inserir o microcontrolador num dispositivo e colocá-lo para trabalhar. Para que isto aconteça, é necessário adicionar mais alguns componentes externos. Primeiro tem-se que dar vida ao microcontrolador fornecendo-lhe a tensão e o oscilador cujo papel é regular (através de pulsações) a execução das instruções do programa. Logo que é aplicada a tensão, o microcontrolador executa uma verificação de si mesmo, e vai para o princípio do programa começando a executá-lo.

O modo como o dispositivo vai trabalhar depende de muitos parâmetros, os mais importantes dos quais são a competência da pessoa que desenvolve o hardware e do programador que, com o seu programa, deve tirar o máximo do dispositivo.

CAPÍTULO 3 – O PIC 16F84

O **PIC 16F84** pertence a uma classe de microcontroladores de 8 bits, com uma arquitetura RISC (1). A estrutura genérica é a do mapa da figura 3.1, que nos mostra os seus blocos básicos:

1. **Memória de programa (FLASH)** - para armazenar o programa que se escreveu, como ela pode ser programada e limpa mais de uma vez, ela é adequada para o desenvolvimento desses dispositivos.

2. **EEPROM** - memória dos dados que necessitam ser armazenados quando a alimentação é desligada. Normalmente é usada para guardar dados importantes que não podem ser perdidos quando a alimentação é cortada de repente.

3. **RAM** - memória de dados usada por um programa, durante a sua execução. Na RAM, são guardados todos os resultados intermediários ou dados temporários durante a execução do programa e que não são necessários serem mantidos após a sua execução.

4. **Portas de E/S (PORT A e PORT B)** - são ligações físicas entre o microcontrolador e o mundo exterior. O port A tem cinco pinos e o port B oito pinos.

5. **CONTADOR/TEMPORIZADOR** - é um registro de 8 bits no interior do microcontrolador que trabalha independente do programa.

6. **UNIDADE DE PROCESSAMENTO CENTRAL** - faz a conexão com todos os outros blocos do microcontrolador. Ele coordena o trabalho dos outros blocos e executa o programa.

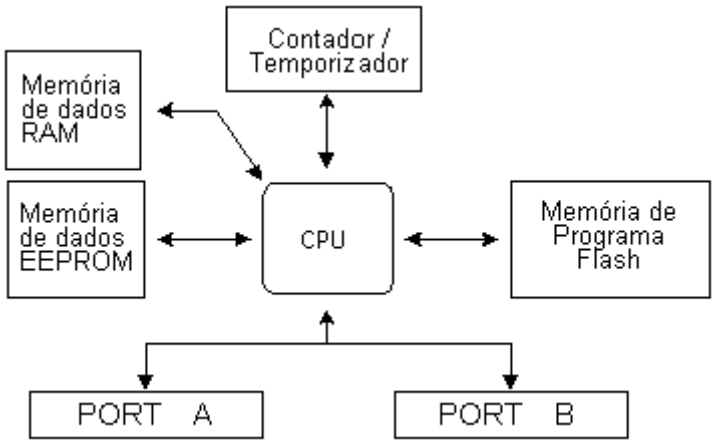


Figura 3.1 – Esquema do Microcontrolador PIC 16F84

3.1 – CISC, RISC

Já foi dito que o PIC16F84 tem uma arquitetura RISC. A arquitetura de Harvard é um conceito mais recente que a de Von-Neumann. Ela surgiu da necessidade de se fazer com que o microcontrolador trabalhasse mais rápido (2). Na arquitetura de Harvard, a memória de dados está separada da memória de programa (figura 3.2). Assim, é possível ter um maior fluxo de dados através da unidade central de processamento e, claro, uma maior velocidade de funcionamento. A separação da memória de dados da memória de programa, faz com que as instruções possam ser representadas por palavras de mais que 8 bits. O PIC16F84, usa 14 bits para cada instrução, o que permite que todas as instruções ocupem uma só palavra de instrução.

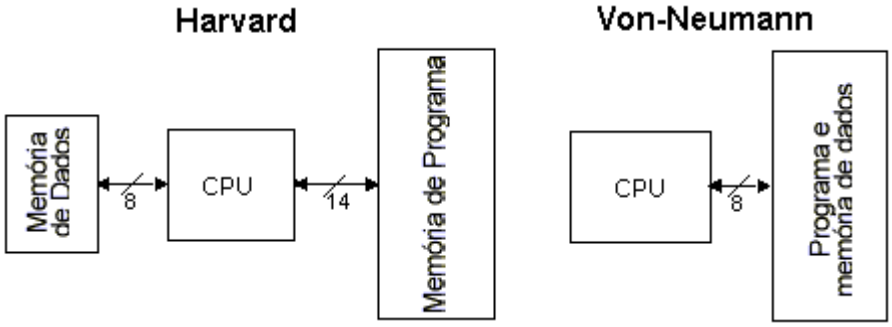


Figura 3.2 – Arquitetura Harvard x Von Neumann

Os microcontroladores com a arquitetura Harvard, são também designados por "microcontroladores RISC". RISC provém de Computador com um Conjunto Reduzido de Instruções (*Reduced Instruction Set Computer*). Os microcontroladores com uma arquitetura Von-Neumann são designados por 'microcontroladores CISC'. O nome CISC deriva de Computador com um Conjunto Complexo de Instruções (*Complex Instruction Set Computer*).

O PIC16F84 é um microcontrolador RISC, porisso possui um número reduzido de instruções, mais precisamente 35 (por exemplo, os microcontroladores da Intel e da Motorola têm mais de cem instruções). Todas estas instruções são executadas num único ciclo, exceto no caso de instruções de salto e de ramificação (1). De acordo com o que o seu fabricante refere, o PIC16F84 geralmente atinge resultados de 2 para 1 na compressão de código e 4 para 1 na velocidade, em relação aos outros microcontroladores de 8 bits da sua classe.

3.2 – Aplicações

O PIC16F84, é adequado para muitas variedades de aplicações, como a indústria automobilística, sensores remotos, fechaduras elétricas e dispositivos de segurança (3). É também um dispositivo ideal para cartões inteligentes, bem como para

dispositivos alimentados por baterias, devido ao seu baixo consumo. A memória EEPROM, faz com que se torne mais fácil usar microcontroladores em dispositivos onde o armazenamento permanente de vários parâmetros, seja necessário (códigos para transmissores, velocidade de um motor, freqüências de recepção, etc.). O baixo custo, baixo consumo, facilidade de manuseio e flexibilidade fazem com que o PIC16F84 possa ser utilizado em áreas que os microcontroladores não eram anteriormente empregados (exemplo: funções de temporização, substituição de interfaces em sistemas de grande porte, aplicações de pré-processamento, etc.).

3.3 – Relógio / Ciclo de Instrução

O relógio (*clock*), é quem dá o sinal de partida para o microcontrolador e é obtido a partir de um componente externo chamado “oscilador”. O clock do oscilador, é ligado ao microcontrolador através do pino OSC1, aqui, o circuito interno do microcontrolador divide o sinal de clock em quatro fases, Q1, Q2, Q3 e Q4 que não se sobrepõem. Estas quatro pulsações fazem um ciclo de instrução ou ciclo de máquina e durante o qual uma instrução é executada.

No diagrama da figura 3.3 é observada a relação entre o ciclo de instrução e o clock do oscilador (OSC1) assim como as fases Q1-Q4. O contador de programa (Program Counter ou PC) guarda o endereço da próxima instrução que será executada.

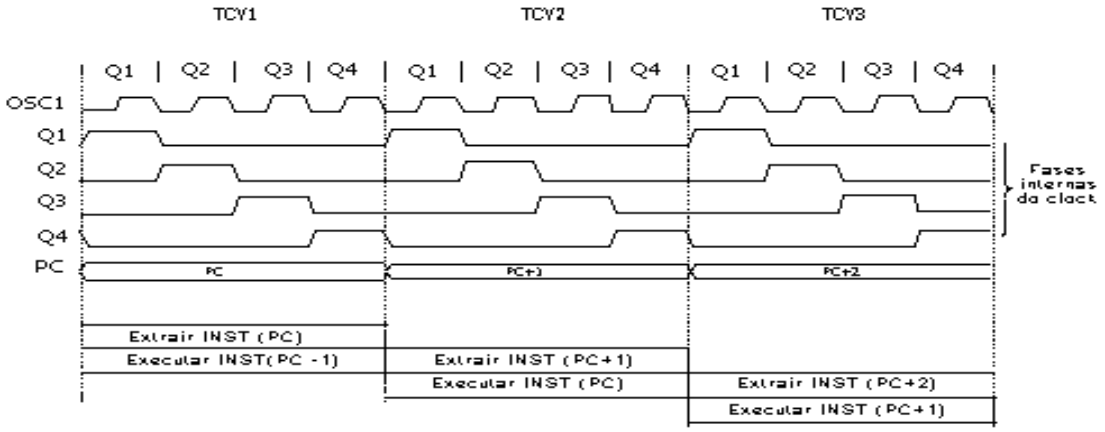


Figura 3.3 – Relação entre o Ciclo e o Clock

3.4 – Pipelining

Cada ciclo de instrução inclui as fases Q1, Q2, Q3 e Q4. A extração do código de uma instrução da memória de programa, é feita num ciclo de instrução, enquanto que a sua decodificação e execução, são feitos no ciclo de instrução seguinte (1). Contudo, devido à sobreposição – *pipelining* (o microcontrolador ao mesmo tempo em que executa uma instrução extrai simultaneamente da memória o código da instrução seguinte), pode-se considerar que, cada instrução demora um ciclo de instrução a ser executada. No entanto, se a instrução provocar uma mudança no conteúdo do contador de programa (PC), ou seja, se o PC não tiver que apontar para o endereço seguinte na memória de programa, mas sim para outro deverá considerar-se que a execução desta instrução demora dois ciclos. Isto acontece, porque a instrução vai ter que ser processada novamente, desta vez a partir do endereço correto.

3.5 – Pinagem

O PIC16F84 possui 18 pinos. É encontrado freqüentemente num tipo de encapsulamento DIP18, mas, também pode ser encontrado numa cápsula SMD de menores dimensões que a DIP. DIP é uma abreviatura para *Dual In Package* (Empacotamento em duas linhas). SMD é uma abreviatura para *Surface Mount Devices* (Dispositivos de Montagem em Superfície), o que sugere que os pinos não precisam passar pelos orifícios da placa em que são inseridos, quando se solda este tipo de componente.

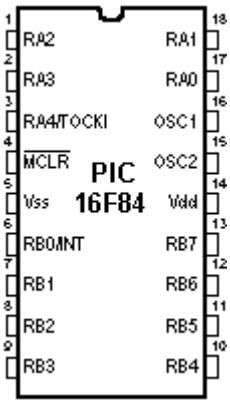


Figura 3.4 – Pinagem

Os pinos no microcontrolador PIC16F84 (figura 3.4), têm o seguinte significado:

Pino nº 1, **RA2** Segundo pino do port A. Não tem nenhuma função adicional.

Pino nº 2, **RA3** Terceiro pino do port A. Não tem nenhuma função adicional.

Pino nº 3, **RA4** Quarto pino do port A. O TOCK1 que funciona como entrada do temporizador, também utiliza este pino.

Pino nº 4, **MCLR** Entrada de reset e entrada da tensão de programação Vpp do microcontrolador .

Pino nº 5, **Vss** massa da alimentação.

Pino nº 6, **RB0** bit 0 do port B. Tem uma função adicional que é a de entrada de interrupção.

Pino nº 7, **RB1** bit 1do port B. Não tem nenhuma função adicional.

Pino nº 8, **RB2** bit 2 do port B. Não tem nenhuma função adicional.

Pino nº 9, **RB3** bit 3 do port B. Não tem nenhuma função adicional.

Pino nº 10, **RB4** bit 4 do port B. Não tem nenhuma função adicional.

Pino nº 11, **RB5** bit 5 do port B. Não tem nenhuma função adicional.

Pino nº 12, **RB6** bit 6 do port B. No modo de programa é a linha de clock

Pino nº 13, **RB7** bit 7 do port B. Linha de dados no modo de programa

Pino nº 14, **Vdd** pólo positivo da tensão de alimentação.

Pino nº 15, **OSC2** para ser ligado a um oscilador.

Pino nº 16, **OSC1** para ser ligado a um oscilador.

Pino nº 17, **RA0** bit 0 do port A. Sem função adicional.

Pino nº 18, **RA1** bit 1 do port A. Sem função adicional.

3.6 – Gerador de Clock – Oscilador

O circuito do oscilador é usado para fornecer um relógio (*clock*), ao microcontrolador. O clock é necessário para que o microcontrolador possa executar um programa ou as instruções de um programa (2).

O PIC16F84 pode trabalhar com quatro configurações de oscilador. Uma vez que as configurações com um oscilador de cristal e resistência-condensador (RC) são as mais utilizadas. Quando o oscilador é de cristal, a designação da configuração é de XT, se o oscilador for uma resistência em série com um condensador, tem a designação RC. A importância deve-se há necessidade de optar entre os diversos tipos de oscilador, quando se escolhe um microcontrolador.

3.7 – Reset

O reset é usado para pôr o microcontrolador num estado conhecido, pois às vezes o microcontrolador pode comportar-se de modo inadequado em determinadas condições indesejáveis (6). De modo que o seu funcionamento normal seja restabelecido, é preciso fazer o reset do microcontrolador, isto significa que todos os seus registros vão conter valores iniciais pré-definidos, correspondentes a uma posição inicial. O reset é usado quando o microcontrolador não se comporta da maneira esperada, mas também pode ser usado, quando ocorre uma interrupção por parte de outro dispositivo, ou quando se quer que o microcontrolador esteja pronto para executar um programa.

As formas de reset:

- Reset quando se liga a alimentação, POR (*Power-On Reset*).
- Reset durante o funcionamento normal, quando se põe a nível lógico baixo o pino MCLR do microcontrolador.
- Reset durante o regime de SLEEP (dormir).
- Reset quando o temporizador do watchdog (WDT) transborda (passa para 0 depois de atingir o valor máximo).

- Reset quando o temporizador do watchdog (WDT) transborda estando no regime de SLEEP.

3.8 – Unidade Central de Processamento

A unidade central de processamento (CPU) é o cérebro de um microcontrolador. Essa parte é responsável por extrair a instrução, decodificá-la e executá-la.

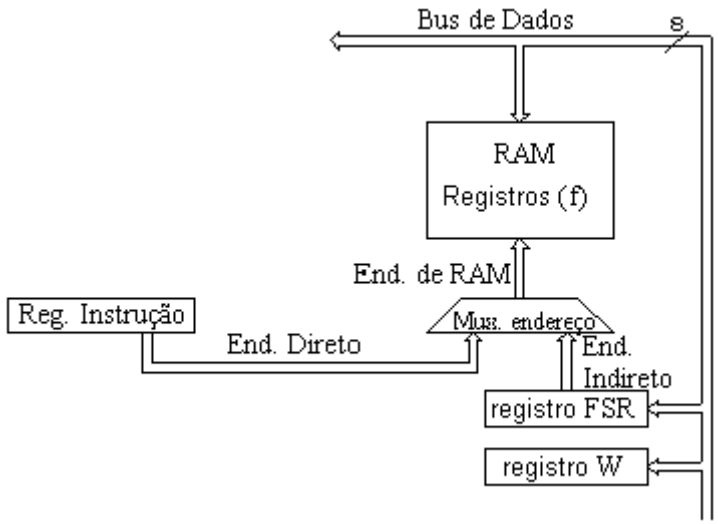


Figura 3.5 – Esquema da unidade central de processamento - CPU

A unidade central de processamento, interliga todas as partes do microcontrolador (figura 3.5) (10). Uma das suas funções mais importantes é decodificar as instruções do programa. Quando um programa é escrito, as instruções assumem um significado claro, porém, para que um microcontrolador possa interpretá-las, esta forma escrita de uma instrução tem que ser traduzida numa série de zeros e uns que é o “opcode” (*operation code* ou código da operação). Esta tradução de uma palavra escrita para a forma binária é executada por tradutores assembler (ou simplesmente assembler). O código da instrução extraído da memória de programa, tem que ser decodificado pela unidade central de processamento (CPU).

A unidade lógica aritmética (ALU – *Arithmetic Logic Unit*), é responsável pela execução de operações de adição, subtração, deslocamento (para a esquerda ou para a direita dentro de um registro) e operações lógicas. Como é mostrado na figura 3.6, o PIC16F84 contém uma unidade lógica aritmética e registros de uso genérico, ambos de 8 bits cada um.

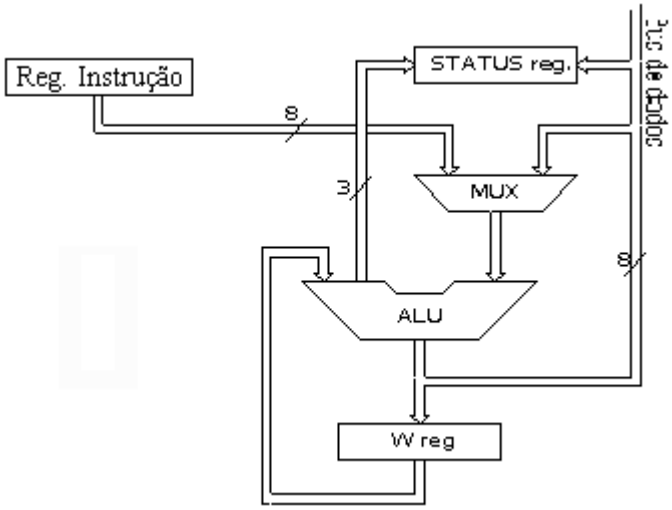


Figura 3.6 – Funcionamento da Unidade lógica-aritmética

3.9 – Portas (PORT'S)

Port, é um grupo de pinos do microcontrolador que podem ser habitados simultaneamente, no qual pode-se colocar uma combinação de zeros e uns ou ler um estado existente (11). Fisicamente, port é um registro dentro de um microcontrolador que está ligado por fios aos pinos do microcontrolador, eles representam a conexão física da Unidade Central de Processamento (CPU) com o mundo exterior e são utilizados para observar e/ou comandar outros componentes ou dispositivos. Para um aumento da sua funcionalidade, os mesmos pinos podem ter duas aplicações distintas, a escolha de uma destas duas funções é feita através

dos registros de configuração e ao se selecionar uma das funções, a outra é bloqueada automaticamente.

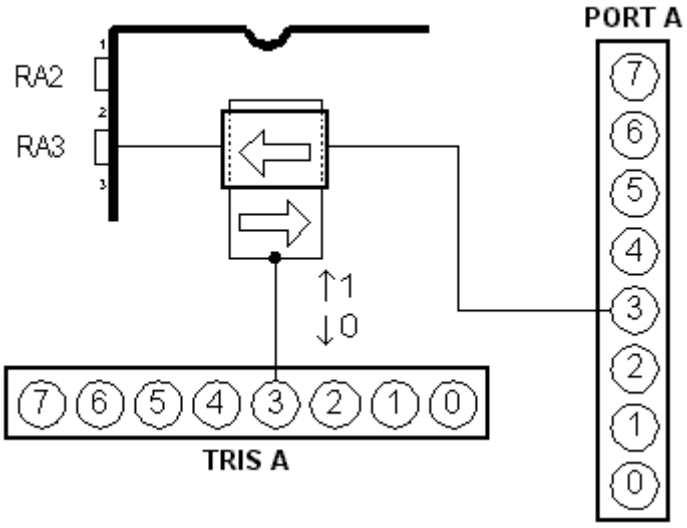


Figura 3.7 – Relação entre os registros TRIS A e PORT A

Todos os pinos dos port's podem ser definidos como entrada ou saída, de acordo com as necessidades do projeto (10). Para definir um pino como entrada ou como saída, é preciso, em primeiro lugar, escrever no registro TRIS, a combinação apropriada de zeros e uns (figura 3.7). Se no local apropriado de um registro TRIS for escrito o valor lógico "1", então o pino correspondente do port é definido como entrada, caso contrário, é definido como saída. Todos os port's, têm um registro TRIS associado, assim para o port A, existe o registro TRIS A no endereço 85h e, para o port B existe o registro TRIS B, no endereço 86h.

3.9.1 – Port A

O port A tem 5 pinos associados a ele. O registro de direção de dados chama-se TRIS A e tem o endereço 85h.

3.9.2 – Port B

O port B tem 8 pinos associados a ele. O registro de direção de dados chama-se TRIS B e tem o endereço 86h.

3.10 – Organização da Memória

O PIC16F84 tem dois blocos de memória separados, um para dados e o outro para o programa. A memória EEPROM e os registros de uso genérico (GPR) na memória RAM constituem o bloco para dados e a memória FLASH constitui o bloco de programa.

3.10.1 – Memória de programa

A memória de programa é implementada usando tecnologia FLASH, o que torna possível programar o microcontrolador muitas vezes antes de este ser instalado num dispositivo, e, mesmo depois da sua instalação, pode-se alterar o programa e parâmetros contidos. O tamanho da memória de programa é de 1024 endereços de palavras de 14 bits, destes, os endereços zero e quatro estão reservados respectivamente para o reset e para o vetor de interrupção.

3.10.2 – Memória de dados

A memória de dados compreende memória EEPROM e memória RAM. A memória EEPROM consiste em 64 posições para palavras de oito bits e cujos conteúdos não se perdem durante uma falha na alimentação. A memória EEPROM não faz parte diretamente do espaço de memória mas é acessada indiretamente através dos registros EEADR e EEDATA (figura 3.8) (10). A memória RAM para dados, ocupa um espaço no mapa de memória desde o endereço 0x0C até 0x4F, o que corresponde a 68 localizações. Os locais da memória RAM são também chamados registros GPR (*General Purpose Registers* = registros de uso genérico).

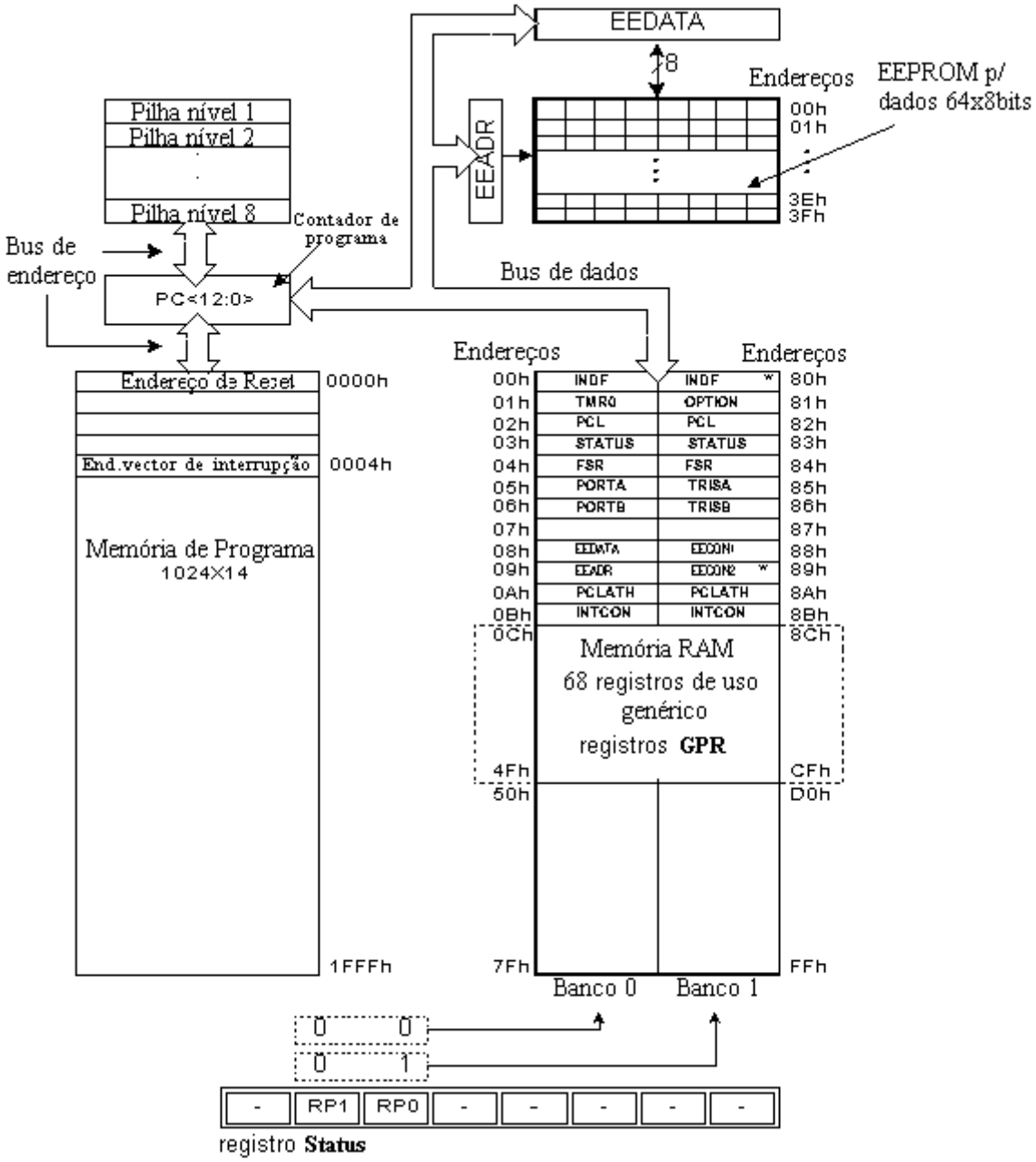


Figura 3.8 – Organização da memória no microcontrolador PIC16F84

3.10.3 – Contador de programa

O contador de programa (PC = *Program Counter*), é um registro de 13 bits que contém o endereço da instrução que vai ser executada.

3.10.4 – Pilha

O PIC16F84 tem uma pilha (*stack*) de 13 bits e 8 níveis de profundidade, o que corresponde a 8 locais de memória com 13 bits de largura. Sua função é guardar o valor do contador de programa quando ocorre um salto do programa principal para o endereço de um subprograma a ser executado.

3.10.5 – Modos de endereçamento

Os locais da memória RAM podem ser acessados direta ou indiretamente.

O endereçamento direto é feito através de um endereço de 9 bits. Este endereço obtém-se juntando aos sete bits do endereço direto de uma instrução, mais dois bits (RP1 e RP0) do registro STATUS, como é mostrado na figura 3.9:

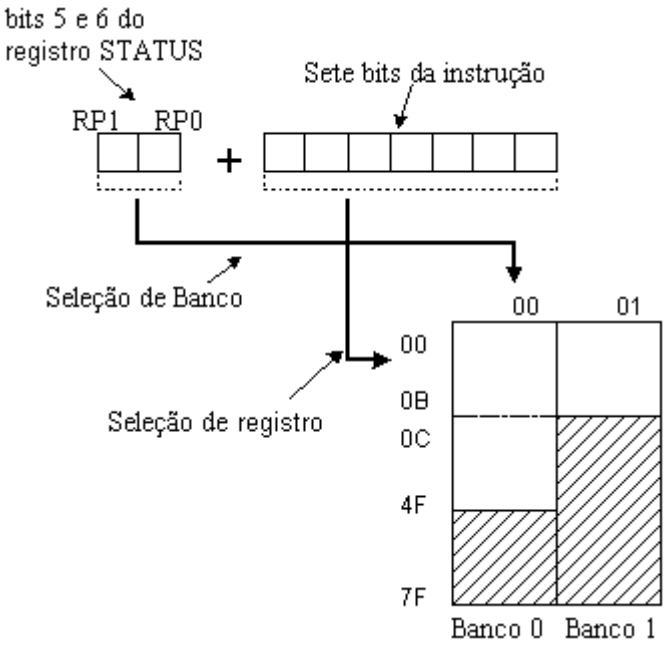


Figura 3.9 – Endereçamento direto

O endereçamento indireto, ao contrário do direto, não tira um endereço do código, mas faz com a ajuda do bit IRP do registro STATUS e do registro FSR (figura 3.10). O local endereçado é acessado através do registro INDF e coincide com o endereço contido em FSR. Por outras palavras, qualquer instrução que use INDF como registro, na realidade acessa aos dados apontados pelo registro FSR.

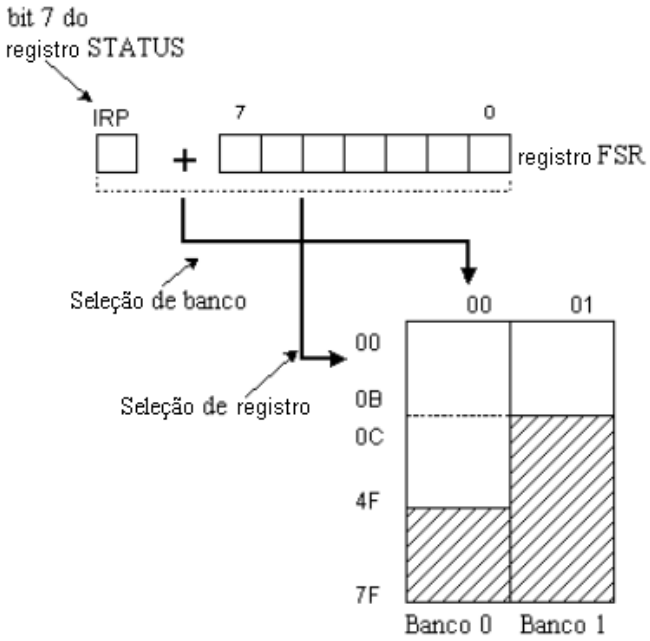


Figura 3.10 – Endereçamento indireto

3.11 – Interrupções

As interrupções são um mecanismo que o microcontrolador possui e que torna possível responder a alguns acontecimentos no momento em que eles ocorrem, mesmo que ele esteja executando uma tarefa no momento. Isto é muito importante, pois interliga o microcontrolador com o exterior. Geralmente, cada interrupção muda a direção de execução do programa, suspendendo a sua execução, enquanto o microcontrolador corre um subprograma que é a rotina de atendimento de interrupção. Depois deste subprograma ter sido executado, o microcontrolador continua com o programa principal, a partir do local em que havia parado (figura 3.11).

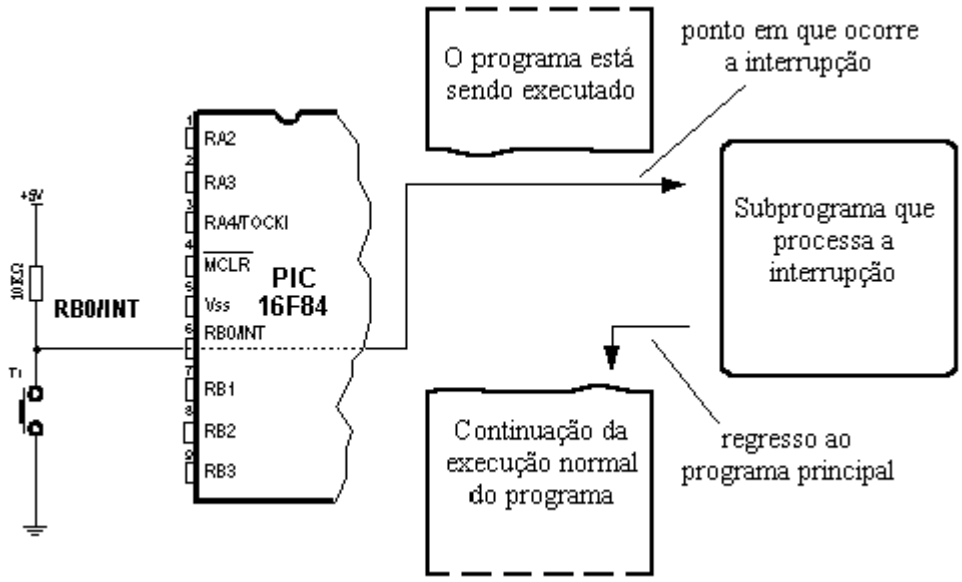


Figura 3.11 – Uma das possíveis fontes de interrupção e como afeta o programa principal

O PIC16F84 possui quatro fontes de interrupção:

- Fim de escrita na EEPROM
- Interrupção em TMR0 causada por transbordo do temporizador
- Interrupção por alteração nos pinos RB4, RB5, RB6 e RB7 do port B.
- Interrupção externa no pino RB0/INT do microcontrolador

De um modo geral, cada fonte de interrupção tem dois bits associados. Um habilita a interrupção e o outro assinala quando a interrupção ocorre. Existe um bit comum a todas as interrupções chamado GIE que pode ser usado para impedir ou habilitar todas as interrupções, simultaneamente.

3.11.1 – Interrupção por fim de escrita na EEPROM

Como escrever num endereço da EEPROM leva cerca de 10ms (o que representa muito tempo quando se fala de um microcontrolador), não é recomendável que se deixe o microcontrolador um grande intervalo de tempo sem fazer nada, à espera do fim da operação da escrita. Assim, existe um mecanismo de interrupção que permite ao microcontrolador continuar executando o programa principal enquanto escreve na EEPROM. Quando esta operação de escrita termina, uma interrupção informa o microcontrolador.

3.11.2 – Interrupção devido ao transbordar (*overflow*) do contador TMR0

O transbordar do contador TMR0 (passagem de FFh para 00h) vai colocar ‘1’ no bit TOIF (INTCON<2>), Esta é uma interrupção muito importante, uma vez que, muitos problemas podem ser resolvidos utilizando esta interrupção. Um exemplo é o da medição de tempo. Se tiver conhecimento de quanto tempo o contador precisa para completar um ciclo de 00h a FFh, então, o número de interrupções multiplicado por esse intervalo de tempo, dá o tempo total decorrido. Na rotina de interrupção uma variável guardada na memória RAM vai sendo incrementada, o valor dessa variável multiplicado pelo tempo que o contador precisa para um ciclo completo de contagem, vai dar o tempo gasto.

3.11.3 – Interrupção por variação nos pinos 4, 5, 6 e 7 do port B

Uma variação em 4 bits de entrada do Port B (bits 4 a 7), põe a ‘1’ o bit RBIF (INTCON<0>). A interrupção ocorre quando os níveis lógicos em RB7, RB6, RB5 e RB4 do port B, mudam de ‘1’ para ‘0’ ou vice-versa. Para que estes pinos detectem as variações, eles devem ser definidos como entradas. Se qualquer um deles for definido como saída, nenhuma interrupção será gerada quando surgir uma variação do nível lógico. Se estes pinos forem definidos como entradas, o seu valor atual é comparado com o valor anterior, que foi guardado quando se fez a leitura anterior do port B.

3.11.4 – Interrupção externa no pino RB0/INT do microcontrolador

A interrupção externa no pino RB0/ INT é desencadeada por um impulso ascendente (se o bit INTEDG = 1 no registro OPTION<6>), ou por um impulso descendente (se INTEDG = 0). Quando o sinal correto surge no pino INT, o bit INTF do registro INTCON é colocado a ‘1’. O bit INTF (INTCON<1>) tem que ser reposto a ‘0’ na rotina de interrupção, afim de que a interrupção não possa voltar a ocorrer de novo, no regresso ao programa principal.

3.12 – Temporizador TMR0

Os temporizadores são as partes mais complexas de um microcontrolador. Por meio deles, é possível relacionar uma dimensão real que é o tempo, com uma variável que representa o estado de um temporizador dentro de um microcontrolador. Fisicamente, o temporizador é um registro cujo valor é continuamente incrementado até 255. Chegado a este número, ele reinicia o contador.

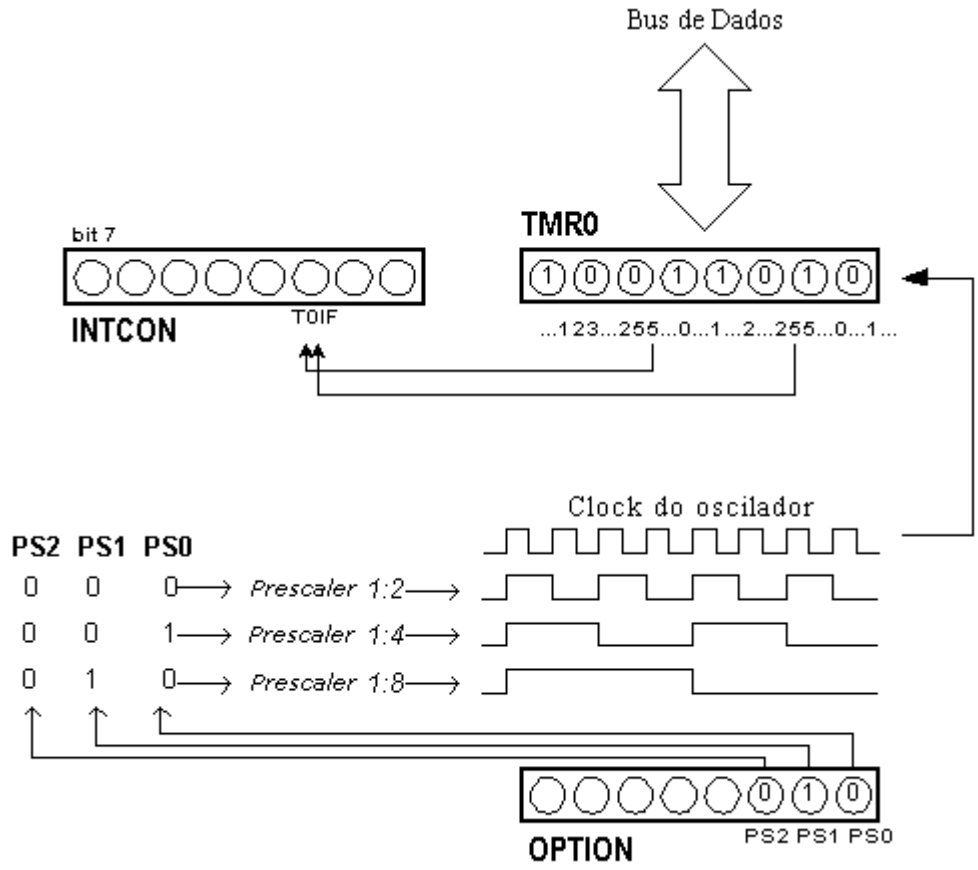


Figura 3.12 – Relação entre o temporizador TMR0 e o prescaler

O incremento do temporizador é feito simultaneamente com tudo o que o microcontrolador faz. Uma das maneiras é incrementar uma variável sempre que o microcontrolador passa de 255 para 0. Caso tenha-se conhecimento de quanto tempo um temporizador precisa para efetuar uma contagem completa (de 0 a 255), basta multiplicar o valor da variável por esse tempo, e obter o tempo total decorrido.

O PIC16F84, possui um temporizador de 8 bits. O número de bits determina a quantidade de valores diferentes que a contagem pode assumir, antes de voltar novamente para zero. No caso de um temporizador de 8 bits esse valor é 256. Um esquema simplificado da relação entre um temporizador e um prescaler está representado no diagrama da figura 3.12 (10). Prescaler é a designação para a parte do microcontrolador que divide a freqüência de oscilação do clock antes que os respectivos impulsos possam incrementar o temporizador. O número pelo qual a freqüência de clock é dividida, está definido nos três primeiros bits do registro OPTION. O maior divisor possível é 256. Neste caso, significa que só após 256

impulsos de clock é que o conteúdo do temporizador é incrementado de uma unidade. Isto permite medir grandes intervalos de tempo.

O prescaler tanto pode ser atribuído ao temporizador TMR0, como ao watchdog. O watchdog é um mecanismo que o microcontrolador usa para se defender contra "estouros" do programa. O microcontrolador também pode ter problemas com o seu programa. Quando isto ocorre, o microcontrolador pára de trabalhar e se mantém nesse estado até que ocorra um reset. Por causa disto, foi introduzido o mecanismo de watchdog (cão de guarda). Depois de um certo período de tempo, o watchdog faz o reset do microcontrolador (o que realmente acontece, é que o microcontrolador executa o reset de si próprio). O watchdog trabalha na base de um princípio simples: se o seu temporizador transbordar, é feito o reset do microcontrolador e este começa a executar de novo o programa a partir do princípio. Deste modo, o reset poderá ocorrer tanto no caso de funcionamento correto como no caso de funcionamento incorreto. O próximo passo é evitar o reset no caso de funcionamento correto, isso é feito escrevendo zero no registro WDT (instrução CLRWDT) sempre que este está próximo de transbordar. Assim, o programa irá evitar um reset enquanto está a funcionando corretamente. Se ocorrer o "estouro" do programa, este zero não será escrito, haverá transbordo do temporizador WDT e irá ocorrer um reset que vai fazer com que o microcontrolador comece de novo a trabalhar corretamente.

3.13 – Memória de Dados EEPROM

O PIC16F84 tem 64 bytes de localizações de memória EEPROM, correspondentes aos endereços de 00h a 63h e onde se pode ler e escrever. A característica mais importante desta memória é de não perder o seu conteúdo quando a alimentação é desligada. Sem alimentação, estes dados permanecem no microcontrolador durante

mais de 40 anos (especificações do fabricante do microcontrolador PIC16F84), além disso, esta memória suporta até 10000 operações de escrita.

Na prática, a memória EEPROM é usada para guardar dados importantes ou alguns parâmetros de processamento.

A memória EEPROM é colocada num espaço de memória especial e pode ser acessada através de registros especiais. Estes registros são:

- **EEDATA** no endereço 08h, que contém o dado lido ou aquele que se quer escrever.

- **EEADR** no endereço 09h, que contém o endereço do local da EEPROM que vai ser acessado.

- **EECON1** no endereço 88h, que contém os bits de controle.

- **EECON2** no endereço 89h. Este registro não existe fisicamente e serve para proteger a EEPROM de uma escrita acidental.

3.13.1 – Lendo a memória EEPROM

Colocando ‘1’ no bit RD inicia-se a transferência do dado do endereço guardado no registro EEADR para o registro EEDATA. Como para ler os dados não é preciso tanto tempo como para escrevê-los, os dados extraídos do registro EEDATA já podem ser usados na instrução seguinte.

3.13.2 – Escrevendo na memória EEPROM

Para escrever dados num local da EEPROM, o programador tem primeiro que endereçar o registro EEADR e introduzir a palavra de dados no registro EEDATA. A seguir, deve-se colocar ‘1’ no bit WR, o que faz desencadear o processo. O bit WR deverá ser posto a ‘0’ e o bit EEIF será posto a ‘1’ a seguir à operação de escrita, o que pode ser usado no processamento de interrupções. Os valores 55h e AAh são a primeira e segunda chaves que tornam impossível que ocorra uma escrita acidental na EEPROM. Estes dois valores são escritos em EECON2 que serve apenas para isto, ou seja, para receber estes dois valores e assim prevenir contra uma escrita acidental na memória EEPROM.

CAPÍTULO 4 – CONJUNTO DE INSTRUÇÕES

Já foi dito que um microcontrolador não é como qualquer outro circuito integrado. Quando saem da cadeia de produção, a maioria dos circuitos integrados, estão prontos para serem inseridos nos dispositivos, o que não é o caso dos microcontroladores. Para que um microcontrolador cumpra a sua tarefa, é necessário lhe dizer exatamente o que fazer, ou, por outras palavras, é necessário escrever o programa que o microcontrolador vai executar. Neste capítulo serão descritas as instruções que constituem o assembler, ou seja, a linguagem de baixo nível para os microcontroladores PIC.

4.1 – Conjunto de Instruções da Família PIC 16CXX de Microcontroladores

O conjunto completo compreende 35 instruções que será mostrado mais adiante. Uma razão para este pequeno número de instruções resulta principalmente do fato de se estar falando de um microcontrolador RISC cujas instruções foram otimizadas tendo em vista a rapidez de funcionamento, simplicidade de arquitetura e compactação de código (1). O único inconveniente, é que o programador tem que dominar a técnica para desenvolver o programa com apenas 35 instruções.

4.2 – Transferência de Dados

A transferência de dados num microcontrolador, ocorre entre o registro de trabalho (W) e um registro ‘f’ que representa um local qualquer, de memória na RAM interna (sendo um registro especial ou um registro de uso genérico).

As primeiras três instruções referem-se à escrita de uma constante no registro W (MOVLW é uma abreviatura para **MOV**a **L**iteral para **W**), à cópia de um dado do registro W na RAM e à cópia de um dado de um registro da RAM no registro W (ou nele próprio, caso em que apenas a flag do zero é afetada) . A instrução CLRF escreve a constante 0 no registro ‘f’ e CLRW escreve a constante 0 no registro W. A instrução SWAPF troca o *nibble* (conjunto de 4 bits) mais significativo com o *nibble* menos significativo de um registro, passando o primeiro a ser o menos significativo e o outro o mais significativo do registro.

4.3 – Lógicas e Aritméticas

De todas as operações aritméticas possíveis, os microcontroladores PIC, tal como a grande maioria dos outros microcontroladores, apenas suportam a subtração e a adição. Os bits ou flags C, DC e Z, são afetados conforme o resultado da adição ou subtração, com uma única exceção: uma vez que a subtração é executada como uma adição com um número negativo, a flag C (*Carry*), comporta-se inversamente no que diz respeito à subtração. Em outras palavras, é colocado ‘1’ se a operação é possível e ‘0’ se um número maior tiver que ser subtraído de outro menor.

A lógica dentro do PIC tem a capacidade de executar as operações AND, OR, EX-OR, complemento (COMF) e rotações (RLF e RRF). Estas últimas instruções, rodam o conteúdo do registro através desse registro e da flag C de uma casa para a esquerda (na direção do bit 7), ou para a direita (na direção do bit 0). O bit que sai do registro é escrito na flag C e o conteúdo anterior desta flag, é escrito no bit situado do lado oposto no registro.

4.4 – Operações sobre Bit's

As instruções BCF e BSF põem '0' ou '1' em qualquer bit de qualquer local da memória. Apesar de parecer uma operação simples, ela é executada do seguinte modo, a CPU primeiro lê o byte completo, altera o valor de um bit e, a seguir, escreve o byte completo no mesmo local.

4.5 – Direção de Execução de um Programa

As instruções GOTO, CALL e RETURN são executadas do mesmo modo que em todos os outros microcontroladores, a diferença é que a pilha é independente da RAM interna e é limitada a oito níveis (5). A instrução 'RETLW k' é idêntica à instrução RETURN, exceto que, ao regressar de um subprograma, é escrita no registro W uma constante definida pelo operando da instrução. Esta instrução, permite implementar facilmente listagens. Na maioria das vezes, são utilizadas

determinando a posição do dado na tabela adicionando-a ao endereço em que a tabela começa e, então, é lido o dado nesse local (que está situado normalmente na memória de programa).

A tabela pode apresentar-se como um subprograma (figura 4.1) que consiste numa série de instruções ‘RETLW k’ onde as constantes ‘k’, são membros da tabela.

```
Main    movlw 2
        call Lookup
Lookup  addwf PCL, f
        retlw k
        retlw k1
        retlw k2
          :
          :
        retlw kn
```

Figura 4.1 – Subprograma

Escreve-se a posição de um membro da tabela no registro W e, usando a instrução CALL, o subprograma que contém a tabela é chamado. A primeira linha do subprograma ‘ADDWF PCL, f’, adiciona a posição na tabela e que está escrita em W, ao endereço do início da tabela e que está no registro PCL, assim, é obtido o endereço real do dado da tabela na memória de programa. Quando se regressa do subprograma, tem-se no registro W o conteúdo do membro da tabela endereçado. No exemplo anterior, a constante ‘k2’ estará no registro W, após o retorno do subprograma.

RETFIE (RETurn From Interrupt – Interrupt Enable ou regresso da rotina de interrupção com as interrupções habilitadas) é um regresso da rotina de interrupção e difere de RETURN pois automaticamente coloca ‘1’ no bit GIE (habilitação global das interrupções). Quando a interrupção começa, este bit é automaticamente recolocado a ‘0’. Também quando a interrupção tem início, somente o valor do contador de programa é colocado no topo da pilha. Não é fornecida uma capacidade automática de armazenamento do registro de estado.

Os saltos condicionais estão resumidos em duas instruções: BTFSC e BTFSS. De acordo com o estado lógico do bit do registro ‘f’ que está sendo testado, a instrução seguinte no programa é ou não executada.

4.6 – Período de Execução da Instrução

Todas as instruções são executadas num único ciclo, exceto as instruções de ramificação condicional se a condição for verdadeira, ou se o conteúdo do contador de programa for alterado pela instrução. Nestes casos, a execução requer dois ciclos de instrução e o segundo ciclo é executado como sendo um NOP (**N**enhuma **OP**eração). Quatro oscilações de clock perfazem um ciclo de instrução. Se estivermos usando um oscilador com 4MHz de freqüência, o tempo normal de execução de uma instrução será de 1ms e, no caso de uma ramificação condicional de 2ms.

4.7 – Listagem das Palavras

f Qualquer local de memória num microcontrolador

W Registro de trabalho

b Posição de bit no registro ‘f’

d Registro de destino

label Grupo de oito caracteres que marca o início de uma parte do programa (rótulo)

TOS Topo da pilha

[ ] Opcional

< > Grupo de bits num registro

4.8 – Algumas Instruções

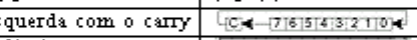
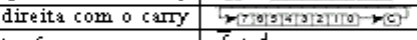
Menemónica		Descrição		Flag	CLK	Notas
		Transferência de dados				
MOVLW	k	Mova literal para W	k → W		1	
MOVWF	f	Mova W para f	W → f		1	
MOVF	f, d	Mova f	f → d	Z	1	1, 2
CLRW	-	Clear W (limpar W)	0 → W	Z	1	
CLRF	f	Clear f (limpar f)	0 → f	Z	1	2
SWAPF	f, d	Swap nibbles in f (trocar)	f(7:4),(3:0)→f(3:0),(7:4)		1	1, 2
		Lógicas e Aritméticas				
ADDLW	k	Adicionar literal a W	W+k → W	C, DC, Z	1	
ADDWF	f, d	Adicionar W a f	W+f → d	C, DC, Z	1	1, 2
SUBLW	k	Subtrair W de literal	k-W →W	C, DC, Z	1	
SUBWF	f, d	Subtrair W de f	f-W → d	C, DC, Z	1	1, 2
ANDLW	k	AND literal com W	W .AND. k → W	Z	1	
ANDWF	f, d	AND W com f	W .AND. f → d	Z	1	1, 2
IORLW	k	Inclusivo OR de literal com W	W .OR. k → W	Z	1	
IORWF	f, d	Inclusivo OR de W com f	W .OR. f → d	Z	1	1, 2
XORWF	f, d	Exclusivo OR de W com f	W . XOR. f → d	Z	1	1, 2
XORLW	k	Exclusivo OR de literal com W	W .XOR. k → W	Z	1	
INCF	f, d	Incrementar f	f+1 → f	Z	1	1, 2
DECF	f, d	Decrementar f	f-1 → f	Z	1	1, 2
RLF	f, d	Rode f p/ esquerda com o carry		C	1	1, 2
RRF	f, d	Rode f p/ a direita com o carry		C	1	1, 2
COMF	f, d	Complementar f	$\overline{f}$ → d	Z	1	1, 2
		Operações sobre bits				
BCF	f, b	Bit Clear f (bit de f a '0')	0 → f (b)		1	1, 2
BSF	f, b	Bit Set f (bit de f a '1')	1 → f (b)		1	1, 2
		Direccionamento do programa				
BTFSC	f, b	Bit Test f, Salte se Clear ('0')	salte se f(b)=0		1(2)	3
BTFSS	f, b	Bit Test f, Salte se Set ('1')	salte se f(b)=1		1(2)	3
DECFSZ	f, d	Decremente f, salte se der 0	f-1 → d, salte se der 0		1(2)	1, 2, 3
INCFSZ	f, d	Incremente f, salte se der 0	f+1 → d, salte se der 0		1(2)	1, 2, 3
GOTO	k	Go to address (Ir p/ endereço)	k → PC		2	
CALL	k	Chamar subrotina	PC → TOS, k → PC		2	
RETURN	-	Retorno de subrotina	TOS → PC		2	
RETLW	k	Retorno com literal em W	k → W, TOS → PC		2	
RETFIE	-	Retorno de interrupção	TOS → PC, 1 → GIE		2	
		Outras instruções				
NOP	-	Nenhuma operação			1	
CLRWDT	-	Temporizador do Watchdog=0	0 → WDT, 1 → TO, 1 → PD	$\overline{TO}$, $\overline{PD}$	1	
SLEEP	-	Entrar no modo 'sleep'	0 → WDT, 1 → TO, 0 → PD	$\overline{TO}$, $\overline{PD}$	1	

Figura 4.2 – Instruções

Notas:

1. Se o port de entrada/saída for o operando origem, é lido o estado dos pinos do microcontrolador.

2. Se esta instrução for executada no registro TMR0 e se d = 1, o prescaler atribuído a esse temporizador é automaticamente limpo.

3. Se o Contador de Programa for modificado ou se o resultado do teste for verdadeiro, a instrução é executada em dois ciclos.

CAPÍTULO 5 – PROGRAMAÇÃO EM LINGUAGEM ASSEMBLY

A capacidade de comunicação é muito importante nesta área. Contudo, isso só é possível se ambas as partes usarem a mesma linguagem, ou seja, se seguirem as mesmas regras para comunicação. Isto mesmo se aplica à comunicação entre os microcontroladores e o homem (5). A linguagem que o microcontrolador e o homem usam para comunicar entre si é designada por “linguagem assembly”. Os programas escritos em linguagem assembly devem ser traduzidos para uma “linguagem de zeros e uns” de modo que um microcontrolador a possa receber (figura 5.1) (10). “Linguagem assembly” e “assembler” são coisas diferentes. A primeira, representa um conjunto de regras usadas para escrever um programa para um microcontrolador e a outra, é um programa que corre num computador pessoal que traduz a linguagem assembly para uma linguagem de zeros e uns. Um programa escrito em “zeros” e “uns” diz-se que está escrito em “linguagem máquina”.

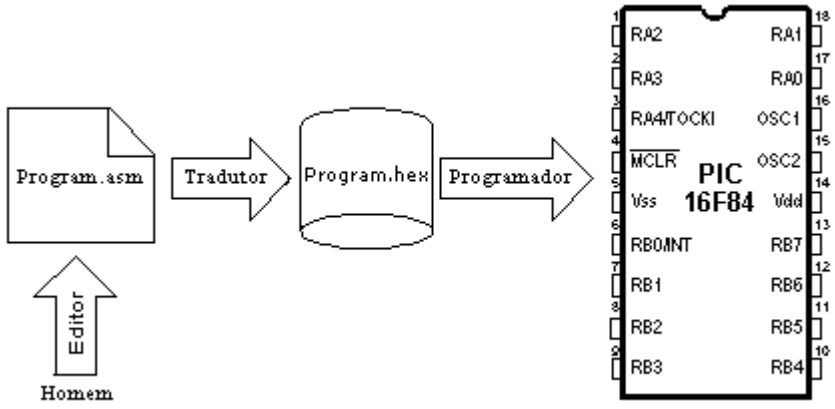


Figura 5.1 – O processo de comunicação entre o homem e o microcontrolador

Fisicamente, “**Programa**” representa um arquivo num disco de computador (ou na memória caso esteja-se lendo de um microcontrolador) e é escrito de acordo com as

regras do assembly ou qualquer outra linguagem de programação de microcontroladores. O homem pode entender a linguagem assembly já que ela é formada por símbolos alfabéticos e palavras. Ao escrever um programa, certas regras devem ser obedecidas para se chegar ao efeito esperado . Um **Tradutor** interpreta cada instrução escrita em linguagem assembly como uma série de zeros e uns com significado para a lógica interna do microcontrolador.

Por exemplo, a instrução “RETURN” que um microcontrolador utiliza para regressar de um subprograma.

Quando o assembler a traduz, obtém-se uma série de uns e zeros correspondentes a 14 bits que o microcontrolador sabe como interpretar.

Exemplo: RETURN 00 0000 0000 1000

Analogamente ao exemplo anterior, cada instrução assembly é interpretada na série de zeros e uns correspondentes.

O resultado desta tradução da linguagem assembly, é designado por um arquivo de “execução”. Muitas vezes encontra-se o nome de arquivo “HEX”. Este nome provém de uma representação hexadecimal desse arquivo, bem como o sufixo “hex" no título, por exemplo “correr.hex". Uma vez produzido, o arquivo de execução é inserido no microcontrolador através de um programador.

Um programa em **Linguagem Assembly** é escrito por intermédio de um processador de texto (editor) e é capaz de produzir um arquivo ASCII no disco de um computador ou em ambientes próprios como o MPLAB – que será explicado no próximo capítulo.

5.1 – Linguagem Assembly

Os elementos básicos da linguagem assembly são:

- Labels (rótulos)
- Instruções
- Operandos
- Diretivas
- Comentários

5.1.1 – Label

Um **Label** (rótulo) é uma designação textual (geralmente de fácil leitura) de uma linha num programa ou de uma seção de um programa para onde um microcontrolador deve saltar ou, ainda, o início de um conjunto de linhas de um programa. Também pode ser usado para executar uma ramificação de um programa (tal como Goto....), o programa pode ainda conter uma condição que deve ser satisfeita, para que uma instrução Goto seja executada. É importante que um rótulo (label) seja iniciado com uma letra do alfabeto ou com um traço baixo “_”. O comprimento de um rótulo pode ir até 32 caracteres. É também importante que o rótulo comece na primeira coluna, como é demonstrado na figura 5.2.

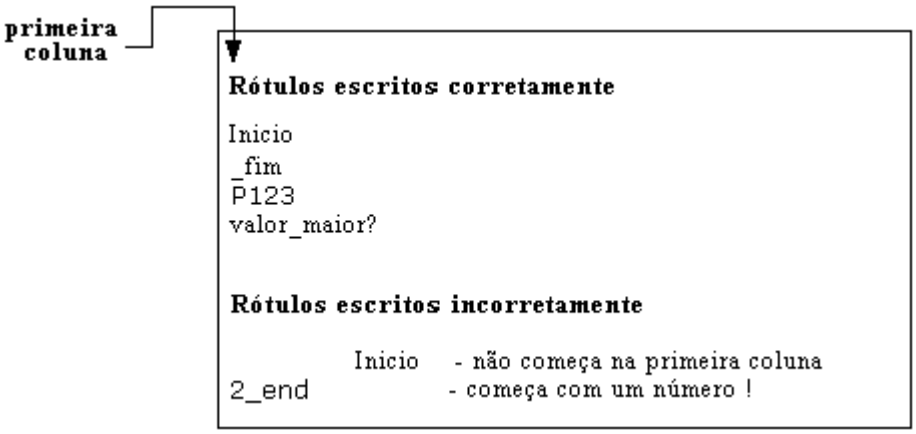


Figura 5.2 – Rótulos Escritos

5.1.2 – Instruções

As **Instruções** são específicas para cada microcontrolador, assim, quando se quer utilizar a linguagem assembly tem-se que estudar as instruções desse microcontrolador. O modo como se escreve uma instrução é designado por "sintaxe". Na figura 5.3, é possível reconhecer os erros de escrita, dado que as instruções movlp e gotto não existem no microcontrolador PIC16F84.

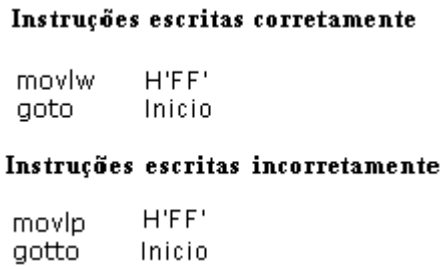


Figura 5.3 – Instruções Escritas

5.1.3 – Operandos

Os **Operandos** são os elementos da instrução necessários para que a instrução possa ser executada. Normalmente são **registros, variáveis e constantes** (figura 5.4). As constantes são designadas por literais (números).

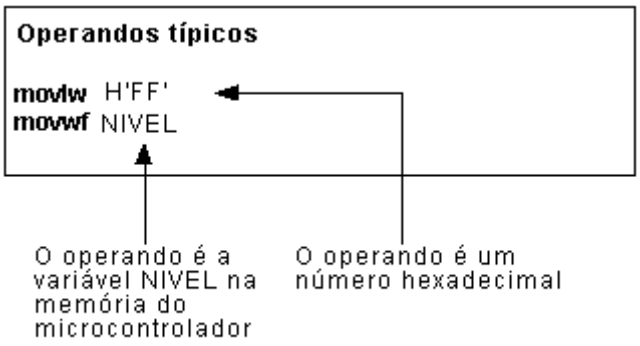


Figura 5.4 – Operandos

5.1.4 – Comentários

Comentário é um texto que o programador escreve com o objetivo de explicar o código, deixando mais claro e legível o programa. É colocado logo a seguir a uma instrução e deve começar com um ponto-e-vírgula ";".

5.1.5 – Diretivas

Uma **Diretiva** é parecida com uma instrução porém, é independente do tipo de microcontrolador e é uma característica inerente à própria linguagem assembly. As diretivas são compostas de variáveis ou registros para realizar determinados propósitos (figura 5.5).

```
Algumas diretivas usadas frequentemente:
        PROCESSOR 16F84
        #include    "p16f84.inc"

        __CONFIG  _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
```

Figura 5.5 – Diretivas

5.2 – Exemplo de como se Escreve um Programa

A figura 5.6 mostra como um programa simples pode ser escrito em linguagem assembly, respeitando regras básicas (10).

Quando se escreve um programa, além das regras fundamentais, existem princípios que, embora não obrigatórios é conveniente, serem seguidos. Um deles, é escrever no seu início, o nome do programa, aquilo que o programa faz, a versão deste, a data em que foi escrito, tipo de microcontrolador para o qual foi escrito e o nome do programador.

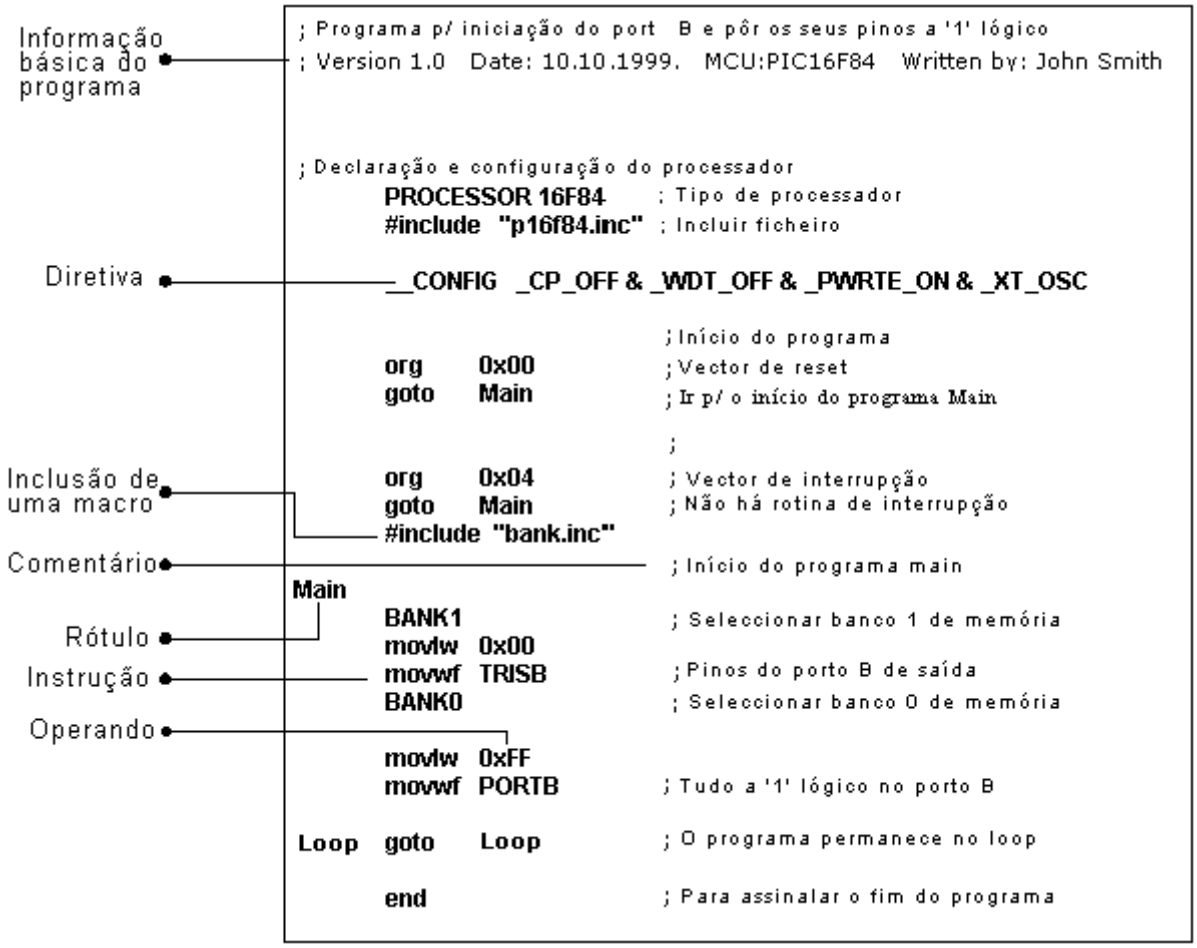


Figura 5.6 – Estrutura do Programa

Uma vez que estes dados não interessam ao tradutor de assembly, são escritos na forma de **comentários**. Vale lembrar que um comentário começa sempre com ponto e vírgula e pode ser colocado na linha seguinte ou logo a seguir à instrução. Depois deste comentário inicial ter sido escrito, deverão ser incluídas as diretivas.

Para que o seu funcionamento seja correto, é preciso definir vários parâmetros para o microcontrolador, tais como:

- Tipo de oscilador;
- Quando o temporizador do watchdog está ligado e
- Quando o circuito interno de reset está habilitado.

Tudo isto é definido na seguinte diretiva:

__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

5.3 – Diretivas de Controle

5.3.1– #DEFINE Troca de uma porção de texto por outra

Sintaxe: #define< nome > [< texto atribuído a nome >]

Descrição: Sempre que a palavra < nome > aparecer no programa, será substituída por < texto atribuído a nome >.

5.3.2 – INCLUDE Incluir um arquivo adicional num programa

Sintaxe: include <<nome_do_arquivo>>

include “<nome_do_arquivo>”

Descrição: A aplicação desta diretiva faz com que um arquivo completo seja copiado para o local em que a diretiva “include” se encontra. Se o nome do arquivo estiver entre aspas, trata-se de um arquivo do sistema, se não estiver entre aspas, mas sim entre os sinais < >, trata-se de um arquivo do utilizador.

5.3.3 – CONSTANT Atribui um valor numérico constante a uma designação textual

Sintaxe: constant < nome > = < valor >

Descrição: Cada vez que < nome > aparecer no programa, será substituído por < valor >.

5.3.4 – VARIABLE Atribui um valor numérico variável à designação textual

Sintaxe: variable < nome > = < valor >

Descrição: Ao utilizar esta diretiva, a designação textual muda o seu valor. Diferente da diretiva CONSTANT no fato de que, depois da diretiva ser aplicada, o valor da designação textual pode variar.

5.3.5 – SET Definir uma variável assembler

Sintaxe: < nome_variavel > set < valor >

Descrição: À variável < nome_variavel > é atribuída a expressão < valor >. A diretiva SET é semelhante a EQU, porém com ela é possível definir a variável com outro valor.

5.3.6 – EQU Definindo uma constante em assembler

Sintaxe: < nome_da_constante > equ < valor >

Descrição: Ao nome de uma constante < nome_de_constante > é atribuído um valor < valor >.

5.3.7 – ORG Define o endereço a partir do qual o programa é armazenado na memória do microcontrolador

Sintaxe: < rótulo > org < valor >

Descrição: Esta é a diretiva mais utilizada. Com esta diretiva é possível definir em que local na memória o programa começará.

5.3.8 – END Fim do programa

Sintaxe: end

Descrição: No fim do programa, é necessário colocar a diretiva 'end', para que o tradutor do assembly (assembler), saiba que não existem mais instruções no programa.

5.4 – Instruções Condicionais

5.4.1 – IF Ramificação condicional do programa

Sintaxe: if < termo_condicional >

Descrição: Se a condição em < termo_condicional > estiver satisfeita, a parte do programa que se segue à diretiva IF, deverá ser executada. Se a condição não for satisfeita, então é executada a parte que se segue às diretivas ELSE ou ENDIF.

5.4.2 – ELSE Assinala um bloco alternativo se a condição termo_condicional presente em 'IF' não se verificar

Sintaxe: Else

Descrição: Usado com a diretiva IF como alternativa no caso de termo_condicional ser falso.

5.4.3 – ENDIF Fim de uma seção condicional do programa

Sintaxe: endif

Descrição: Esta diretiva é escrita no fim de um bloco condicional, para informar o tradutor do assembly de que o bloco condicional terminou.

5.4.4 – WHILE A execução da seção do programa prossegue, enquanto a condição se verificar

Sintaxe: while < condição >

endw

Descrição: As linhas do programa situadas entre WHILE e ENDW devem ser executadas, enquanto a condição for verdadeira. Se a condição deixar de se verificar, o programa deverá executar as instruções a partir da linha que sucede a ENDW. O número de instruções compreendidas entre WHILE e ENDW pode ir até 100 e podem ser executadas até 256 vezes.

5.4.5 – ENDW Fim da parte condicional do programa

Sintaxe: endw

Descrição: Esta diretiva é escrita no fim do bloco condicional correspondente a WHILE, assim, o assembler fica sabendo que o bloco condicional chegou ao fim.

5.4.6 – IFDEF Executar uma parte do programa se um símbolo estiver definido

Sintaxe: ifdef < designação >

Descrição: Se a designação < designação > tiver sido previamente definida (normalmente através da diretiva #DEFINE), as instruções posteriores serão executadas até serem encontradas as diretivas ELSE ou ENDIF.

5.4.7 – IFNDEF Execução de uma parte do programa se o símbolo não tiver sido definido

Sintaxe: ifndef < designação >

Descrição: Se a designação < designação > não tiver sido previamente definida ou se esta definição tiver sido mandada ignorar através da diretiva #UNDEFINE, as instruções que se seguem deverão ser executadas, até que as diretivas ELSE ou ENDIF, sejam alcançadas.

5.5 – Diretivas de Dados

5.5.1 – CBLOCK Definir um bloco para as constantes nomeadas

Sintaxe: Cblock [< termo >]

< rótulo > [:< incremente >], < rótulo > [:< incremente >].....

endc

Descrição: Esta diretiva é usada para atribuir valores às constantes a seguir nomeadas. A cada termo seguinte, é atribuído um valor superior em uma unidade ao

anterior. No caso de < incremente > estar preenchido, então é o valor de < incremente > que é adicionado à constante anterior.

O valor do parâmetro < termo >, é o valor inicial. Se não for dado, então, por *default*, é considerado igual a zero.

5.5.2 – ENDC Fim da definição de um bloco de constantes

Sintaxe: endc

Descrição: Esta diretiva é utilizada no fim da definição de um bloco de constantes, para que o tradutor de assembly saiba que não há mais constantes.

5.5.3 – DB Definir um byte de dados

Sintaxe: [< termo >] db < termo > [, < termo >,.....,< termo >]

Descrição: Esta diretiva reserva um byte na memória de programa. Quando há mais termos a quem é preciso atribuir bytes, eles serão atribuídos um após outro.

5.5.4 – DE Definir byte na memória EEPROM

Sintaxe: [< termo >] de < termo > [, < termo >,.....,< termo >]

Descrição: Esta diretiva reserva um byte na memória EEPROM. Apesar de ser destinada em primeiro lugar para a memória EEPROM, também pode ser usada em qualquer outro local de memória.

5.5.5 – DT Definindo uma tabela de dados

Sintaxe: [< termo >] dt < termo > [, < termo >,.....,< termo >]

Descrição: Esta diretiva vai gerar uma série de instruções RETLW, uma instrução para cada termo.

5.6 – Configurando uma Diretiva

5.6.1 – __CONFIG Estabelecer os bits de configuração

Sintaxe: __config< termo > ou __config < endereço >, < termo >

Descrição: São definidos o tipo de oscilador, e a utilização do watchdog e do circuito de reset interno. Antes de usar esta diretiva, tem que se declarar o processador através da diretiva PROCESSOR.

5.6.2 – PROCESSOR Definindo o modelo de microcontrolador

Sintaxe: processor < tipo_de_microcontrolador >

Descrição: Esta diretiva, estabelece o tipo de microcontrolador em que o programa vai correr.

5.7 – Operadores Aritméticos de Assembler

Operador	Descrição	Exemplo
$	Valor actual do contador de programa	goto $ +3
(	Parêntesis esquerdo	1 + (d * 4)
)	Parêntesis direito	(Length + 1) * 256
!	NOT (complemento lógico)	if ! (a - b)
-	Complemento	flags = -flags
-	Negação (complemento p/ 2)	-1 * Length
high	Return o byte mais alto	movlw high CTR_Table
low	Return o byte - significativo	movlw low CTR_Table
*	Multiplicação	a = b * c
/	Divisão	a = b / c
%	Módulo	entry_len = tot_len % 16
+	Adição	tot_len = entry_len * 8 + 1
-	Subtracção	entry_len = (tot - 1) / 8
<<	Deslocamento p/ a esquerda	val = flags << 1
>>	Deslocamento p/ a direita	val = flags >> 1
>=	Maior ou igual	if entry_idx >= num_entries
>	Maior que	if entry_idx > num_entries
<	Menor que	if entry_idx < num_entries
<=	Menor que ou igual	if entry_idx <= num_entries
==	Igual	if entry_idx == num_entries
!=	Não igual	if entry_idx != num_entries
&	'E' bit a bit	flags = flags & ERROR_BIT
^	OU exclusivo bit a bit	flags = flags ^ ERROR_BIT
|	OU bit a bit	flags = flags | ERROR_BIT
&&	'E' lógico	if (len == 512) && (b == c)
||	'OU' lógico	if (len == 512) || (b == c)
=	Igual a	entry_index = 0
+=	Somar e igualar	entry_index += 1
-=	Subtrair e igualar	entry_index -= 1
*=	Multiplicar e igualar	entry_index *= entry_length
/=	Dividir e igualar	entry_total /= entry_length
%=	Igualar ao módulo	entry_index %= 8
<<=	Mover p/ esquerda e igual	flags <<= 3
>>=	Mover p/ direita e igual	flags >>= 3
&=	'E' lógico e igual	flags &= ERROR_FLAG
|=	'OU' bit a bit e igual	flags |= ERROR_FLAG
^=	OU exclusivo bit a bit e igual	flags^= ERROR_FLAG
++	Incrementar uma unidade	i ++
- -	Decrementar uma unidade	i --

Figura 5.7 – Operadores

5.8 – Arquivos criados ao Compilar um Programa

Os arquivos resultantes da tradução de um programa escrito em linguagem assembly são os seguintes:

- Arquivo de execução (nome_do_programa.hex);
- Arquivo de erros no programa (nome_do_programa.err);
- Arquivo de listagem (nome_do_programa.lst).

O primeiro arquivo contém o programa traduzido e que será introduzido no microcontrolador quando for programado (12). O conteúdo deste arquivo não dá grande informação ao programador.

O segundo arquivo contém erros possíveis que foram cometidos no processo de escrita e que foram notificados pelo assembler durante a tradução. Estes erros também são mencionados no arquivo de listagem “list”. No entanto é preferível utilizar este arquivo de erros “err”, em casos em que o arquivo “lst” é muito grande e, portanto, difícil de consultar.

O terceiro arquivo é o mais útil para o programador. Contém várias informações, como o posicionamento das instruções e variáveis na memória e a sinalização dos erros.

Na figura 5.8, apresenta-se o arquivo ‘list’ de um programa. No início de cada página, encontra-se informação acerca do nome do arquivo, data em que foi criado e número de página. A primeira coluna, contém o endereço da memória de programa, onde a instrução mencionada nessa linha, é colocada. A segunda coluna, contém os valores de quaisquer símbolos definidos com as diretivas: SET, EQU, VARIABLE, CONSTANT ou CBLOCK. A terceira coluna, tem, o código da instrução que o PIC irá executar. A quarta coluna contém instruções assembler e comentários do programador. Possíveis erros são mencionados entre as linhas, a seguir à linha em que o erro ocorreu.

Makro: Proba.lst

```
MPASM 02.40Released            PROBA.ASM   4-26-2000  7:18:17          PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT
  VALUE

                     00001   ;programa p/ iniciação do portoB e pôr os seus pinos
                     00002   ;no estado lógico '1'
                     00003   ;Version: 1.0  Date: 10.05.2000.      MCU: PIC16F84  Written
                     00004   ;by:  Petar Petrovic
                     00005
                     00006   ;Declaração e configuração do processador
                     00007   PROCESSOR 16F84         ;Tipo de processador
                     00008   #include "p16f84.inc"  ;Cabeçalho do processador
                     00001   LIST
                     00002   ;P16F84.INC  Standard Header File, Version 2.00  Microchip
                             ;Technology, Inc.
                     00136   LIST
                     00009
2007 3FF1            00010   __CONFIG    _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
                     00011
000C                 00012   CONSTANT BASE = 0x0c
                     00013
                     00014   ;Início do programa
0000                 00015   org  0x00       ;Vector de reset
0000 2805            00016   goto   Main     ;Ir p/ o início do programa Main
                     00017
                     00018   ;
0004                 00019   org      0x04   ;Vector de interrupção
0004    2805         00020   goto     Main   ;Não há rotina de interrupção
                     00021
                     00022 ;Início do programa main
                     00023   #include "Bank.inc"    ; Incluir ficheiro com macros
                     00001 ;*****************************************************
                     00002 ;              Macros BANK0 e BANK1
                     00003 ;*****************************************************
                     00004
0000    0010         00005   W_Temp          set     BASE+4
0000    0011         00006   Stat_Temp       set     BASE+5
0000    0012         00007   Option_Temp     set     BASE+6
                     00008
                     00009
                     00010   BANK0   macro
                     00011   bcf     STATUS,RP0      ; Seleccionar o banco 0 de memória
                     00012   endm
                     00013
                     00014   BANK1   macro
                     00015   bsf     STATUS,RP0      ; Seleccionar o banco 1 de memória
                     00016   endm
                     00017
0005                 00024   Main
                     00025   BANK1                   ; Seleccionar o banco 1 de memória
0005    1683    M    00026   bsf     STATUS,RP0      ; Seleccionar o banco 1 de memória
0006    3000         00026   movlw    0x00
Message[302]: Register in operand not in bank 0. Ensure that bank bits are
correct.
0007    0086         00027   movwf   TRISB           ;Pinos do porto B de saída
                     00028
                     00029   BANK0                   ;Seleccionar banco 0 de memória
0008    1283    M    00030   bcf     STATUS,RP0      ;Seleccionar banco 0 de memória
0009    30FF         00030   movlw   0xFF
000A    0086         00031   movwf   PORTB           ;Tudo a '1' lógico no porto B
                     00032
000B    280B         00033   Loop    goto    Loop    ;o programa permanece no loop
                     00034
                     00035   END             ;Para assinalar o fim do programa

MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : X---XXXXXXXX---- ---------------- ---------------- ----------------
2000 : -------X-------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:    9
Program Memory Words Free:    1015

Errors:         0
Warnings:       0 reported,      0 suppressed
Messages:       1 reported,      0 suppressed
```

Figura 5.8 – Arquivo List

No fim do arquivo de listagem, é apresentada uma tabela dos símbolos usados no programa. Uma característica útil do arquivo ‘list’ é a apresentação de um mapa da memória utilizada. Mesmo no fim, existe uma estatística dos erros, também contém uma indicação da memória de programa utilizada e da disponível.

5.9 – Macros

As macros são elementos muito úteis em linguagem assembly. Uma macro pode ser descrita em poucas palavras como “um grupo de instruções definido pelo utilizador que é acrescentado ao programa pelo assembler, sempre que a macro for invocada”. É possível escrever um programa sem usar macros. Porém, se for utilizada, o programa torna-se muito mais legível, especialmente se estiverem vários programadores trabalhando no mesmo programa. As macros têm afinidades com as funções nas linguagens de alto nível.

Sintaxe: < rótulo > macro

[< argumento1 >,< argumento2 >,.....,< argumentoN >]

.....

.....

endm

Descrição: Pelo modo como são escritas, as macros podem aceitar argumentos, o que também é muito útil em programação.

Quando o argumento é invocado no interior de uma macro, ele vai ser substituído pelo valor < argumentoN >.

Exemplo:

```
ON_PORTB        macro ARG1
                BANK0                ;Selecionar banco 0 de memória
                movlw ARG1           ;valor do argumento ARG1
                                     ;guardado no registro de trabalho
                movwf PORTB          ;valor do argumento ARG1
                                     ;guardado no Port B
                endm                 ;fim de macro
```

Figura 5.9 – Macro

A figura 5.9, mostra uma macro cujo propósito é enviar para o port B, o argumento ARG1, definido quando a macro foi invocada. Para utiliza-la num programa, basta escrever uma única linha: ON_PORTB 0xFF e, assim, colocar o valor 0xFF no port B. Para utilizar uma macro no programa, é necessário incluir o arquivo macro no programa principal, por intermédio da instrução #include “nome_da_macro.inc”. O conteúdo da macro é automaticamente copiado para o local em que esta macro está escrita. Isto pode ver-se melhor no arquivo ‘lst’ visto anteriormente, onde a macro é copiada abaixo da linha #include “bank.inc”.

CAPÍTULO 6 – MPLAB

O MPLAB é um pacote de programas que rodam no Windows e que torna mais fácil escrever ou desenvolver um programa, ou seja, é um ambiente de desenvolvimento para uma linguagem de programação standard com a finalidade de rodar num computador pessoal (PC) (12). Anteriormente, as operações incidiam sobre uma linha de instrução e continham um grande número de parâmetros, até que se introduziu o IDE "*Integrated Development Environment*" (Ambiente Integrado de Desenvolvimento) e as operações tornaram-se mais fáceis, utilizando o MPLAB (figura 6.1).

Figura 6.1 – MPLab

O MPLAB compreende várias partes:

- Agrupamento de todos os arquivos do mesmo projeto, num único projeto (*Project Manager*);
- Escrever e processar um programa (Editor de Texto);

- Simular o funcionamento no microcontrolador, do programa que se acabou de escrever (Simulador).

6.1 – O Ambiente de Trabalho

Como pode ser visto na figura 6.2, o aspecto do MPLAB é o mesmo da maioria dos programas Windows. Perto da área de trabalho existe um menu, *toolbar* e a linha de status no fundo da janela.

Assim, pretende-se seguir uma regra no Windows, que é tornar também acessíveis abaixo do menu, as opções usadas freqüentemente no programa,. Deste modo, é possível acessá-las de um modo mais fácil e tornar o trabalho mais rápido.

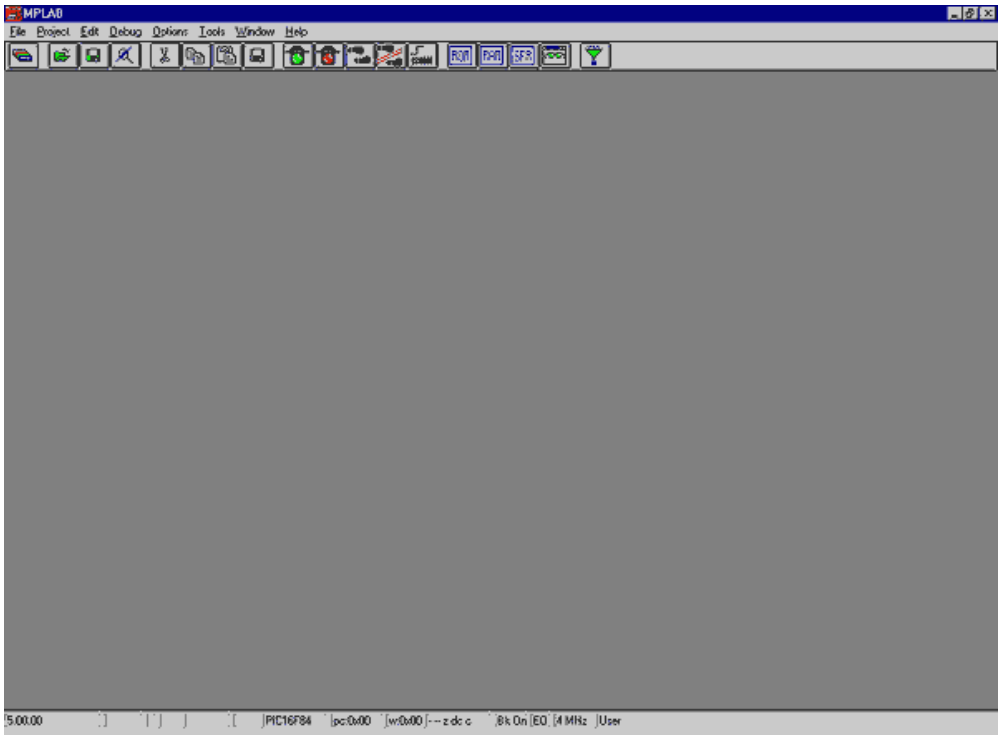


Figura 6.2 – Janela depois do MPLAB ser iniciado

6.2 – Simulador MPSIM

O simulador, é a parte do MPLAB que fornece a melhor visão interna do modo como o microcontrolador funciona (13). Através do simulador é possível monitorar os valores atuais das variáveis, os valores dos registros e os estados lógicos dos pinos dos port's. O simulador é de grande ajuda em programas complicados que incluem temporizadores, diferentes condições em que algo aconteça e outros requisitos semelhantes (especialmente com operações matemáticas). Como o microcontrolador executa as instruções uma a uma, o simulador é capaz de executar o programa passo a passo (linha a linha), mostrando o que acontece aos dados dentro do microcontrolador.

A primeira coisa que é preciso fazer numa situação real, é o reset do microcontrolador com o comando DEBUG > RUN > RESET, como é mostrado na figura 6.3. Este comando faz com que surja em negrito a linha de início do programa e que, o contador de programa contenha o valor zero como se pode verificar na linha de estado (pc: 0x00).

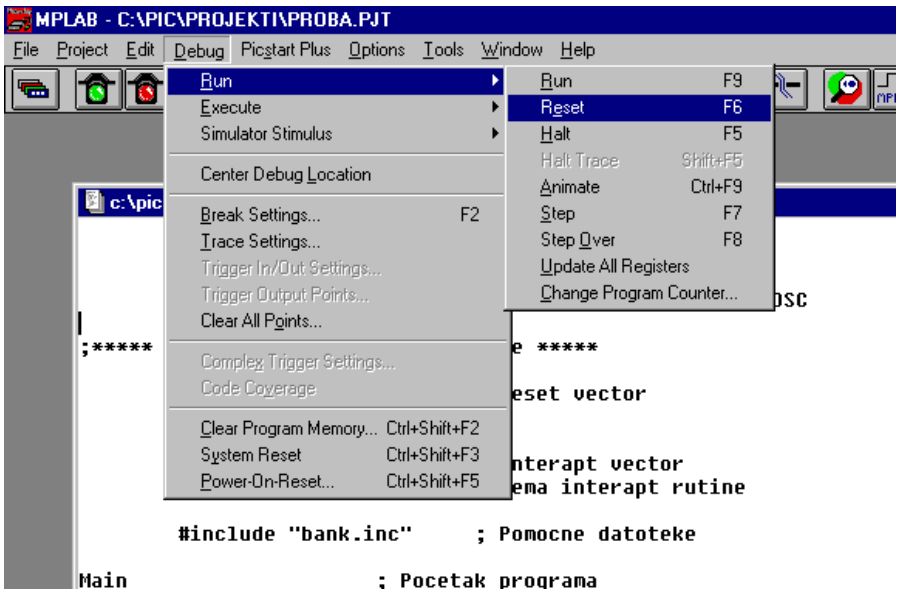


Figura 6.3 – O início da simulação do programa faz-se com o reset do microcontrolador

Uma das principais características de um simulador, é a possibilidade de observar o estado dos registros dentro do microcontrolador, assim como é observado na figura 6.4.

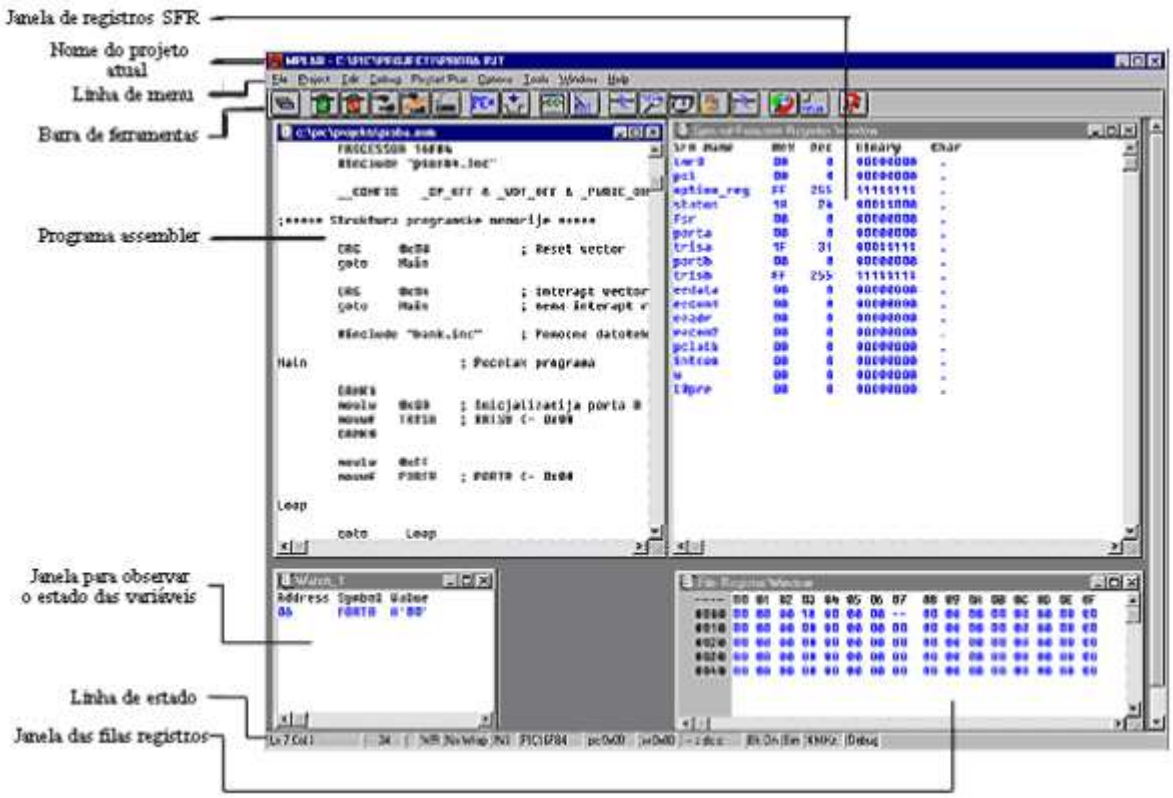


Figura 6.4 – Simulador com janelas abertas para registros SFR, filas registros e variáveis

O próximo comando num simulador é DEBUG>RUN>STEP que inicia a simulação passo a passo do programa

6.3 – Barra de ferramentas

Como o MPLAB tem vários componentes, cada um desses componentes tem a sua própria barra de ferramentas (*toolbar*). Contudo, existe uma barra de ferramentas que é uma espécie de resumo de todas as barras de ferramentas.

6.3.1 – Significado dos ícones na barra de ferramentas

	Sempre que se clica neste ícone, uma nova 'toolbar' aparece. Quando se clica quatro vezes seguidas, a barra atual reaparece.
	Ícone para abrir um projeto. O projeto aberto desta maneira contém todos os ajustes da janela e de todos os elementos que são cruciais para o projeto atual.
	Ícone para salvar um projeto. O projeto salvo armazena todos os ajustes de janelas e parâmetros. Quando é aberto o programa novamente, tudo retorna à janela, como quando o programa foi fechado.
	Para procurar uma parte do programa ou palavras, em programas maiores. Usando este ícone pode-se encontrar rapidamente uma parte do programa, rótulos, macros, etc.

	Ícone para cortar uma parte do texto. Este e os três ícones seguintes são standardisados em todos os programas que lidam com arquivos de texto. Como cada programa é representado por uma porção de texto, estas operações são muito úteis.
	Para copiar uma porção de texto. Existe uma diferença entre este e o ícone anterior. Quando se corta, tira-se uma parte de texto para fora da janela (e do programa), que poderá ser colado a seguir. Mas, tratando-se de uma operação de cópia, o texto é copiado mas não cortado e permanece visualizado na janela.
	Quando uma porção de texto é copiada ou cortada, ela move-se para uma parte da memória que serve para transferir dados no sistema operacional Windows. Mais tarde, clicando neste ícone, ela pode ser 'colada' num texto, no local onde se encontra o cursor.
	Salvar um programa (arquivo assembler).
	Começa a execução do programa em velocidade máxima. Essa execução, é reconhecida pelo aparecimento de uma linha de estado amarela. Nesta modalidade de execução, o simulador executa o programa a toda a velocidade até que é interrompido pelo clique no ícone de tráfego vermelho.
	Pára a execução do programa em velocidade máxima. Depois de clicar neste ícone, a linha de status torna-se de novo cinzenta e a execução do programa pode continuar passo a passo.
	Execução do programa passo a passo. Clicando neste ícone, começa-se a executar uma instrução na linha de programa a seguir à linha atual.

	‘Passar por cima’. Como um simulador não é mais que uma simulação da realidade por software, é possível passar por cima de algumas instruções e o programa prosseguir normalmente, depois disso. Normalmente, a parte do programa que interessa ao programador é a que se segue a esse ‘salto por cima’.
	Faz o reset do microcontrolador. Clicando neste ícone, o contador de programa é posicionado no início do programa e a simulação pode começar.
	Clicando neste ícone obtém-se uma janela com um programa, mas neste caso na memória de programa, onde se pode ver quais as instruções e em que endereços.
	Com a ajuda deste ícone, observa-se uma janela com o conteúdo da memória RAM do microcontrolador.
	Clicando neste ícone, aparece a janela dos registros SFR (registros com funções especiais). Como os registros SFR são usados em todos os programas, recomenda-se que, no simulador, esta janela esteja sempre ativa.
	Se um programa contiver variáveis cujos valores são necessários acompanhar (por exemplo um contador), então é necessária uma janela para elas, isso pode ser feito usando este ícone.
	Quando certos erros num programa se manifestam durante o processo de simulação, o programa tem que ser corrigido. Como o simulador usa o arquivo HEX como entrada, a seguir à correção dos erros, é preciso traduzir novamente o programa, de maneira que estas mudanças também sejam transferidas para o simulador. Clicando neste ícone, o projeto completo é de novo traduzido e, assim, obtém-se a nova versão do arquivo HEX para o simulador.

CAPÍTULO 7 – IMPLEMENTAÇÃO

Para demonstrar a funcionalidade de um microcontrolador, foi implementado um programa em Assembler para que o microcontrolador PIC 16F628A funcione como um letreiro, sendo acionado por um pêndulo em um bastão (figura 7.1), o mesmo “escreve” no ar (persistência retiniana) uma palavra previamente escrita em sua linha de código, ou seja, quando o pêndulo bate, a entrada do PIC vai para nível 0 e com isso ele dispara o processo que faz com que os led’s se acendam na seqüência correspondente, dentro de um curto intervalo de tempo.

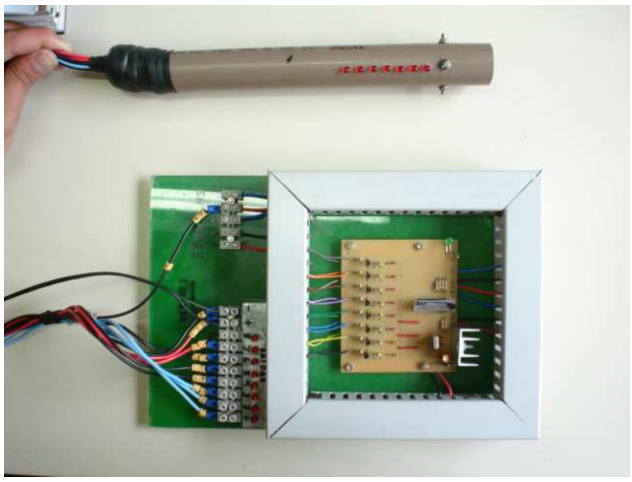


Figura 7.1 – Circuito do Bastão

Um trecho do programa implementado pode ser observado abaixo, onde o código descreve a palavra UNIPAC.

```
;LISTA DOS BITES DE SAIDA

CONVERT

   MOVF CONT,W        ;COLOCA O VALOR DO CONTADOR EM W
   ANDLW B'00111111' ;MASCARA O VALOR DE CONTADOR
   ADDWF PCL,F

         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'00000001'    ;TXT
         RETLW  B'00111111'    ;TXT
         RETLW  B'01000001'    ;TXT
         RETLW  B'01000000'    ;TXT
         RETLW  B'01000000'    ;TXT
         RETLW  B'01000000'    ;TXT
         RETLW  B'01000001'    ;TXT
         RETLW  B'00111111'    ;TXT
         RETLW  B'00000001'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'01000001'    ;TXT
         RETLW  B'01111111'    ;TXT
         RETLW  B'01000010'    ;TXT
         RETLW  B'00000100'    ;TXT
         RETLW  B'00001000'    ;TXT
         RETLW  B'00010000'    ;TXT
         RETLW  B'00100001'    ;TXT
         RETLW  B'01111111'    ;TXT
         RETLW  B'01000001'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'01000001'    ;TXT
         RETLW  B'01111111'    ;TXT
         RETLW  B'01000001'    ;TXT
         RETLW  B'00000000'    ;TXT
         RETLW  B'01000000'    ;TXT
         RETLW  B'01111111'    ;TXT
         RETLW  B'01001001'    ;TXT
         RETLW  B'00001001'    ;TXT
         RETLW  B'00001001'    ;TXT
         RETLW  B'00001001'    ;TXT
         RETLW  B'00001001'    ;TXT
         RETLW  B'00000110'    ;TXT
         RETLW  B'00000000'    ;TXT
```

```
        RETLW  B'01000000'    ;TXT
        RETLW  B'01111110'    ;TXT
        RETLW  B'01001001'    ;TXT
        RETLW  B'00001001'    ;TXT
        RETLW  B'00001001'    ;TXT
        RETLW  B'00001001'    ;TXT
        RETLW  B'01001001'    ;TXT
        RETLW  B'01111110'    ;TXT
        RETLW  B'01000000'    ;TXT
        RETLW  B'00000000'    ;TXT
        RETLW  B'00111110'    ;TXT
        RETLW  B'01000001'    ;TXT
        RETLW  B'01000001'    ;TXT
        RETLW  B'01000001'    ;TXT
        RETLW  B'01000001'    ;TXT
        RETLW  B'01000001'    ;TXT
        RETLW  B'00100010'    ;TXT
        RETURN
;       GOTO VARIAVEIS
```

Com isso obtém-se o seguinte efeito visual, que é mostrado na figura 7.2.

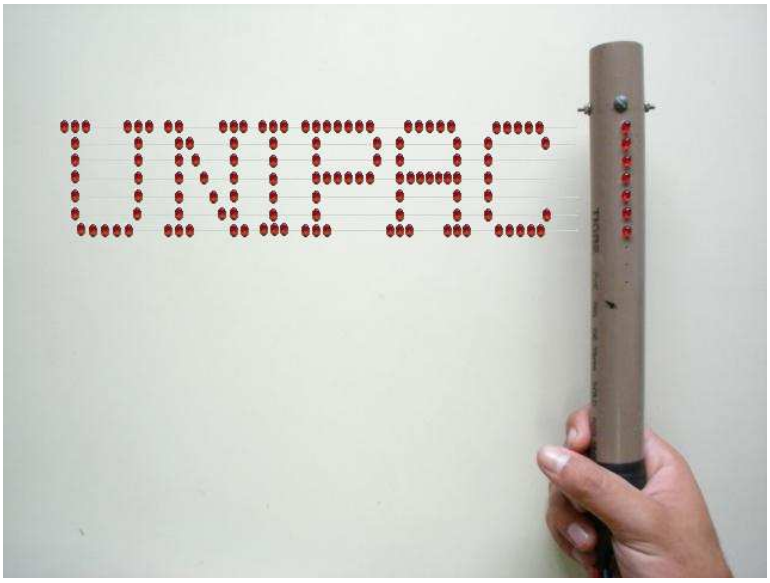


Figura 7.2 – Efeito Visual

Com este projeto, é possível perceber a incrível capacidade deste pequeno microcontrolador em habilitar e desabilitar suas saídas em um curto período de tempo, sem falar em sua capacidade de processamento, tendo como gerador de clock um oscilador externo de 4Mhz para uma maior precisão.

CAPÍTULO 8 – CONCLUSÃO

Sem dúvida alguma, o mundo moderno é “microcontrolado”. Estes pequenos circuitos estão presentes em quase tudo que manipulamos no nosso dia-a-dia (7). Eles estão presentes em aparelhos como:

- Televisores;
- DVD’s e Vídeo Cassetes;
- Fornos microondas;
- Em grande parte dos eletrodomésticos “inteligentes”;
- Alarmes e equipamentos segurança;
- Em periféricos para o PC (teclado, mouse, ...);
- No controle de injeção de veículos mais modernos e outros acessórios;
- Telefones e aparelhos celulares;
- Equipamentos para controle de processos industriais.

Como podemos ver, estamos cercados por “microcontroladores”. E o futuro indica que será cada vez maior o número o número de dispositivos microcontrolados (14).

Não se pode ir contra o futuro! Devemos ir de encontro a ele. Portanto, aprender é vital para ingressar no mercado de trabalho e significa manter-se nele (15).

BIBLIOGRAFIAS

(1) SOUZA, D.J. **Desbravando o PIC:** Baseado no microcontrolador PIC 16F84. 5. ed. São Paulo: Érica, 2000.

(2) SOUZA, D.J. **Desbravando o PIC:** Ampliado e Atualizado para PIC 16F628A. 6 ed. São Paulo: Érica, 2003.

(3) SOARES, M.J. Microcontroladores PIC. **Mecatrônica Fácil***,* v. 1 n. 6, p. 45-51, set. out. 2002.

(4) SOUZA, D.J; LAVINIA, N.C. **Conectando o PIC 16F877A:** Recursos Avançados. São Paulo: Érica, 2003.

(5) PEREIRA, F. **Microcontroladores PIC:** Programação em C. 2. ed. São Paulo: Érica, 2003.

(6) PEREIRA, F. **Microcontroladores PIC:** Técnicas Avançadas. 2. ed. São Paulo: Érica, 2002.

(7) SOARES, M.J. Microcontroladores e suas Ferramentas de Desenvolvimento. **Eletrônica Total***,* v. 1 n. 103, p. 60-64, nov. dez. 2004.

(8) SOARES, M.J. Microcontroladores Embedded. **Mecatônica Fácil***,* v. 1 n. 9, p. 22-25, mar. abr. 2003.

(9) SOARES, M.J. Microcontroladores, como iniciar? **Mecatrônica Fácil***,* v. 1 n. 15, p. 34-37, mar. abr. 2004.

(10) MICROCHIP. Products Microchip. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/30430c.pdf> Acesso em: 15 jul. 2004.

(11) SOARES, M.J. Como ampliar as portas de um microcontrolador. **Mecatrônica Fácil**, v. 1 n. 16, p. 20-24, maio jun. 2004.

(12) SOARES, M.J. Como Gravar Microcontrolador PIC. **Mecatrônica Fácil**, v. 1 n. 7, p. 43-49, nov. dez. 2002.

(13) SOARES, M.J. Gravador de PIC ProgPicII. **Mecatrônica Fácil**, v. 1 n. 13, p. 42-47, nov. dez. 2003.

(14) SOARES, M.J. Controle de motores de passo com o PIC. **Mecatrônica Fácil**, v. 1 n. 10, p. 21-25, maio jun. 2003.

(15) SOARES, M.J. Controle PWM com PIC. **Mecatrônica Fácil**, v. 1 n. 8, p. 36-40, jan. fev. 2003.

MECATRÔNICA FÁCIL. Saber. Disponível em: <http://www.mecatronicafacil.com.br/artigos/grav_pic_13/index.asp?posicao=2> Acesso em: 23 jul. 2004.