

Realização



{Apostila Didática PICMinas}

**Sistemas
Microcontrolados**



Autores:

**Fernando Esquírio Torres
Henrique Resende Martins**

Co-Autores:

**Bruno Silveira Avelar
James Hamilton Oliveira Junior
Mauricio Ferrari S. Correa**

www.picminas.com.br

Sumário

LISTA DE FIGURAS.....	4
LISTA DE TABELAS.....	7
Capítulo 1 – Introdução	8
1.1 SURGIMENTO DAS MÁQUINAS MODERNAS	8
1.2 SURGIMENTO DOS PROCESSADORES E DOS COMPUTADORES PESSOAIS.....	10
1.3 SISTEMAS MICROPROCESSADOS.....	13
1.3.1 UNIDADE CENTRAL DE PROCESSAMENTO (CPU)	14
1.3.2 MEMÓRIAS.....	15
1.3.3 REGISTRADORES	17
1.3.4 REGISTRADORES DE FUNÇÃO ESPECIAL (SFR)	18
1.3.5 PORTAS I/O (ENTRADA/SAÍDA)	18
1.3.6 BARRAMENTOS	19
1.3.7 INTERRUPÇÕES	20
1.4 MICROCONTROLADORES	21
1.5 LINGUAGEM DE MÁQUINA	23
1.6 REPRESENTAÇÃO NUMÉRICA	24
1.7 CONVERSÕES DE SISTEMAS NUMÉRICOS	26
1.7.1 CONVERSÃO BINÁRIA PARA DECIMAL	26
1.7.2 CONVERSÃO HEXADECIMAL PARA DECIMAL.....	26
1.7.3 CONVERSÃO HEXADECIMAL PARA BINÁRIA	27
1.8 O BIT	28
1.9 O BYTE	28
1.10 PORTAS LÓGICAS.....	29
1.10.1 PORTA AND.....	29
1.10.2 PORTA OR	30
1.10.3 PORTA NOT	30
1.10.4 PORTA XOR (OU-EXCLUSIVA).....	31
Capítulo 2 – Arquitetura do PIC e Kit de Desenvolvimento	32
2.1 PIC18F4550 e PIC32MX775F256L.....	34
2.1.1. PIC18F4550.....	34
2.2 KIT DE DESENVOLVIMENTO com PIC18F4550.....	49
2.2.1. CIRCUITOS ATUADORES.....	49
2.2.2. CHAVES/TECLAS.....	50
2.2.3. CIRCUITOS SENsoRES	51
2.2.4. DISPLAYS.....	53

2.2.5.	CIRCUITOS DE GRAVAÇÃO IN-CIRCUIT (ICSP)	55
2.2.6.	Jumpers do KIT PICMINAS	55
2.3	COMPONENTES DO KIT DIDÁTICO PIC18	57
Capítulo 3 – Ferramentas de Desenvolvimento		58
3.1	AMBIENTE DE DESENVOLVIMENTO – MPLAB	58
3.2	COMPILEDORES C18 E C32 DA MICROCHIP	63
3.2.1.	Instalação do Compilador C18.....	65
3.2.2.	Instalação do Compilador C32.....	69
3.3	FIRMWARE BOOTLOADER	71
3.4	COMO CRIAR UM PROJETO NO MPLAB	75
3.5	COMO COMPIRAR E GRAVAR UM FIRMWARE NO KIT DIDÁTICO UTILIZANDO BOOTLOADER	82
3.6	COMO GRAVAR UM FIRMWARE NO PIC UTILIZANDO UMA GRAVADORA (ICD2 MICROCHIP).....	84
3.7	COMO UTILIZAR A FERRAMENTA DE SIMULAÇÃO DO MPLAB	91
Capítulo 4 – Programação de Microcontroladores.....		94
4.1.	Vantagens e desvantagens de se programar em Assembly e em C:	94
4.2	PRINCÍPIOS DE PROGRAMAÇÃO.....	95
4.2.1.	ALGORITMOS ESTRUTURADOS.....	95
4.2.2.	FLUXOGRAMAS.....	96
4.2.3.	VARIÁVEIS E DADOS	97
4.3	CARACTERÍSTICAS DO COMPILADOR C18	98
4.3.1.	TIPOS DE DADOS E LIMITES	98
4.4.	CAMINHO DE PROCURA DE ARQUIVOS	100
4.5	LINGUAGEM DE PROGRAMAÇÃO C	100
4.5.1	PRIMEIROS PASSOS.....	101
4.5.2	VISÃO GERAL DE UM PROGRAMA	103
4.5.3	PALAVRAS RESERVADAS	103
4.5.4	IDENTIFICADORES	104
4.5.5	VARIÁVEIS E TIPOS DE DADOS	104
4.5.6	CONSTANTES.....	105
4.5.7	OPERADORES	106
4.5.8	COMANDOS E FUNÇÕES IMPORTANTES.....	110
4.6	BOAS PRÁTICAS DE PROGRAMAÇÃO	115

LISTA DE FIGURAS

Figura 1.1 - Figura do ábaco	8
Figura 1.2 - A Calculadora de Pascal e a Máquina Diferencial de Babbage	9
Figura 1.3 - Operação de um ENIAC	9
Figura 1.4 - Válvula dos Computadores Antigos	10
Figura 1.5 - Processador Pentium 4	10
Figura 1.6 - Computador PDP8	11
Figura 1.7 - Primeiro Processador da História (4004)	11
Figura 1.8 - Evolução Histórica do Microprocessadores	13
Figura 1.9 - Elementos Internos de um Microprocessador Básico	14
Figura 1.10 - CPU e suas unidades	14
Figura 1.11 - Foto com diversos tipos de memórias RAM	15
Figura 1.12 - Memória EPROM – Os dados são apagados com raios ultravioleta.....	16
Figura 1.13 - Tipos de dispositivos que utilizam memória FLASH para armazenamento de dados	16
Figura 1.14 - O Registrador é um tipo de memória que tem acesso direto a CPU.....	17
Figura 1.15 - Registradores com funções especiais	18
Figura 1.16 - Portas do microcontrolador	19
Figura 1.17 - Placa Mãe	19
Figura 1.18 - Atendimento de interrupção	20
Figura 1.19 - Desenho esquemático de um Microcontrolador	21
Figura 1.20 - Diagrama de blocos geral de um microcontrolador	22
Figura 1.21 - Representação de um número na base-10	24
Figura 1.22 - Transformação de um número na base-2 para base-10	25
Figura 1.23 - Comparação de um número na base-16 e na base-2	25
Figura 1.24 - Conversão de um número na base-2 para base-10	26
Figura 1.25 - Conversão de Hexadecimal para decimal	27
Figura 1.26 - Conversão de binário para hexadecimal	27
Figura 1.27 - Representação de um número na base-2	28
Figura 1.28 - Representação de nibbles	29
Figura 1.29 - Símbolo gráfico e Tabela verdade da porta AND	29
Figura 1.30 - Operação lógica AND	30
Figura 1.31 - Símbolo gráfico e Tabela verdade da porta OR	30
Figura 1.32 - Operação lógica OR	30
Figura 1.33 - Símbolo gráfico e Tabela verdade da porta NOT	30
Figura 1.34 - Operação lógica NOT	31
Figura 1.35 - Símbolo gráfico e Tabela verdade da porta XOR	31
Figura 1.36 - Operação lógica XOR	31
Figura 2.1 - Arquitetura Von-Neumann	33
Figura 2.2 - Arquitetura Harvard	33
Figura 2.3- Distribuição dos pinos no microcontrolador PIC18F4550 (Fonte: datasheet PIC18F4550).....	36
Figura 2.4 - Estrutura interna do microcontrolador PIC18F4550 (Fonte: datasheet PIC18F4550)	42
Figura 2.5 - Arquitetura do PIC18F4550 destacando as partes de um sistema microprocessado.....	43
Figura 2.6 - Processo Pipeline	44
Figura 2.7 - Detalhes da memória FLASH e da Pilha	45
Figura 2.8 - Mapa da memória de dados	47
Figura 2.9 - Registrador Status	48
Figura 2.10 - Ligação dos led's no PIC	49
Figura 2.11 - Ligação do Relé no PIC	49
Figura 2.12 - Ligação do Buzzer (Sirene) no PIC	50
Figura 2.13 - Ligação da Chave Push Button (Chaves Táteis) no PIC	50
Figura 2.14 - Ligação da Chave Dip switch no PIC.....	51
Figura 2.15 - Ligação do LDR no PIC	52
Figura 2.16 - Ligação do sensor de temperatura no PIC	52
Figura 2.17 - Ligação do potenciômetro no PIC	53
Figura 2.18 - Ligação do display duplo de 7 segmentos no PIC	53
Figura 2.19 - Ligação do display de LCD no PIC	54
Figura 2.20 – Conector padrão MICROCHIP ICSP	55

Figura 2. 21 - Jumpes do KIT PICMinas.	56
Figura 2.22 - Componentes principais do kit de desenvolvimento.	57
Figura 3. 1 - Ícone Instaladores do DVD Didático. Instalação do MPLAB.	59
Figura 3.2 - Tela de boas vindas do instalador do MPLAB.	59
Figura 3.3 - Termo de licença.	60
Figura 3.4 - Tipo de instalação. Escolha a opção "Completa".	60
Figura 3.5 - Escolha do diretório onde o MPLAB será instalado.	61
Figura 3.6 - Resumo do processo de instalação.	61
Figura 3.7 - Instalador do programa HI-TECH.	62
Figura 3.8 - Finalização da instalação.	62
Figura 3.9 - Tela meramente informativa.	63
Figura 3.10 - Função do compilador.	63
Figura 3.11 - Processo de compilação do código.	64
Figura 3.12 - Ícone Instaladores do DVD Didático - Instalador do Compilador C18.	65
Figura 3.13 - Tela de boas vindas do instalador do compilador C18.	66
Figura 3.14 - Licença do compilador C18.	66
Figura 3.15 - Escolha do diretório onde será instalado o C18.	67
Figura 3.16 - Final do processo de instalação do compilador C18.	67
Figura 3.17 - Ícone Instaladores do DVD Didático - Instalador do Compilador C32.	69
Figura 3.18 - Janela de boas vindas do instalador do compilador C32.	69
Figura 3.19 - Termo de licença do compilador C32.	70
Figura 3.20 - Escolha do diretório onde será instalado o C32.	70
Figura 3.21 - Final do processo de instalação do compilador C32.	71
Figura 3.22 - Método de Gravação Off-Board.	72
Figura 3.23 - Modo de gravação <i>in-circuit</i> via hardware.	73
Figura 3.24 - Método de Auto-Gravação.	74
Figura 3.25 - Arquivos modelo no DVD didático.	75
Figura 3.26 - Janela do MPLAB - Acessando o "Project Wizard".	76
Figura 3.27 - Janela de Boas Vindas.	76
Figura 3.28 - Escolha do microcontrolador.	77
Figura 3.29 - Escolha da "Active Toolsuite".	77
Figura 3.30 - Escolha do diretório.	78
Figura 3. 31 - Inserir arquivos.	78
Figura 3.32 - Resumo do projeto.	79
Figura 3.33 - Janela "Project".	79
Figura 3.34 - Adicionando arquivos ao projeto.	80
Figura 3. 35 - Adicionando arquivos ao projeto.	80
Figura 3.36 - Janela do MPLAB mostrando o arquivo main.c do projeto teste.mcp aberto.	81
Figura 3.37 - Botão de compilação - Build All.	82
Figura 3.38 - DVD Didático – Ícone “Gravar PIC”.	83
Figura 3.39 - Escolha do arquivo.hex que se deseja gravar no PIC.	83
Figura 3.40 - Processo de gravação do PIC.	84
Figura 3.41 - Módulo de gravação ICD2 da Microchip.	84
Figura 3.42 - Janela de reconhecimento do dispositivo.	85
Figura 3. 43 - Janela de instalação do driver.	85
Figura 3.44 - Janela de conclusão.	86
Figura 3. 45 - DVD Didático - Ícone Projetos - Projeto do Bootloader.	86
Figura 3.46 - Seleção do módulo de gravação.	87
Figura 3.47 - Seleção da opção Setup Wizard.	87
Figura 3.48 - Seleção da Porta de Comunicação.	88
Figura 3.49 - Seleção da alimentação do dispositivo.	88
Figura 3.50 - Janela de Conexão.	89
Figura 3.51 - Janela Download.	89
Figura 3.52 - Ícones de gravação habilitados.	90
Figura 3.53 - Mensagem informando que a programação foi realizada com sucesso.	90
Figura 3.54 - Selecioneando o simulador MPLAB.	91
Figura 3.55 - Botões do simulador MPLAB.	91
Figura 3.56 - Aba do MPLAB SIM na janela Output.	91
Figura 3.57 - Barra de Status indicando MPLAB SIM.	92
Figura 3.58 - Opção Watch.	92

Figura 3.59 - Janela Watch.	92
Figura 3.60 - Alterando para exibição em decimal.	93
Figura 4.1 - Ordenação Little-Endian.	99
Figura 4.2 - Conteúdo do arquivo main.c.	101
Figura 4.3 - Mensagem de compilação bem sucedida.	102
Figura 4.4 - Detalhamento da função main ().	102
Figura 4.5 - Exemplo de um programa C.	103
Figura 4.6 - Uso das estruturas if else.	110
Figura 4.7 - Uso da estrutura switch.	111
Figura 4.8 - Uso da estrutura while.	112
Figura 4.9 - Uso da estrutura for.	113
Figura 4.10 - Uso da estrutura for com decremento de índice.	113
Figura 4.11 - Uso da estrutura do-while.	114
Figura 4.12 - Exemplo do uso de espaços e comentários.	115
Figura 4.13 - Cabeçalho de arquivo.	116
Figura 4.14 - Uso de Defines.	117

LISTA DE TABELAS

Tabela 1 - Representação de números nas três bases.	27
Tabela 2 - Funcionalidades do PIC18F4550 (fonte datasheet PIC18F4550).	35
Tabela 3 - Pinos do PIC18F4550 (Fonte: datasheet do PIC18F4550).	41
Tabela 4 - Informações de ligação do LCD.	54
Tabela 5 - Descrição dos sub-diretórios do MCC18.	68
Tabela 6 - Descrição dos ícones de gravação.	90
Tabela 7 - Função de cada botão.	91
Tabela 8 - Vantagens e desvantagens entre linguagem Assempply e C.	94
Tabela 9 - Tipos de dados inteiros e seus limites.	98
Tabela 10 - Palavras reservadas em C.	104
Tabela 11 - Exemplos de variáveis válidas e inválidas.	104
Tabela 12 - Código de barra invertida.	106
Tabela 13 - Bases numéricas.	106
Tabela 14 - Operadores aritméticos.	107
Tabela 15 - Operadores relacionais.	107
Tabela 16 - Operadores lógicos booleanos.	107
Tabela 17 - Operadores lógicos bit a bit.	107
Tabela 18 - Operadores de memória.	108
Tabela 19 - Outros operadores.	108
Tabela 20 - Associação de operadores.	109
Tabela 21 - Precedência de operadores.	109

Capítulo 1 – Introdução

O objetivo principal deste capítulo é familiarizar o leitor com alguns conceitos e nomenclaturas existentes no mundo dos microprocessadores e microcontroladores. Além disso, o capítulo irá abordar um pouco da história da evolução tecnológica que culminou no mundo moderno que vivemos hoje.

1.1 SURGIMENTO DAS MÁQUINAS MODERNAS

Durante toda sua história, o homem buscou desenvolver meios para facilitar a realização de atividades complicadas e repetitivas. Atualmente, vivemos a era da automação total em que boa parte das atividades, sem cunho criativo, são ou serão realizadas por máquinas (robôs), desde complexas manipulações cirúrgicas até máquinas incumbidas da retirada do lixo. Por exemplo, hoje em dia é comum entrarmos em um banheiro de *Shopping Center* em que a luz acende sozinha, a torneira ativa automaticamente e até mesmo para acionar a descarga é preciso apenas que o usuário se afaste do sanitário. Na indústria moderna, a intervenção do homem nos processos vem sendo cada vez menor, restringindo-se apenas ao monitoramento de alarmes e eventuais falhas do sistema. Carros que falam ou que obedecem a comandos de voz; aviões que voam sozinhos; sondas espaciais não tripuladas que pousam em Marte, enfim são inúmeros os exemplos em que a automação das máquinas está presente nos dias atuais.

Para entendermos como chegamos a este momento, vamos voltar um pouco na história. Com o crescimento do conhecimento matemático com o passar dos anos, e seu grande uso nas atividades comerciais, um dos maiores problemas encontrados passou a ser a realização de cálculos mais complexos, o que levou à criação de dispositivos MECÂNICOS para auxiliar estas tarefas. Há aproximadamente 5.500 anos atrás, provavelmente na Mesopotâmia, surgiu o primeiro desses dispositivos, o Ábaco (Figura 1.1). Ele permitia fazer operações de adição, subtração, multiplicação e divisão. O mais intrigante é que esse dispositivo de construção tão simples criaria um conceito de máquina que muitos séculos depois seria a base para o surgimento de sistemas extremamente complexos conhecidos como **MICROPROCESSADORES**.



Figura 1.1 - Figura do ábaco.

Outros exemplos de máquinas utilizadas para a realização de cálculos matemáticos foram: **A Calculadora de Pascal** e **a Máquina Diferencial de Babbage** (Figura 1.2) que revolucionaram seu tempo e ajudaram o homem a dar passos mais largos em sua evolução.

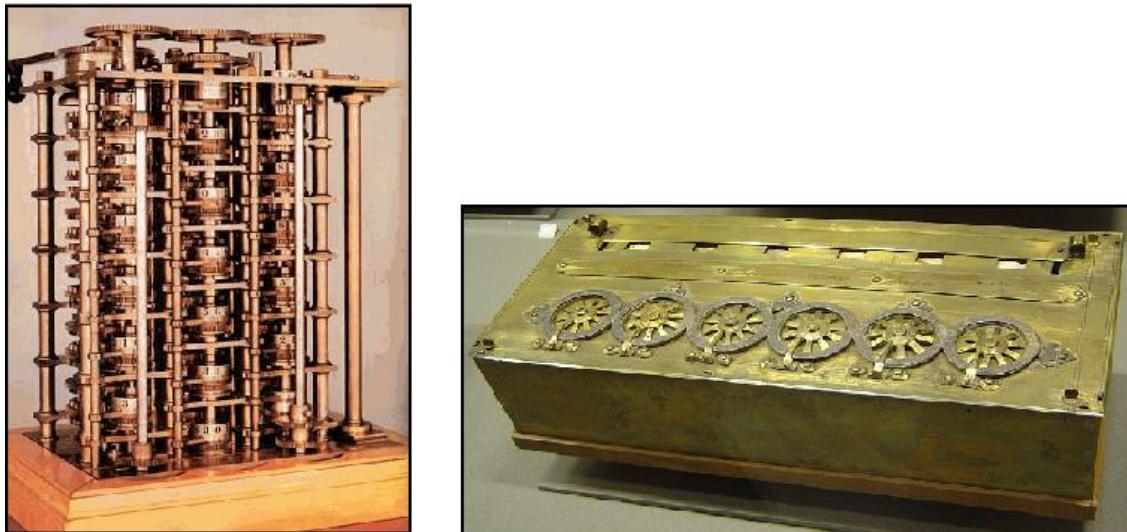


Figura 1.2 - A Calculadora de Pascal e a Máquina Diferencial de Babbage

Outro marco no desenvolvimento deste tipo de máquinas deu-se quando o homem passou a manipular a tensão e a corrente elétrica. Neste momento, os dispositivos que antes eram mecânicos passaram a ser elétricos, o que reduzia muito o seu tamanho, facilitando sua construção e seu aperfeiçoamento. Esta evolução tornou possível o surgimento dos primeiros computadores, sistemas esses inconcebíveis com dispositivos mecânicos. Mesmo assim o tamanho destes computadores eram gigantescos devido ao acúmulo de atividades e tarefas que eles possuíam se comparados aos dispositivos mecânicos. O ENIAC foi o primeiro computador digital eletrônico valvulado. Foi criado em 1946, e tinha as seguintes características: pesava 30 toneladas, media 5,5m de altura, 25m de comprimento e ocupava uma área de 180m² de área. Foi construído sobre estruturas metálicas e contava com 70 mil resistores e aproximadamente 18.000 válvulas a vácuo ocupando a área de um ginásio desportivo. Essa máquina não tinha sistema operacional e o funcionamento era parecido com uma calculadora simples de hoje. Era operado manualmente e efetuava os cálculos a partir de teclas que faziam interação direta com o Hardware. A Figura 1.3 trás um desenho do ENIAC sendo operado.

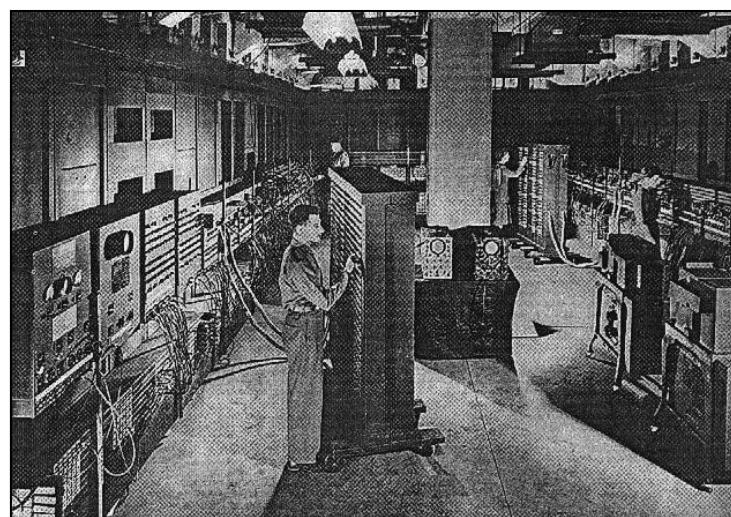
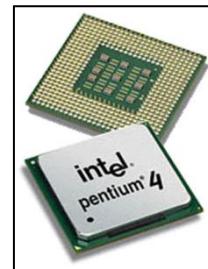


Figura 1.3 - Operação de um ENIAC.

Com o surgimento da microeletrônica foram desenvolvidos os TRANSISTORES que substituíram as válvulas dos computadores antigos, que possuíam até 10 cm de comprimento cada uma (ver Figura 1.4), ao passo que os transistores atuais, presentes como principal elemento na concepção dos microprocessadores, ocupam poucas unidades de micrometros quadrados (μm^2 - a milionésima parte de um metro quadrado), o que possibilitou inserir dentro de um CHIP de processador (Figura 1.5) funcionalidades que não caberia em centenas de prédios em um computador valvulado.

**Figura 1.5 - Processador Pentium 4.****Figura 1.4 - Válvula dos Computadores Antigos.**

O mundo tecnológico em que vivemos hoje, dominado por dispositivos e sistemas microprocessados e microcontrolados, teve origem no desenvolvimento da tecnologia de circuitos integrados (Microeletrônica Digital). Esta evolução possibilitou incorporar centenas de milhares de transistores (elemento básico de qualquer sistema digital e que substituíram as antigas válvulas elétricas dos primeiros computadores) em um único CHIP com dimensões de alguns centímetros quadrados (cm^2). Essa foi a condição prévia para a fabricação dos **MICROprocessadores** (constituídos basicamente de transistores com dimensões micrométricas), que possibilitou o surgimento dos primeiros computadores pessoais (*PC – Personal Computer*), onde além do microprocessador, foram adicionados periféricos externos, tais como, memória, linhas de entrada/saída e temporizadores. Com a contínua evolução da tecnologia dos circuitos integrados, deu-se origem a um sistema que em um único CHIP embarcava tanto os elementos básicos necessários para o funcionamento de um computador (microprocessador, memória e linhas de entrada/saída), quanto dispositivos periféricos como, por exemplo, conversores A/D e D/A¹, interfaces de comunicação paralela e serial, sistemas de interrupção entre outros, dando origem a uma nova era de sistemas microprocessados. Esse novo CHIP, que inicialmente era conhecido como **MICROcomputador**, originou os **MICROcontroladores** da atualidade. Mas antes de nos aprofundarmos neste assunto, vamos entender, por meio de fatos históricos, como surgiram os microprocessadores e os computadores modernos.

1.2 SURGIMENTO DOS PROCESSADORES E DOS COMPUTADORES PESSOAIS

Em 1969 uma equipe de engenheiros japoneses da empresa BUSICOM chegou aos EUA com um pedido de circuitos integrados para calculadoras que deveriam ser desenvolvidos de acordo com as necessidades de seus produtos. O pedido foi enviado para a Intel que encarregou Marcian Hoff para execução deste projeto. Tendo Marcian já trabalhado com o desenvolvimento de um computador (computador à válvula), o PDP8 (ver Figura 1.6), ele sugeriu uma ideia fundamentalmente diferente do pedido da BUSICOM. A sua ideia presumia que o funcionamento do circuito integrado seria baseado em um programa armazenado nele ao invés dos projetos engessados da época, em que cada circuito integrado possuía uma e somente uma funcionalidade. Isso significaria que a configuração dos produtos da empresa japonesa ficaria mais simples, tendo em vista que um único circuito integrado poderia ser utilizado em mais de um produto, mudando-se apenas a programação gravada nos mesmos. Por outro lado, esta nova filosofia de projeto iria requerer mais memória do sistema que o projeto proposto pelos engenheiros japoneses, o que na época era algo ainda de valor elevado.

¹ **Conversor A/D** – Circuito Eletrônico utilizado para converter sinais de tensão ou de corrente elétrica (sinais analógicos) em números binários (sinais digitais). **Conversor D/A** – Converte números binários em sinais de tensão ou corrente elétrica.

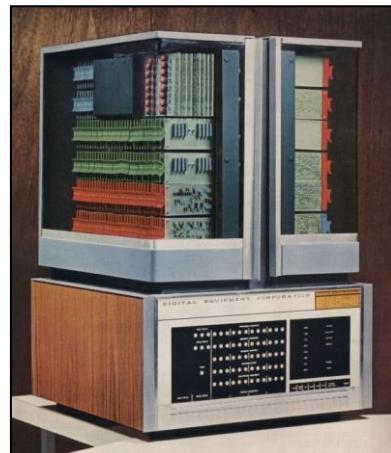


Figura 1.6 - Computador PDP8.

Depois de um tempo, embora os engenheiros japoneses tivessem tentando encontrar uma solução mais fácil, a ideia de Marcian ganhou força e o primeiro microprocessador nasceu. Esse microprocessador foi batizado de 4004 (ver Figura 1.7) e era capaz de tratar apenas 4-bits em uma velocidade de 6.000 operações por segundo, ou 6kHz (Os processadores atuais possuem 64-bits e trabalham com até 3,33 Bilhões de operações por segundo – 3.33GHz, ou seja, quase um milhão de vezes mais rápido).

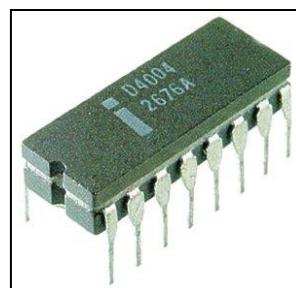


Figura 1.7 - Primeiro Processador da História (4004).

Algum tempo depois, uma empresa chamada CTC solicitou à Intel e à Texas Instruments que fabricassem um microprocessador de 8-bits para ser utilizado em seus terminais de computador. Devido aos altos custos e sem saber o valor do que estava desenvolvendo, a CTC desistiu do projeto. No entanto, a Intel e a Texas resolveram continuar custeando o projeto e, em abril de 1971, o primeiro processador de 8-bits, chamado de 8008, surgiu no mercado. Ele possuía uma memória de incríveis 16k-bits (que para época era algo inovador), um conjunto de 45 instruções diferentes para sua programação (eram operações lógicas como AND, OR e NOT, e aritméticas como ADD e SUB) e uma velocidade de até 300.000 instruções por segundo (300 kHz). Este processador foi a base de todos os microprocessadores modernos de hoje e deu origem, em abril de 1971, ao microprocessador 8080 que também era de 8-bits, mas possuía 64kbits de memória, 75 instruções para programação e um preço inicial de US\$ 360,00 (Trezentos e sessenta dólares), o que possibilitou o surgimento de computadores pessoais com valores acessíveis. A consequência disso foi a inesperada popularização da ideia e crescimento exponencial das vendas. Para ter ideia a prospecção mais otimista da Intel era da venda de 3.000 microprocessadores no primeiro ano de lançamento, mas as vendas superaram as dezenas de milhares, baixando ainda mais o valor desses processadores e, consequentemente, dos computadores.

Outra empresa americana, a Motorola, notou rapidamente o que estava acontecendo e lançou no mercado um novo microprocessador de 8-bits, o 6800. O construtor chefe foi Chuck Peddle e além do microprocessador propriamente dito, a Motorola foi a primeira companhia a fabricar outros periféricos como os 6820 e 6850, com outras funcionalidades que os microprocessadores não realizavam. Muitas outras companhias perceberam a grande importância dos microprocessadores e começaram a produzir seus

próprios projetos. Chuck Peddle deixou a Motorola para entrar para a MOS Technology e continuar trabalhando intensivamente no desenvolvimento dos microprocessadores.

Em 1975, na exposição WESCON nos Estados Unidos, aconteceu um episódio na história dos microprocessadores. A MOS Technology anunciou que iria lançar no mercado os microprocessadores 6501 e 6502 ao preço de US\$ 25 cada e que poderia atender de imediato todas as encomendas. Isto pareceu tão sensacional que muitos pensaram tratar-se de uma espécie de fraude, considerando que os competidores vendiam os processadores 8080 e 6800 por aproximadamente US\$ 179,00 cada. Para responder a este competidor, tanto a Intel quanto a Motorola baixaram seus preços para US\$ 69,95 logo no primeiro dia da exposição. Rapidamente a Motorola, antiga empresa onde Peddle trabalhava, colocou uma ação em tribunal contra a MOS Technology e contra o próprio Chuck Peddle por violação dos direitos autorais por copiarem o processador 6800. Com isso, a MOS Technology deixou de fabricar o 6501, mas continuou com o 6502. O 6502 era um microprocessador de 8-bits com 56 instruções e uma capacidade de endereçamento de 64kBytes de memória. Devido ao seu baixo custo, os processadores 6502 tornaram-se muito popular, sendo utilizado em computadores como o KIM-1, Apple I, Apple II, Atari, Comodore, Acorn, Oric, Galeb, Orao, Ultra e muitos outros. Logo apareceram várias outras empresas interessadas em fabricar o 6502 (Rockwell, Sznertek, GTE, NCR, Ricoh e Comodore) que, no auge da sua prosperidade, chegou a vender microprocessadores à razão de 15 milhões de processadores por ano.

Contudo, os outros não baixaram os braços. Frederico Faggin deixou a Intel e fundou a Zilog Inc. Em 1976, a Zilog anunciou o Z80. Durante a concepção deste microprocessador, Faggin tomou uma decisão crítica. Sabendo que existia uma enorme quantidade de programas já desenvolvidos para o microprocessador 8080, Faggin concluiu que muitos permaneceriam fiéis a este microprocessador devido às grandes despesas que representaria a troca do processador. Assim, ele decidiu que o novo microprocessador deveria ser **COMPATÍVEL** (nasceu a ideia de compatibilidade) com o 8080, ou seja, deveria ser capaz de executar todos os programas já escritos para o 8080. Além disto, outras características adicionais foram incorporadas ao CHIP, de tal modo que o Z80 se tornou um microprocessador muito potente em seu tempo. Ele podia endereçar diretamente 64kBytes de memória, tinha 176 instruções para programação, um grande número de registros, uma opção para refresh de memória RAM dinâmica, maior velocidade de funcionamento, etc. O Z80 tornou-se um grande sucesso e todos que utilizavam o processador 8080 migraram para o Z80. Pode-se dizer que o Z80 se constituiu como o microprocessador de 8-bits de maior sucesso em seu tempo. Além da Zilog surgiram outros novos fabricantes como, por exemplo, Mostek, NEC, SHARP e SGS. O Z80 foi o coração de muitos computadores como o Spectrum, Partner, TRS703, Z-3 e Galaxy, que fizeram muito sucesso devido sua grande utilização.

Em 1976, a Intel lançou uma versão melhorada do microprocessador de 8-bits, chamado de 8085. Contudo, a superioridade do Z80 fez com que a Intel desistisse do projeto. Apesar do surgimento, na época, de outros microprocessadores no mercado (6809, 2650, SC/MP etc.), o cenário da época estava bem definido. Já não existiam grandes melhorias a serem introduzidas pelos fabricantes que justificassem o lançamento de um novo microprocessador. Desta forma, o 6502 e o Z80, acompanhados pelo 6800, mantiveram-se como os mais representativos microprocessadores de 8-bits do período. Restou à Intel trabalhar no lançamento do primeiro microprocessador de 16-bits – o 8088 – que fez muito sucesso e deu origem a todos os processadores atuais. Ele foi seguido pelo 80186, depois pelo 80286, que evoluiu para o 80386 (popularmente conhecido como 386), chegando ao 80486 (popularmente conhecido como 486). Depois surgiu o Pentium que evoluiu até o Pentium 4. Nesse momento, a tecnologia se deparou com uma barreira na evolução dos microprocessadores que até então era baseada, quase que exclusivamente, no aumento da velocidade de processamento. Em seguida, a evolução dos processadores atrelou-se à evolução da sua **arquitetura interna** (forma como se estruturam e interagem os componentes básicos utilizados em sua construção). Atualmente, os processadores que encontramos no mercado são processados Dual Core, Core 2 Duo, Quad Core, Xeon, Turion, etc., que diferem entre si pelo número e tipo de Núcleos de Processamento (Core) existentes em sua arquitetura interna.

A Figura 1.8 mostra a evolução histórica dos processadores, evidenciando alguns dos principais sistemas desenvolvidos até os dias atuais.

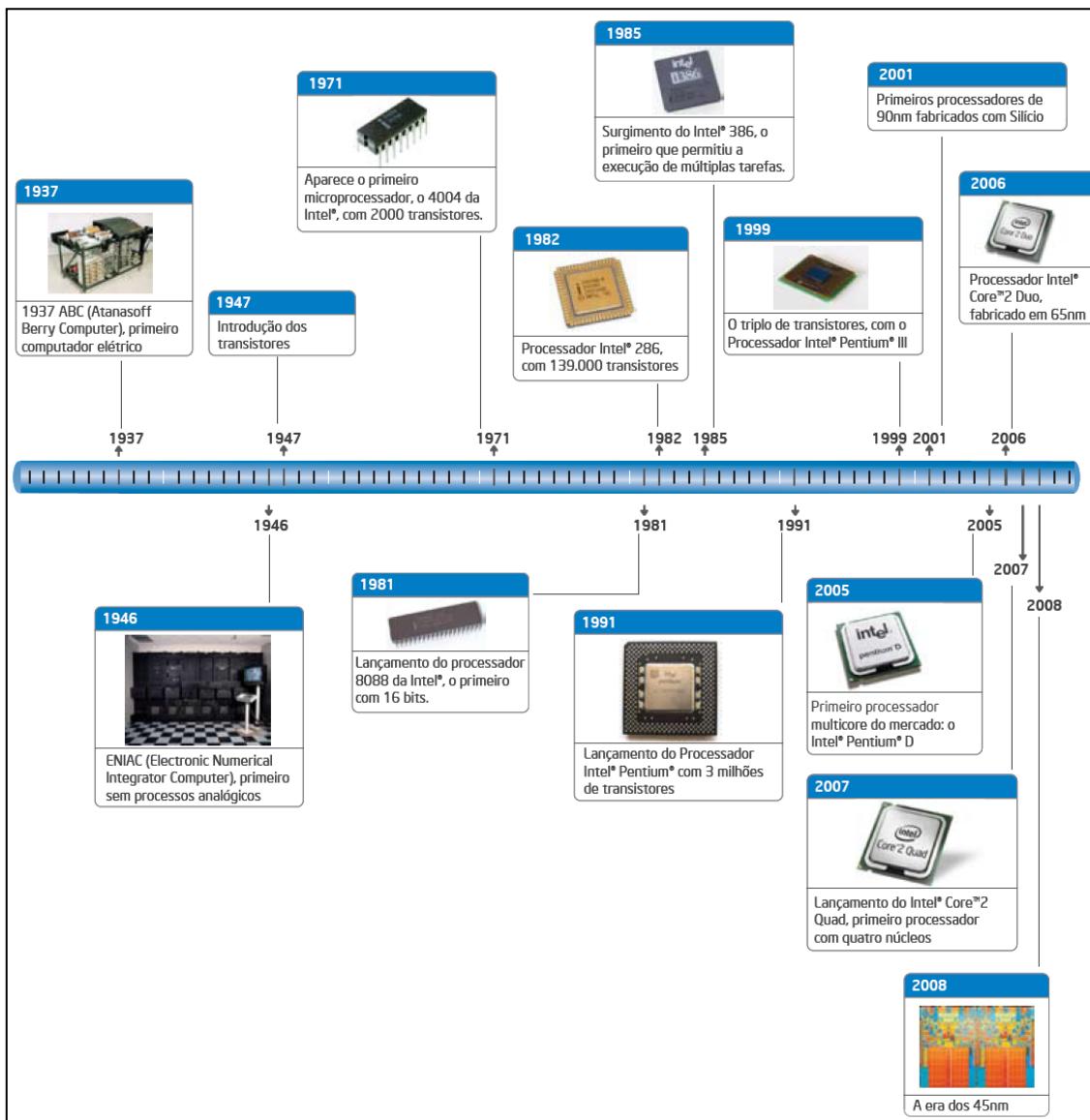


Figura 1.8 - Evolução Histórica dos Microprocessadores.

1.3 SISTEMAS MICROPROCESSADOS

Como visto, os microprocessadores são máquinas elétricas onde podemos armazenar instruções lógicas, aritméticas e de tomada de decisão permitindo que o mesmo funcione de várias formas diferentes dependendo das instruções (programa) que foram armazenadas em sua memória. Em sistemas microprocessados, o microprocessador é geralmente conhecido como CPU (*Central Processing Unit – Unidade Central de Processamento*) e funciona como o cérebro de todo o sistema, ou seja, é ele quem comanda tudo que é feito pelo sistema microprocessado. Um exemplo muito conhecido de sistemas microprocessados é o computador pessoal (Laptop ou Desktop) que utilizamos todos os dias. Outros exemplos são: Celulares, Ipods, Palm Tops, etc.

Por mais diferentes que possam parecer os diversos exemplos de sistemas microprocessados, todos eles são compostos basicamente por uma **CPU**, um conjunto de **Memórias**, Portas de **Entrada/Saída** (onde são conectados os periféricos) e **Barramentos** (Figura 1.9).

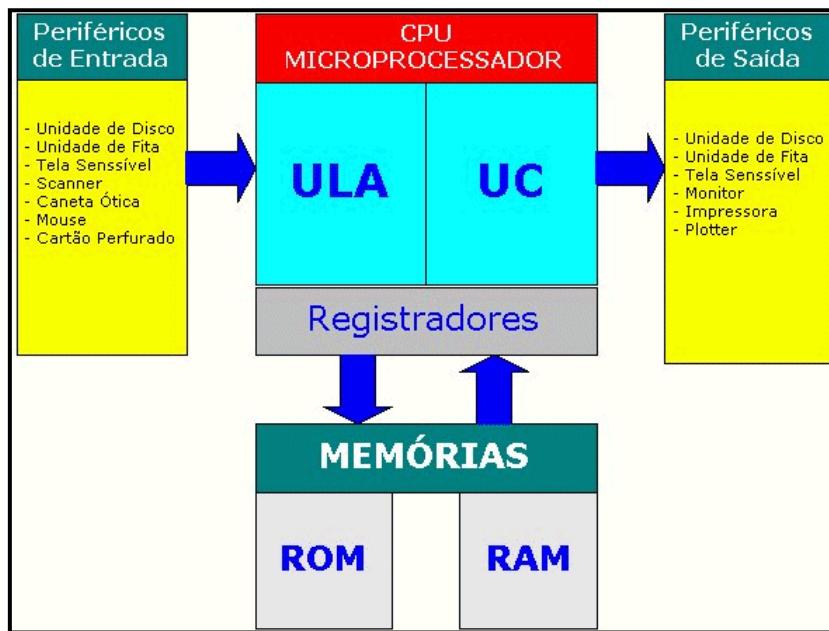


Figura 1.9 - Elementos Internos de um Microprocessador Básico.

1.3.1 UNIDADE CENTRAL DE PROCESSAMENTO (CPU)

A **CPU** (*Central Processing Unit* em inglês, ou **Unidade Central de Processamento**), também conhecida como **processador** ou ainda **microprocessador**, que processa as instruções contidas no *programa*. Na maioria das CPUs, essa tarefa é dividida entre a **unidade de controle (UC)** que dirige o fluxo do programa e uma ou mais **unidades lógicas e aritméticas (ULA)** que executam operações de acordo com as instruções recebidas pelo programa. (Figura 1.9)

A função da CPU é reconhecer um conjunto básico de instruções utilizadas para escrever programas que comandam o seu funcionamento, ou seja, que controlam toda a operação e funcionamento do computador ou de um sistema microprocessado. Ela é composta de várias unidades menores, das quais as mais importantes são: (Figura 1.10)

- **Decodificador de Instrução:** É o circuito que reconhece instruções de programa, e executa outros circuitos com base nessas instruções. O tamanho do Conjunto de Instruções (*Instructions Set*) refere-se a quantidade de instruções que esse circuito é capaz de realizar e é diferente para cada família de microcontrolador.
- **Unidade Lógica e Aritmética:** Realiza todas as operações matemáticas (aritméticas) e lógicas sobre os dados do programa.
- **Registradores:** Um registrador ou célula de memória é um circuito eletrônico capaz de armazenar uma pequena quantidade de Bytes (geralmente 1 Byte), porém muito rápido. O conjunto de registradores pode ou não estar contido na CPU e tem como função o armazenamento temporário dos dados e as instruções que estão em processamento.



Figura 1.10 - CPU e suas unidades.

1.3.2 MEMÓRIAS

Em sistemas microprocessados, memória são todos os dispositivos que permitem que os dados sejam guardados, temporariamente ou permanentemente. As memórias podem ser classificadas basicamente de duas maneiras. A primeira classificação está associada à forma como os dados são armazenados, neste caso, existem dois tipos de memória, as **Voláteis** e as **Não-Voláteis**.

- **Memórias Voláteis:** são as que requerem energia para manter a informação armazenada. São fabricadas com base em duas tecnologias: *dinâmica* e *estática*. A dinâmica é a mais de menor custo entre a esse tipo de memória. São popularmente conhecidas como Memória RAM (do inglês *Randomic Access Memory* - Memória de Acesso Aleatório), que significa que os dados, nela armazenados, podem ser acessados a partir de qualquer endereço. As memórias RAM se contrapõem com as de acesso sequencial, que exigem que qualquer leitura/escrita seja feita a partir do primeiro endereço e, sequencialmente, vai “pulando” de um em um até atingir o endereço desejado. Na realidade, existem outras memórias de acesso aleatório nos computadores, inclusive não-voláteis, portanto, é importante ter o conhecimento de que o nome RAM é apenas uma popularização do nome da memória principal dos computadores, utilizada para armazenar os programas e dados no momento da execução. (Figura 1.11)

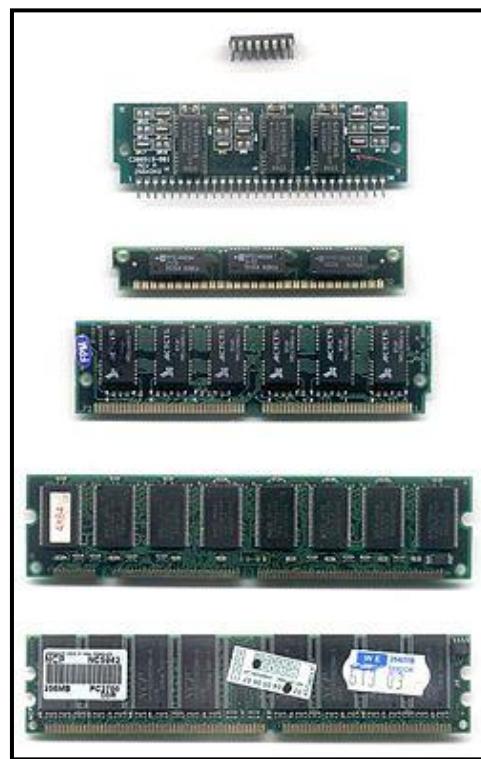


Figura 1.11 - Foto com diversos tipos de memórias RAM.

O nome *dinâmica* é referente à tecnologia utilizada para armazenar programas e dados e não à forma de acessá-los. De modo simplificado, ela funciona como uma bateria que deve ser recarregada sempre que apresentar carga insuficiente para alimentar o equipamento.

Todas as vezes que a CPU (unidade de processamento central) acessar a memória, para escrita ou para leitura, cada célula dessa memória é atualizada. Se ela tem “1” (um) lógico armazenado, sua “bateria” será recarregada; se ela tem “0” (zero) lógico, a “bateria” será descarregada. Este procedimento é chamado de refresco de memória, em inglês, *refresh*.

A memória estática não necessita ser analisada ou recarregada a cada momento. Fabricada com circuitos eletrônicos conhecidos como *latch*, guardam a informação por todo o tempo em que estiver a receber alimentação.

- Memórias Não-Voláteis:** São aquelas que guardam todas as informações mesmo quando não estiverem a receber alimentação. Como exemplos, citam-se as memórias conhecidas por ROM e FLASH, bem como os dispositivos de armazenamento em massa, disco rígido, CDs e disquetes. As memórias somente para leitura, do tipo ROM (sigla de Read Only Memory), permitem o acesso aleatório e são conhecidas pelo fato de o usuário não poder alterar o seu conteúdo. Para gravar uma memória deste tipo são necessários equipamentos específicos. Dentre as memórias do tipo ROM destacam-se as seguintes:

SIGLA	NOME	TECNOLOGIA
ROM	<i>Read Only Memory</i> (memória somente de leitura).	Gravada na fábrica uma única vez.
PROM	<i>Programable Read Only Memory</i> (memória programável somente de leitura).	Gravada pelo usuário uma única vez.
EPROM	<i>Erasable Programmable Read Only Memory</i> (memória programável e apagável somente de leitura).	Pode ser gravada ou regravada por meio de um equipamento que fornece as tensões elétricas adequadas em cada pino. Para apagar os dados nela contidos, basta iluminar o <i>chip</i> com raios ultravioleta. Isto pode ser feito através de uma pequena janela presente na parte de cima do CHIP (Figura 1.12).
EEPROM	<i>Electrically Erasable Programmable Read Only Memory</i> (memória programável e apagável eletronicamente somente de leitura).	Pode ser gravada, apagada ou regravada utilizando um equipamento que fornece as voltagens adequadas em cada pino.



Figura 1.12 - Memória EPROM – Os dados são apagados com raios ultravioleta.

O tipo de memória conhecido como FLASH é o tipo mais moderno dentre os apresentados aqui, mas é uma variação do tipo EEPROM. Tornaram-se muito populares por dois motivos: a utilização de dispositivos de armazenamento removíveis como os chamados *Pen Drives*, a aplicação em equipamentos de som que reproduzem música no formato MP3 e os cartões de memória das câmeras digitais (Figura 1.13). Os dados armazenados neste tipo de memória permanecem ali sem a necessidade de alimentação. Sua gravação é feita em geral através da porta USB.



Figura 1.13 - Tipos de dispositivos que utilizam memória FLASH para armazenamento de dados.

A segunda classificação dada às memórias está relacionada ao seu modo de acesso, onde temos dois tipos de memória, as **Memórias Primárias** e as **Memórias Secundárias**.

- **Memórias Primárias:** também chamadas de memória real, são memórias que o processador pode acessar diretamente, sem as quais os sistemas computadorizados não podem funcionar. Geralmente fornecem uma “ponte” para as secundárias, mas a sua função principal é a de conter a informação necessária para o processador num determinado momento; esta informação pode ser, por exemplo, os programas em execução. Nesta categoria insere-se a memória **RAM** (volátil), memória **ROM** (não-volátil), **registradores** e **memórias cache**. Apesar desse tipo de memória ser bem mais rápida que a memória secundária, elas possuem baixa capacidade de armazenamento, onde é armazenado somente o programa que está sendo executado pelo processador.
- **Memórias Secundárias:** memórias que não podem ser acessadas diretamente, a informação precisa ser carregada em memória primária antes de poder ser tratada pelo processador. Não são estritamente necessárias para a operação dos sistemas computadorizados. São geralmente **não-voláteis**, permitindo guardar os dados permanentemente. Incluem-se, nesta categoria, os discos rígidos (HDs), CDs, DVDs e disquetes. São memórias de armazenamento em massa, guardam grandes quantidades de dados.

Para entendermos melhor a aplicação dessas memórias vamos pensar em um exemplo de computador, sendo a RAM a memória primária e o HD a memória secundária. Quando clicamos em um ícone para abrir um programa (por exemplo, o WORD), este programa, que estava armazenado na memória secundária (HD), é carregado na memória primária (RAM) e ficará lá até que o programa seja encerrado.

1.3.3 REGISTRADORES

Como já foi dito, um registrador ou célula de memória é um circuito eletrônico capaz de armazenar uma pequena quantidade de Bytes (geralmente 1 Byte). Os registradores estão no topo da hierarquia de memória, sendo assim, são o meio mais rápido e caro de se armazenar um dado. É uma espécie de secretaria utilizada para armazenar todos os dados sobre os quais serão realizados algum tipo de operação (ADD, MOVE, SHIFT, etc.). Ele também armazena os resultados dessas operações. Por exemplo, quando uma instrução de soma entre duas variáveis quaisquer (A e B) é executada, antes da soma ser realizada, as variáveis serão carregadas, da memória RAM, para dentro de dois registradores. Só depois a soma é realizada. Isso acontece porque somente os registradores possuem comunicação direta com a CPU de um processador (Figura 1.14). O resultado da operação também será armazenado em outro registrador antes de ser carregado na memória RAM. No PIC18F4550 esses registradores são chamados de **GPR** (*General Purpose Register – Registradores de Propósito Geral*).

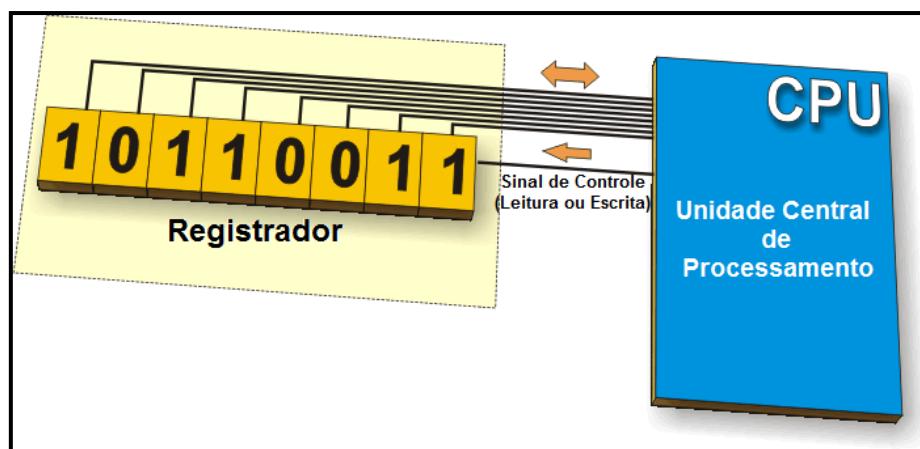


Figura 1.14 - O Registrador é um tipo de memória que tem acesso direto a CPU.

1.3.4 REGISTRADORES DE FUNÇÃO ESPECIAL (SFR)

Além dos registradores que não possuem nenhuma função especial ou pré-determinada, todo microprocessador ou microcontrolador possui um conjunto de registradores cujas funções são pré-determinadas pelo fabricante. Seus bits são conectados (literalmente) a circuitos internos, tais como: temporizadores (timers), conversores A/D e osciladores (Figura 1.15). Isso significa que eles estão diretamente no comando de operações de hardware, sem precisar da intervenção da CPU, ou seja, para modificar alguma característica de hardware que está associada a um SFR, basta modificar os bits desse registrador. No caso dos registradores de 8-bits, cada SFR funcionará como 8 chaves (liga/desliga) que comandam alguns pequenos circuitos dentro do microcontrolador. Um exemplo disso seriam os circuitos que determinam (controlam) se alguns pinos de um microcontrolador funcionaram como pinos de entrada de sinal ou pinos de saída de sinal. No PIC18F4550 este circuito está associado a um SFR chamado **TRIS**. Quando um dos bits do TRIS for ajustado para lógica “1” (esse ajuste é feito pelo projetista por meio do programa), a pino do PIC que estiver associado àquele bit funcionará como Entrada (Input). Já quando o bit for ajustado para a lógica “0”, o pino passará a funcionar como Saída (Output). Ou seja, o registrador TRIS interfere diretamente no funcionamento do hardware do microcontrolador.

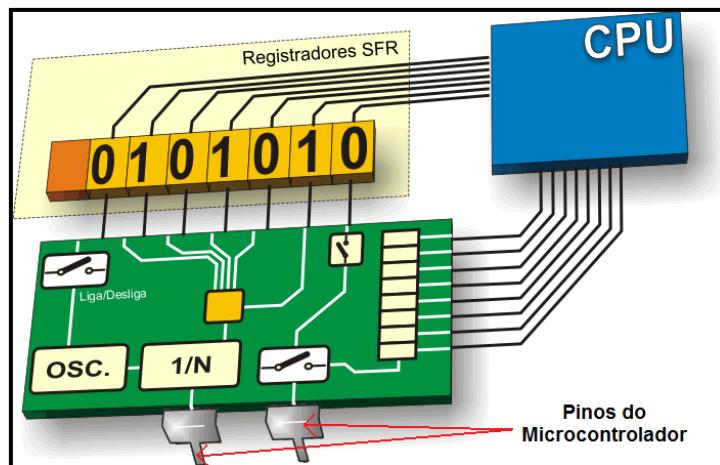


Figura 1.15 - Registradores com funções especiais.

Outros exemplos de registradores de função especial (SFR), são:

- **STATUS**: utilizado para armazenamento de flags matemáticos e de estado da CPU, além dos bits de seleção do banco de memória RAM.
- **INTCON**: utilizado para controle de interrupções.
- **OPTION_REG**: utilizado para configurar o funcionamento de alguns periféricos internos ao PIC.
- **PORT**: utilizado para a leitura ou escrita nos pinos do PIC (Secção 1.3.5).
- **TRIS**: utilizado para configurar os pinos das portas como entrada ou saída (input/output).

Nos PICs, tanto os SFR quanto os GPR são muitas vezes referenciados pela letra “f”. Isso decorre da nomenclatura utilizada pela Microchip: file registers.

É importante também sabermos que os registradores “f” (SFR e GPR) são mapeados dentro da memória RAM dos PICs. Desta forma, o acesso aos registradores é feito pelo seu endereço de localização na memória.

1.3.5 PORTAS I/O (ENTRADA/SAÍDA)

É por meio destas portas que o processador interage com o meio exterior. É onde estão conectados os **Periféricos** dos sistemas microprocessados (figura 1.9). São exemplos de periféricos: impressoras, mouses, sensores, teclados, displays de LCD, etc.

Os microcontroladores possuem um ou mais registradores de funções especiais chamados **PORT** conectados aos seus pinos I/O (pinos de Entrada ou Saída). Esses pinos são chamados de I/O porquê, como vimos anteriormente, eles podem ser configurados tanto como entradas quanto como saídas,

dependendo da necessidade da aplicação. Quando os pinos estão configurados como entrada, o registrador PORT passa a funcionar como um registrador de leitura, onde é possível ler o estado (“0” ou “1”) de cada um dos pinos que estão conectados àquele registrador. Já quando os pinos estão configurados como saída, o registrador PORT funciona como um registrador de escrita, onde é possível colocar lógica “0” ou lógica “1” nos pinos (Figura 1.16 - Portas do microcontrolador.).

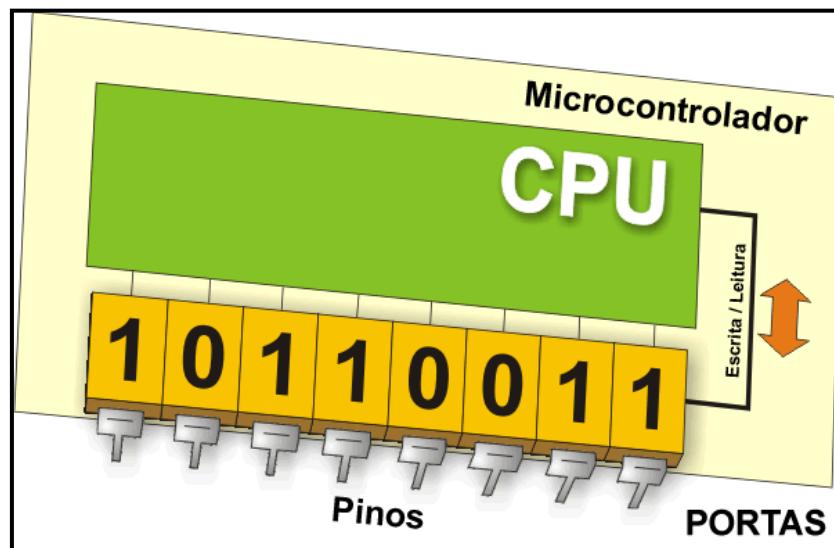


Figura 1.16 - Portas do microcontrolador.

Uma das especificações mais importantes dos pinos I/O é a corrente máxima com que eles podem operar. Para a maioria dos microcontroladores, a corrente fornecida por um pino é suficiente para acender um LED² ou outro dispositivo similar de baixa corrente (10/20mA). No entanto, se um microcontrolador possuir muitos pinos I/O, e todos eles estiverem ativos, a corrente máxima em cada pino será menor. Isso acontece porque além de cada pino especificar a corrente máxima com a qual ele pode trabalhar, o microcontrolador de modo geral também tem um limite de corrente de operação. Desta forma, se vários pinos forem acionados simultaneamente, esse limite não poderá ser ultrapassado, fazendo com que o limite de cada pino diminua. É importantíssimo verificar no **datasheet** do microcontrolador, que está sendo utilizado, qual é o valor máximo de corrente por pino I/O (operando isoladamente) e por conjunto de pinos.

1.3.6 BARRAMENTOS

Em sistemas microprocessados, o **barramento** é um conjunto de linhas de comunicação que permitem a interligação entre dispositivos, como a CPU, a memória e outros periféricos. Em um computador, esses barramentos são as trilhas do circuito impresso da **Placa Mãe – Mother Board** – (Figura 1.17).

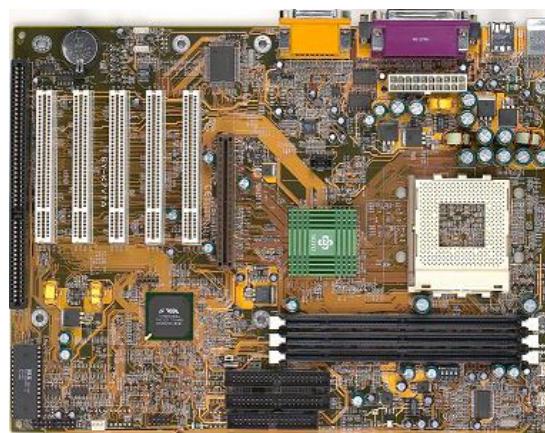


Figura 1.17 - Placa Mãe.

² LED – Light Emitting Diode, o que significa Diodo Emissor de Luz. É um componente eletrônico muito utilizado como luz de sinalização que acende com um valor de corrente elétrica bem baixa.

Os barramentos podem ser divididos em três conjuntos:

- **Barramento de dados:** onde trafegam os dados dos programas;
- **Barramento de endereços:** onde trafegam os endereços;
- **Barramento de controle:** sinais de controle que sincronizam as duas anteriores.

O desempenho do barramento é medido pela sua largura de banda (quantidade de bits que podem ser transmitidos ao mesmo tempo), geralmente potências de 2 (8-bits, 16-bits, 32-bits, 64-bits, etc.). E também pela velocidade da transmissão medida em bps (bits por segundo), por exemplo, 10 bps, 160 Kbps, 100 Mbps, 1 Gbps etc.

1.3.7 INTERRUPÇÕES

Um conceito importante a ser abordado quando se fala de sistemas microprocessados são as interrupções. São sinais emitidos por um ou mais dispositivos periféricos que tipicamente resultam em uma troca de contexto, isso é, o processador pára de atender a tarefa que estava tratando e atende ao dispositivo que fez o “pedido de interrupção” (IRQ – Interrupt Request) (SILBERSCHATZ, 2004).

Alguns processadores oferecem uma maneira de iniciar rotinas de software em resposta a eventos eletrônicos assíncronos. Esses eventos são sinalizados para o processador através de IRQs. Segundo SILBERSCHATZ, 2004, o processamento da interrupção é uma troca de contexto entre a rotina corrente e uma rotina dedicada a tratar a interrupção. Essa última é chamada “rotina de serviço de interrupção”, ou “tratador de interrupção” (interrupt handler). Os endereços dessas rotinas são chamados “vetores de interrupção” e são armazenados, geralmente, em uma tabela na memória RAM, permitindo sua modificação caso seja necessário. A Figura 1.18 mostra, esquematicamente, uma interrupção sendo atendida (MARTINS, 2008).

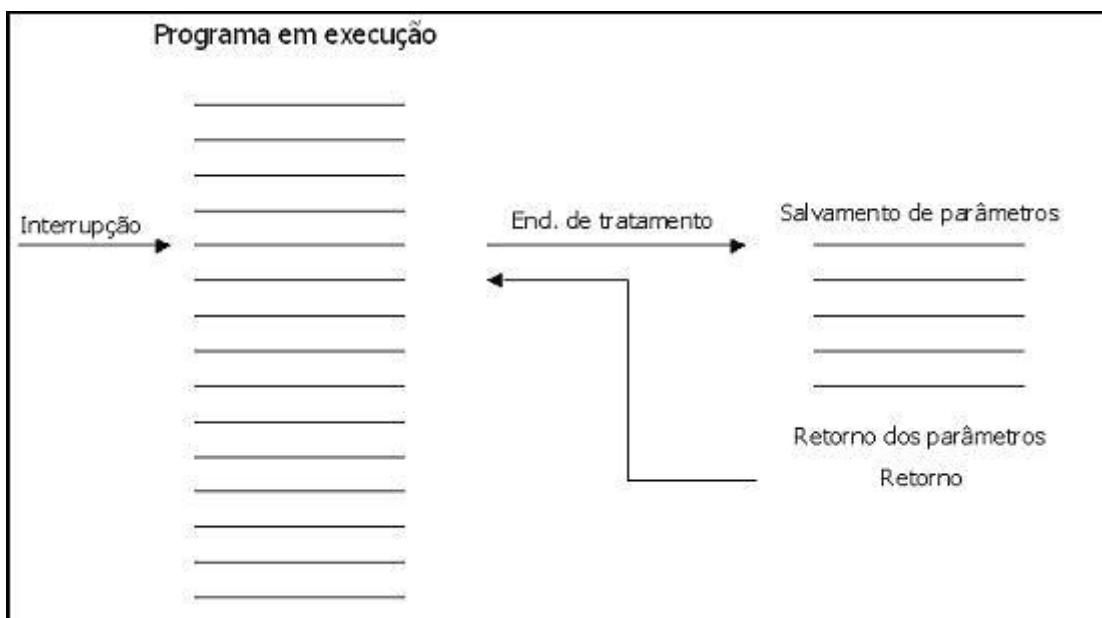


Figura 1.18 - Atendimento de interrupção.

Segundo TANEMBAUM, 2007, *Interrupções foram concebidas para evitar o desperdício de tempo computacional em loops de software (chamados polling loops) esperando eventos que serão disparados por dispositivos de modo assíncrono*. Ao invés do processador ficar esperando o evento acontecer, ele torna-se capaz de realizar outras tarefas enquanto os eventos estão pendentes. A interrupção avisa ao processador quando o evento ocorreu, permitindo uma acomodação eficiente dos dispositivos mais lentos.

1.4 MICROCONTROLADORES

Com o passar do tempo começaram a surgir dispositivos eletrônicos com funcionalidades específicas, como as máquinas de lavar, os controles remotos, os celulares, os mini games (vídeo games de bolso), sistemas de controle de alguns processos industriais, entre outros. A utilização de um microprocessador nesses dispositivos seria como utilizar um canhão para matar uma formiga no quesito velocidade e capacidade de processamento. Desta forma, surgiu a necessidade de um microprocessador com funcionalidades específicas e com capacidade de processamento bem inferior aos microprocessadores utilizados em sistemas computadorizados. Esses novos microprocessadores inicialmente ficaram conhecidos como minicomputadores ou computadores dedicados. Na evolução desses minicomputadores, foram incorporados vários periféricos (funcionalidades de outros CHIPs) em um mesmo CHIP, o que posteriormente deu origem aos **MICROCONTROLADORES**. De modo geral, um microcontrolador possui o mesmo princípio de funcionamento que um microprocessador, só que diferente dele, o microcontrolador não necessita de nenhum dispositivo (CHIP) externo para funcionar. Na verdade, podemos pensar em um microcontrolador como um CHIP que possui além do microprocessador, diversos outros dispositivos necessários para o funcionamento de um sistema computadorizado, como memórias, barramentos e periféricos. A Figura 1.19 mostra vários CHIPs, com funcionalidades diversas, sendo incorporados em um único CHIP, formando assim o microcontrolador. Os primeiros microcontroladores foram criados no final da década de 70 e atualmente seu uso é amplamente difundido. Estima-se que em um lar comum de um país desenvolvido seja possível encontrar até 24 microcontroladores em funcionamento nos diversos aparelhos eletroeletrônicos.

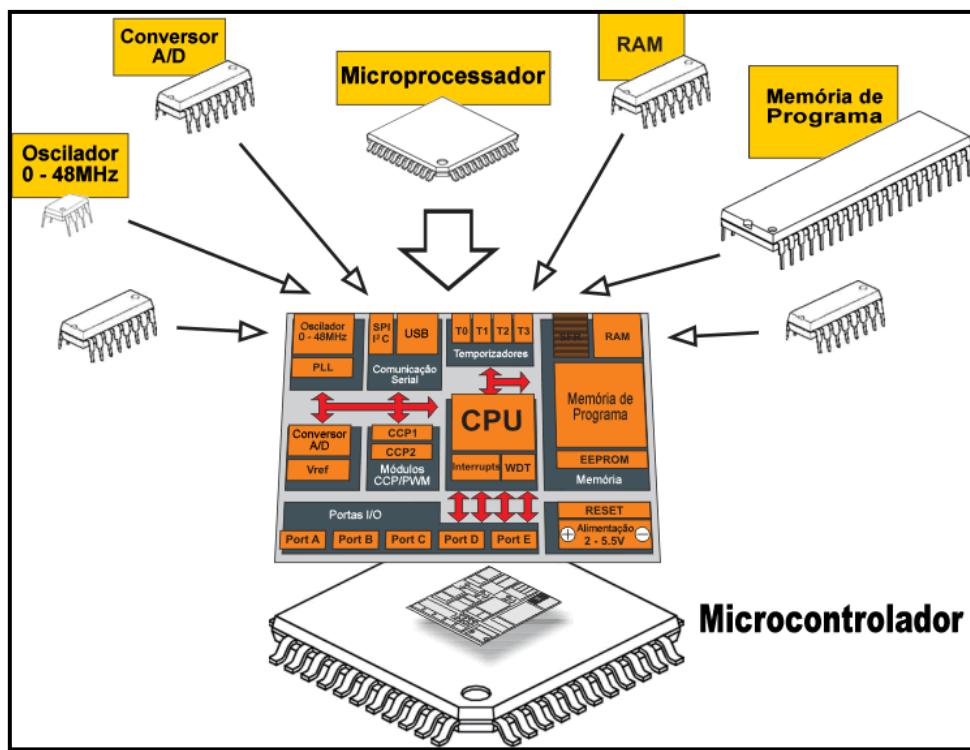


Figura 1.19 - Desenho esquemático de um Microcontrolador.

Apesar de os microcontroladores possuírem um desempenho bem inferior aos microprocessadores, o seu custo é muito menor. Além disso, são muito úteis em aplicações onde as dimensões, custo, tamanho e consumo do produto são muito importantes.

O primeiro microcontrolador produzido foi o 8048 da Intel, substituído mais tarde pela família 8051, que se tornou muito popular junto com o 6811 da Motorola. Atualmente no mercado, existem vários modelos e fornecedores desses componentes, sendo usados em veículos, equipamentos domésticos, celulares,

forno de micro-ondas, dispositivos periféricos de computadores, pequenos sistemas de controle, brinquedos, equipamentos médicos, etc.

A Figura 1.20 mostra o diagrama de blocos da arquitetura interna de um microcontrolador padrão.

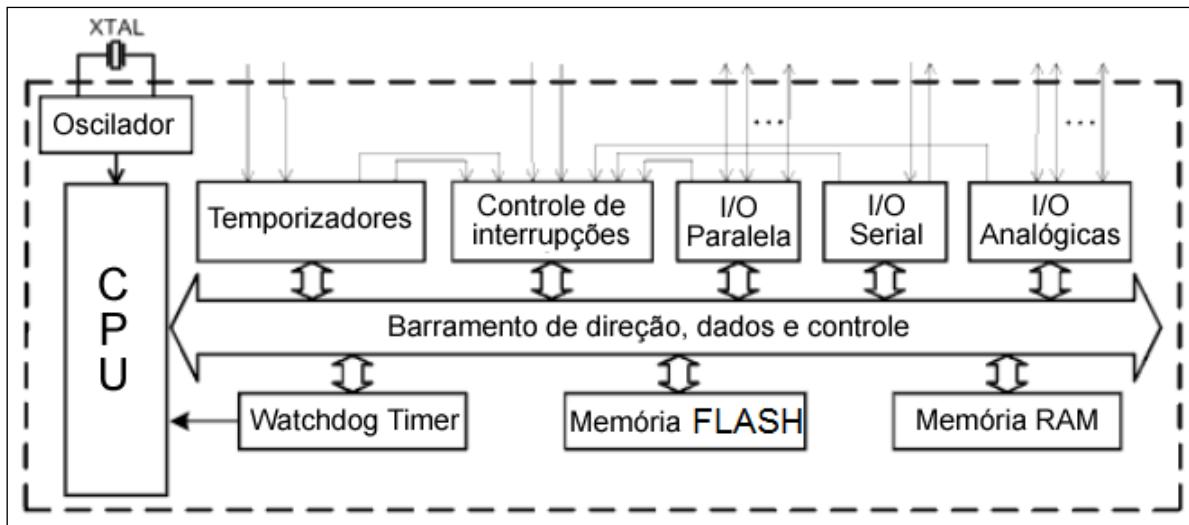


Figura 1.20 - Diagrama de blocos geral de um microcontrolador.

Os microcontroladores dispõem de um oscilador que gera os pulsos que sincronizam todas as operações internas ditando o ritmo de trabalho de todo o sistema. A velocidade de execução das instruções do programa está diretamente relacionada com a frequência do oscilador do microcontrolador (VALDÉS, PALLAS, 2007). Existem vários tipos de circuitos osciladores, como por exemplo os RC (resistivos e capacitivos) e os de cristal, sendo esses últimos mais utilizados devido sua grande estabilidade e precisão em altas frequências.

Assim como para um computador, a CPU é considerada “o cérebro” do microcontrolador. Ela é responsável por carregar as instruções do programa, uma a uma, da memória onde estão armazenadas para dentro de seus registradores de uso geral. A partir daí, a CPU realiza a decodificação, execução e armazenamento dos resultados dessas operações na memória, caso seja necessário. A CPU é constituída, basicamente, por uma Unidade Lógica Aritmética (ULA), responsável por realizar operações aritméticas e lógicas e de tomada de decisão; uma Unidade de Controle, responsável por realizar todo o controle de acesso a memória e por decodificar as instruções; e, finalmente, por seus registradores, responsáveis por armazenar dados e instruções temporariamente, enquanto estão sendo processados (VALDÉS, PALLAS, 2007).

A CPU de um microcontrolador dispõe de diferentes tipos de registradores, alguns de propósito geral e outros de propósitos específicos. Entre estes últimos estão os Registradores de Instruções, o Registrador Work, o registrador STATUS, o contador de programa, o registrador de endereço de dados e a pilha (STACK).

O Registrador de Instruções armazena a instrução que está sendo executada pela CPU. O Registrador de Instruções não é visto pelo programador.

O Registrador Work é o registrador associado às operações aritméticas e lógicas que se podem realizar na ULA. Em qualquer operação, um dos dados deve estar no registrador Work e o resultado obtido é armazenado nele.

O registrador STATUS agrupa os bits indicadores das características do resultado das operações aritméticas e lógicas realizadas na ULA e de informações referentes ao funcionamento da CPU. Entre estes indicadores está o sinal do resultado (positivo ou negativo), se o resultado é zero, se houve estouro de registradores e o tipo de paridade do resultado (PEREIRA, 2008).

O Contador de Programa é o registrador da CPU onde armazenam os endereçamentos das instruções. Ele é responsável pelo controle da sequência de instruções, cada vez que a CPU busca uma

instrução na memória, o contador de programa se incrementa, apontando assim para a próxima instrução. Em um dado instante de tempo, o contador de programa contém o endereço da instrução que será executada na continuação (PEREIRA, 2008).

O Registrador de Endereços de Dados armazena os endereços dos dados situados na memória. Este registrador é essencial para o tratamento de dados indiretos na memória.

A Pilha (STACK) tem como principal função armazenar endereços de retorno quando são utilizadas instruções de chamadas de sub-rotinas. O endereço da próxima instrução do programa é armazenado na pilha e o programa é desviado para o endereço da sub-rotina. Quando o programa é desviado para o começo de uma rotina por meio da instrução correta, o endereço seguinte ao ponto que estava sendo rodado é armazenado na pilha para que, ao fim da rotina, o programa possa retornar (SOUZA, 2005).

A memória do microcontrolador é o lugar onde estão armazenadas, dentre outras coisas, as instruções e os dados do programa. Em um microcontrolador sempre há, pelo menos, dois tipos de memórias: uma volátil de rápido acesso (memória RAM) e outra com velocidade inferior com maior capacidade de armazenamento e não volátil (memória FLASH). A RAM perde a informação armazenada quando não há alimentação de energia. Sua função é armazenar temporariamente dados e instruções a serem executadas pela CPU do microcontrolador. A memória FLASH, por sua vez, é utilizada para armazenar permanentemente todo o programa que deve ser executado pelo microcontrolador. Um número crescente de microcontroladores dispõe de alguma memória não volátil do tipo EEPROM para armazenar dados fixos que só são trocados esporadicamente.

A quantidade de memória FLASH disponível é normalmente superior a quantidade de memória RAM. Isso obedece a duas razões: a primeira está relacionada ao preço dessas memórias, sendo maior o custo por Byte das memórias RAM se comparadas com as memórias FLASH. O outro motivo é que a memória RAM deve armazenar apenas partes do programa que serão executadas naquele momento, não se fazendo necessário o armazenamento completo do código.

As entradas e saídas são particularmente importantes nos microcontroladores, pois é através delas que o microcontrolador interage com o meio exterior. O microcontrolador possui tanto entradas e saídas digitais (portas I/O, portas de comunicação serial, PWM, etc.), quanto portas analógicas associadas aos conversores A/D e D/A.

Outro recurso importante e cada vez mais comum nos microcontroladores atuais é o *Watchdog Timer*, que reinicia todo o sistema caso o programa principal apresente alguma falha.

1.5 LINGUAGEM DE MÁQUINA

Como visto, os microprocessadores são máquinas elétricas onde podemos armazenar instruções (programas) permitindo que o mesmo funcione de várias formas diferentes dependendo do que for armazenado em sua memória. Mas que instruções são essas? Como um microprocessador é capaz de entender o que o homem escreve? Para respondermos estas perguntas, primeiro devemos conhecer a língua “falada” pelos sistemas microprocessados, conhecida como linguagem binária ou código binário.

O código binário é composto de dois algarismos apenas o “0” e o “1”. Na prática, dentro de um processador (máquina elétrica), o algarismo “0” equivale a uma tensão elétrica na faixa de 0 a 0,8 Volts e o algarismo “1” equivale a uma tensão elétrica na faixa de 2 a 5 Volts (dependendo da tecnologia utilizada). Valores entre 0,8 e 2 Volts são indefinidos, ou seja, podem ser interpretados tanto como lógica “0” ou lógica “1”). Isso foi feito para simplificar os circuitos envolvidos na construção das máquinas elétricas e porque os sistemas computacionais iniciais eram valvulados, onde válvulas abertas equivaliam a lógica “0” e válvulas fechadas a lógica “1”.

Apesar de possuir apenas dois algarismos, é possível representarmos qualquer número, letra e comando com as palavras binárias (palavras constituídas de 0's e 1's), tendo como limitante apenas o

número de bits, pois quanto maior for o número de bits dessa palavra binária, maior será o número de combinações binárias (números binários diferentes). Por exemplo:

- Com **2-bits** nós só podemos criar **4 palavras binárias** diferentes: **00, 01, 10 e 11**;
- Já com **3-bits** é possível obter até **8 comandos** diferentes: **000, 001, 010, 011, 100, 101, 110 e 111**.

Desta forma, para uma palavra de **n-bits** teremos 2^n combinações diferentes que o sistema microprocessado poderá utilizar para representar números e/ou comandos.

1.6 REPRESENTAÇÃO NUMÉRICA

A matemática é uma ciência extraordinária, tudo é tão lógico e simples. O universo inteiro pode ser descrito com apenas dez dígitos diferentes (0,1,2,3,4,...9). Mas será que este é a única maneira de fazermos isso? Tem de ser exatamente dez dígitos? Claro que não, foi só uma questão de hábito. Lembram das lições da escola? Por exemplo, o que significa o número 764? Significa: 4 unidades, 6 dezenas e 7 centenas. Nós poderíamos descrever esse número de uma maneira mais complicada? Claro que sim, vejam: **4 + 60 + 700**, ou mais complicado ainda: **$4 \times 1 + 6 \times 10 + 7 \times 100$** , ou ainda, **$4 \times 10^0 + 6 \times 10^1 + 7 \times 10^2$** . Mas o que isso realmente significa? Porque nós usamos exatamente os números: 10^0 , 10^1 e 10^2 ? Por que tudo parece girar em torno do número 10? Isso acontece porque na representação numérica decimal nós utilizamos dez dígitos diferentes (0, 1, 2,..., 8, 9). Em outras palavras, porque usamos um sistema de numeração **base-10**, ou seja, **Sistema Decimal de Números**.

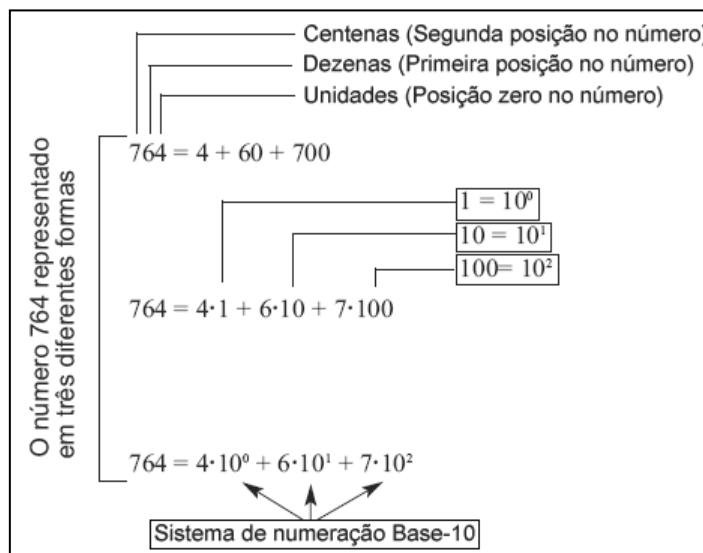


Figura 1.21 - Representação de um número na base-10.

O que aconteceria se existissem apenas dois algarismos (“0” e “1”) para representarmos todos os números? Isso seria possível? Claro que sim! A única coisa que iria mudar seria a representação desses números, ou seja, a representação numérica, que nesse caso só poderia utilizar dois algarismos diferentes, o “0” e o “1”. Fazendo uma analogia ao sistema decimal (Base 10) que possui 10 algarismos diferentes (0, 1, 2,..., 9), quando se usa apenas dois algarismos passamos a trabalhar com o **Sistema Binário de Números**, ou **Base-2**. Para descobrirmos como são representados os números na Base-2, basta utilizarmos a mesma técnica adotada na Figura 1.21, Veja o exemplo da Figura 1.22.

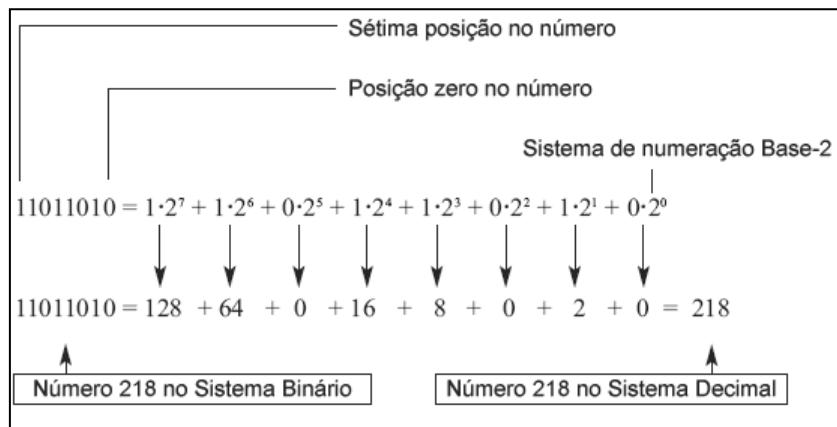


Figura 1.22 - Transformação de um número na base-2 para base-10.

Observe que a palavra binária **11011010** equivale ao número **218** no sistema decimal de numeração. Veja que para representarmos o mesmo número, o sistema binário necessita de 8 algarismos enquanto que o sistema decimal apenas 3. Essa é a principal desvantagem do sistema binário, pois exige que o sistema possua uma memória maior para armazenar a mesma quantidade de informação. Por outro lado, o sistema binário é bem menos complexo, pois só trabalha apenas com dois algarismos, permitindo que os sistemas computadorizados trabalhem com o **sistema de Tudo (lógica 1) ou Nada (lógica 0)**, conhecida como **Linguagem de Máquina** ou **Linguagem de Computador**. Desta forma, se quisermos que um computador (ou mais especificamente o processador do computador) entenda o número 218, na verdade teremos de passar para ele o número 11011010.

Ainda existe outro sistema de numeração, chamado **Hexadecimal**, utilizado para representar os programas escritos em código binário de forma mais compacta. Isso acontece porque, diferente do código binário que possui menos algarismos que o sistema decimal, o hexadecimal possui 16 algarismos, necessitando assim de palavras menores para representar o mesmo código decimal ou binário.

No início do desenvolvimento dos sistemas computadorizados, as pessoas perceberam que tinham muitas dificuldades em lidar com números binários, pois os códigos desenvolvidos em linguagem de máquina ficavam muito extensos e difíceis de serem lidos. Por isso, um novo sistema de numeração foi desenvolvido. Só que desta vez, foi criado um sistema com 16 algarismos diferentes. Os primeiros 10 dígitos são os mesmos do sistema decimal (0, 1, 2, 3,... 9) e os 6 outros dígitos foram criados utilizando as seguintes letras: A, B, C, D, E e F, formando assim um sistema de numeração Alfanumérico, chamado de **Sistema de Numeração Hexadecimal**. O algarismo A equivale ao número 10 do sistema decimal, o B equivale ao 11 e assim sucessivamente até o F que equivale ao 15. Desta forma, ficou mais simples para os projetistas entenderem os códigos de máquina que haviam desenvolvido, pois os mesmos ficaram bem mais compactos uma vez que com um algarismo do sistema Hexadecimal era possível representar até 4 algarismos do sistema Binário, por exemplo, o maior número que pode ser representado com uma palavra binária de 4-bits é o **1111** que corresponde ao número 15 no sistema decimal e **F** no hexadecimal. (Figura 1.23)

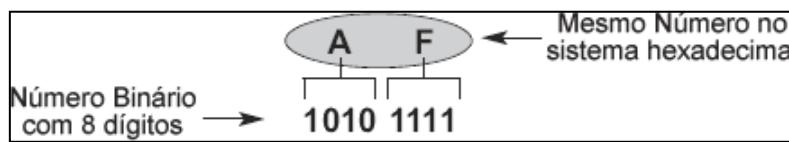


Figura 1.23 - Comparação de um número na base-16 e na base-2.

Observe na Figura 1.23 que um mesmo número com 8 algarismos no sistema binário terá apenas 2 algarismos no sistema hexadecimal. Agora imagine um código de 1.000 linhas escrito utilizando o sistema binário. Ele ficaria muito mais simples e compacto se fosse escrito em hexadecimal. Desta forma, convencionou-se que todo código de máquina seria escrito em hexadecimal para facilitar o entendimento.

1.7 CONVERSÕES DE SISTEMAS NUMÉRICOS

Agora que conhecemos quais são os sistemas numéricos mais utilizados pelos sistemas microprocessados, é muito importante sabermos converter números de um sistema de numeração para o outro, ou seja, como transformar uma série de 0's e 1's em valores comprehensíveis para nós.

1.7.1 CONVERSÃO BINÁRIA PARA DECIMAL

Os dígitos em um número binário possuem pesos diferentes de acordo com sua posição no número. Além disso, cada um desses pesos poderá estar associado a um 1 ou a um 0. Para encontrarmos o valor decimal de um número binário temos de multiplicar cada um de seus dígitos pelo peso da posição em que ele se encontra, desta forma, encontraremos o valor decimal de cada dígito. O valor do número decimal equivalente é igual à soma dos valores decimais dos dígitos binários, veja o exemplo mostrado na Figura 1.24:

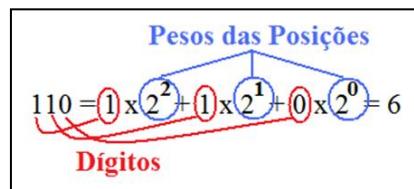


Figura 1.24 - Conversão de um número na base-2 para base-10.

É importante notarmos que para números decimais de 0 a 3, são necessários apenas dois dígitos binários para sua representação. Para valores superiores, dígitos extras devem ser acrescentados. Assim, para números de 0 a 7 são necessários 3 dígitos, 0 a 15 são necessários 4 dígitos, 0 a 31 – 5 dígitos, etc. Desta forma, para sabermos qual é o maior número decimal que podemos representar com um número binário de n dígitos, basta aplicarmos a seguinte fórmula: $2^n - 1$, veja os exemplos:

Com 2 dígitos $\Rightarrow 2^2 - 1 = 3$ (como vimos com números binários de 2 dígitos podemos representar os números decimais de 0 a 3, sendo o 3 o seu valor máximo);

Com 3 dígitos $\Rightarrow 2^3 - 1 = 7$;

Com 4 dígitos $\Rightarrow 2^4 - 1 = 15$;

Com 5 dígitos $\Rightarrow 2^5 - 1 = 31$.

Assim, usando 4 dígitos binários é possível representar números decimais de 0 a 15, incluindo estes dois dígitos, o que equivale a 16 valores diferentes no total.

O microcontrolador PIC18F4550, utilizado em um de nossos cursos, trabalha com palavras binárias de 16 dígitos, desta forma, é possível representar números decimais de 0 a 65.535 ou 65.536 valores diferentes no total.

1.7.2 CONVERSÃO HEXADECIMAL PARA DECIMAL

Para proceder à conversão de um número hexadecimal para decimal, cada dígito hexadecimal deve ser multiplicado pelo número 16 elevado ao valor da posição do dígito. (Figura 1.25)

A37E (Número no sistema hexadecimal)
$14 \cdot 16^0 = 14 \cdot 1 = 14$
$7 \cdot 16^1 = 7 \cdot 16 = 112$
$3 \cdot 16^2 = 3 \cdot 256 = 768$
$10 \cdot 16^3 = 10 \cdot 4096 = 40960$
41854 Mesmo número no sistema decimal

Figura 1.25 - Conversão de Hexadecimal para decimal.

1.7.3 CONVERSÃO HEXADECIMAL PARA BINÁRIA

Esta é a conversão mais simples que existe, não é necessário efetuar qualquer cálculo. Os dígitos hexadecimais são simplesmente substituídos pelos quatro dígitos binários adequados. O valor máximo que um dígito hexadecimal (dígito F) pode representar é o número decimal 15. Desta forma, são necessários quatro dígitos binários para representar um dígito hexadecimal qualquer. (Figura 1.26)

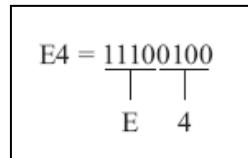


Figura 1.26 - Conversão de binário para hexadecimal.

A Tabela 1 contém os valores dos números 0-255 em três diferentes sistemas de numeração.

DEC.	BINÁRIO								HEX.
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	2
3	0	0	0	0	0	0	1	1	3
4	0	0	0	0	0	1	0	0	4
5	0	0	0	0	0	1	0	1	5
6	0	0	0	0	0	1	1	0	6
7	0	0	0	0	0	1	1	1	7
8	0	0	0	0	1	0	0	0	8
9	0	0	0	0	1	0	0	1	9
10	0	0	0	0	1	0	1	0	A
11	0	0	0	0	1	0	1	1	B
12	0	0	0	0	1	1	0	0	C
13	0	0	0	0	1	1	0	1	D
14	0	0	0	0	1	1	1	0	E
15	0	0	0	0	1	1	1	1	F
16	0	0	0	1	0	0	0	0	10
17	0	0	0	1	0	0	0	1	11
.....									
.....									
.....									
253	1	1	1	1	1	1	0	1	FD
254	1	1	1	1	1	1	1	0	FE
255	1	1	1	1	1	1	1	1	FF

Tabela 1 - Representação de números nas três bases.

O sistema de numeração hexadecimal, juntamente com os sistemas binário e decimal, é considerado o mais importante para o desenvolvimento do mundo computadorizado que vivemos. É fácil fazer conversão de qualquer número hexadecimal para binário e também é fácil de lembrar. No entanto, essas conversões podem causar confusão. Por exemplo, o que significa a afirmação: "Existem 110 produtos na linha de montagem"? Vai depender se estamos falando de um número binário, decimal ou hexadecimal,

o resultado poderia ser de 6, 110 ou 272 produtos, respectivamente. Consequentemente, a fim de evitar mal-entendidos, diferentes prefixos e sufixos são adicionados diretamente aos números. O prefixo **0x** ou **\$** bem como o sufixo **h** marcam os números em hexadecimal. Por exemplo, o número hexadecimal 10AF pode aparecer das seguintes formas: **\$10AF**, **0x10AF** ou **10AFh**. Da mesma forma, os números binários normalmente possuem o sufixo **%** ou **0b**, enquanto números decimais possuem o sufixo **D** ou, mais usualmente, aparecem sem nenhum sufixo ou prefixo.

1.8 O BIT

Teoricamente falando, um bit é a unidade básica de informação em sistemas microprocessados (computacionais). Na prática, um bit é um dígito de uma palavra binária. A exemplo do sistema decimal de numeração, onde um dígito em um número não tem o mesmo valor (por exemplo, os dígitos no número 333 são os mesmos, mas possuem valores diferentes, onde o primeiro 3 representa três unidades o segundo dígito 3 representa trinta unidades e o dígito 3 mais a esquerda do número representa trezentas unidades), a significância de cada bit depende da posição em que ele se encontra no número binário. A diferença entre os números decimais e os números binários é que nos primeiros a cada vez que um dígito é deslocado para a esquerda ele é multiplicado por **10** (Base-10), enquanto que nos binários ele será multiplicado por **2** (Base-2). Desta forma, em um número binário **111**, o **primeiro 1** (sempre da direita para a esquerda) tem o valor de **1**, o **segundo 1** tem o valor de **$1 \times 2^1 = 2$** e o **terceiro 1** tem o valor de **$1 \times 2^2 = 4$** , enquanto que para o número decimal 111, o **primeiro 1** tem o valor de **1**, o **segundo 1** tem o valor de **$1 \times 10^1 = 10$** e o **terceiro 1** tem o valor de **$1 \times 10^2 = 100$** . Desta forma, ficou convencionado que o bit mais a esquerda de uma palavra binária seria o **Bit Mais Significativo** (*Most Significant Bit – MSB*) e o bit mais a direita o **Bit Menos Significativo** (*Least Significant Bit – LSB*).

Não fique confuso se você encontrar algum bit com valor 4, 16 ou 64. Isso significa que o valor do bit está representado no sistema decimal. Seria correto dizer, por exemplo, que, "o valor do sexto bit ou bit 6 do número binário **1000000** é equivalente ao número decimal 64", pois o sexto bit (da direita pra esquerda) está na posição 6 (levando em conta que o primeiro bit está na posição zero), desta forma ele deverá ser multiplicado por 2 seis vezes ($2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^6 = 64$). Neste caso, como todos os outros dígitos são iguais a 0, o valor do número binário completo será igual ao valor do **bit 6 = $2^6 = 64$** .

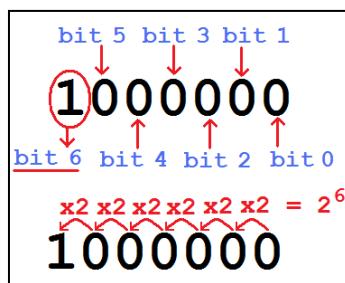


Figura 1.27 - Representação de um número na base-2.

1.9 O BYTE

Um Byte ou palavra de programa consiste em 8-bits agrupados. Todas as operações matemáticas podem ser realizadas com eles da mesma forma que são feitas com os números decimais. Tal como acontece com qualquer outro dígito de um número qualquer, os dígitos de um Byte não possuem o mesmo valor. Como visto, o bit de maior valor é o bit mais a esquerda (MSB). Portanto, o bit de menor valor é o bit mais a direita (LSB). Como em um Byte temos 8 bits, é possível realizar até 256 combinações diferentes, ou seja, um Byte pode representar até 256 ($2^8 = 256$) números binários diferentes, sendo **255** o maior número decimal que pode ser representado.

Um nibble é o nome dado a meio Byte (4-bits). Existem dois tipos de nibble, o “low” e o “high”, onde o primeiro é o nome dado a metade menos significativa do Byte (os primeiros 4-bits) e o último representa a metade do Byte mais significativa (os últimos 4-bits). (Figura 1.28)

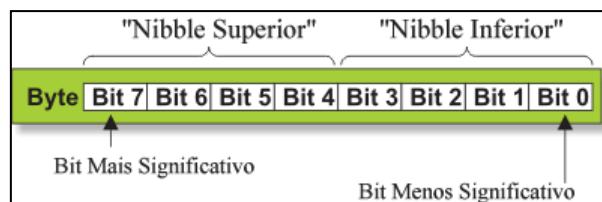


Figura 1.28 - Representação de nibbles.

1.10 PORTAS LÓGICAS

São circuitos eletrônicos existentes dentro dos microprocessadores e são responsáveis por realizar as operações lógicas, aritméticas e de tomada de decisão.

Alguma vez você já se perguntou o que existe dentro de um microprocessador ou um CHIP digital? Como é que esses CHIPS conseguem realizar operações lógicas, aritméticas e de tomada de decisão? Você sabia que por trás dessa aparência tão complicada existem apenas alguns elementos diferentes chamados "circuitos lógicos" ou "portas lógicas"?

O funcionamento destes elementos foi baseado nos princípios estabelecidos pelo matemático britânico George **Boole** em meados do século 19, antes mesmo da primeira lâmpada ser inventada. Em suma, a ideia principal foi a de expressar funções lógicas por meio de funções algébricas. Esse pensamento foi logo transformado em um produto prático, que muito mais tarde evoluiu para o que hoje conhecemos como circuitos lógicos E (**AND**), OU (**OR**) e NÃO (**NOT**).

O princípio de funcionamento destes elementos ficou conhecido como **Álgebra Booleana**. Uma vez que algumas instruções de programas utilizadas por microprocessadores e microcontroladores fazem uso das funcionalidades destas portas lógicas, é importante estudarmos o princípio de funcionamento de cada uma delas.

1.10.1 PORTA AND

Uma porta lógica "AND" tem duas ou mais entradas e apenas uma saída. Vamos supor que a porta utilizada neste caso tenha apenas duas entradas, **A** e **B**. O princípio de funcionamento desta porta diz que só aparecerá a lógica um (1) na saída da porta quanto todas as portas de entrada forem também iguais a um (1), ou seja, somente quando A e B forem iguais a 1 a saída será igual a 1. A Figura 1.29 mostra o símbolo representativo da porta lógica AND e a tabela de dependências entre as entradas e a saída, **Tabela Verdade**.

A	B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Figura 1.29 - Símbolo gráfico e Tabela verdade da porta AND.

Quando usado em um programa, a operação lógica **AND** é executada por uma instrução de programa, o que será discutido mais adiante no capítulo que trata da programação do microcontrolador em linguagem C. Por enquanto, basta lembrar que a lógica **AND**, em um programa, será utilizada para relacionar os bits de dois registros ou variáveis diferentes. Veja o exemplo da Figura 1.30, onde é realizada

uma operação lógica AND entre as variáveis A e B, bit a bit, e os bits do resultado só foram iguais a 1 quanto o bit de A e o bit de B eram iguais a 1, o que aconteceu apenas no **bit3** e no **bit6**.

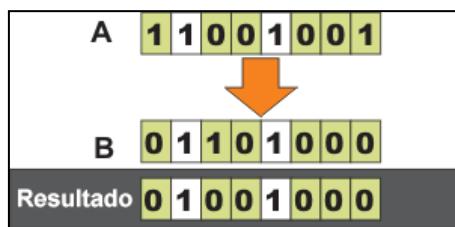


Figura 1.30 - Operação lógica AND.

1.10.2 PORTA OR

Da mesma forma que o caso anterior, as portas **OR** também podem ter duas ou mais entradas e uma saída. A lógica um (1) aparecerá na porta de saída se **qualquer** uma das entradas (**A** ou **B**) estiverem em lógica um (1). Somente se todas as entradas estiverem em lógica zero (0), a saída será modificada para lógica zero (0). A Figura 1.31 mostra o símbolo representativo da porta lógica **OR** e sua **Tabela Verdade**.

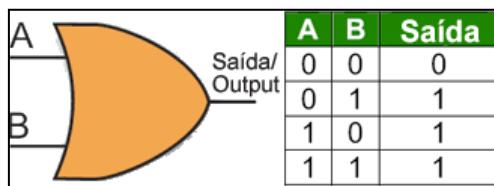


Figura 1.31 - Símbolo gráfico e Tabela verdade da porta OR.

Em instruções de programa a operação lógica OR é utilizada para relacionar variáveis ou registradores bit a bit. Veja o exemplo da Figura 1.32.

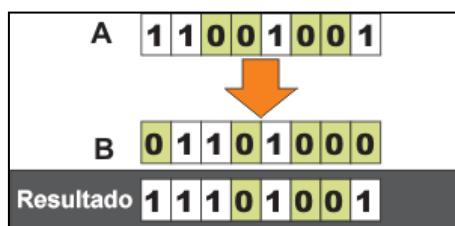


Figura 1.32 - Operação lógica OR.

1.10.3 PORTA NOT

Esta porta lógica tem apenas uma porta de entrada e uma de saída. Ela funciona de uma maneira extremamente simples. Quando é aplicada lógica zero (0) na entrada, a lógica um (1) aparecerá na saída e vice-versa. Isto significa que esta porta inverte o sinal aplicado a ela. É às vezes é chamada de porta inversora. (Figura 1.33)

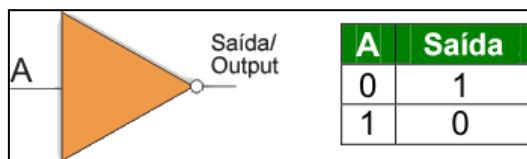


Figura 1.33 - Símbolo gráfico e Tabela verdade da porta NOT.

Se um programa realiza uma operação lógica **NOT** em um Byte, o resultado será um Byte com todos os bits invertidos. (Figura 1.34)

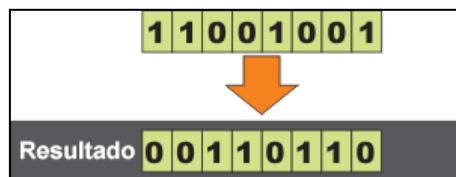


Figura 1.34 - Operação lógica NOT.

1.10.4 PORTA XOR (OU-EXCLUSIVA)

A porta **XOR** é um pouco mais complicada em comparação com outras portas. Ela representa a combinação de todas as outras portas descritas anteriormente. Ela também pode ter duas ou mais entradas e somente uma saída. A lógica um (1) aparece na saída apenas quando uma das entradas está em lógica um (1) e as outras em lógica zero (0). A Figura 1.35 mostra o símbolo representativo da porta lógica **XOR** e sua **Tabela Verdade**.

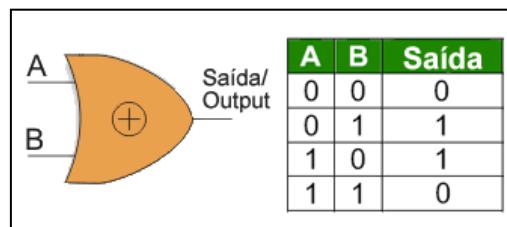


Figura 1.35 - Símbolo gráfico e Tabela verdade da porta XOR.

Em um programa, esta operação é geralmente utilizada para comparar duas variáveis, se o resultado da operação for 0, significa que as duas variáveis são iguais. Observe que na Figura 1.36 os bits do resultado só são iguais a 0 quando os bits equivalentes das variáveis **A** e **B** são iguais, independente se são 1 ou 0.

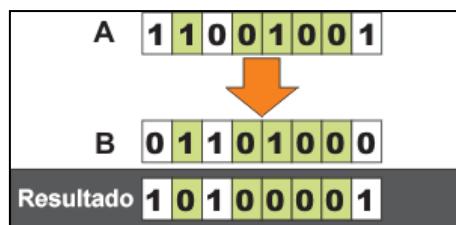
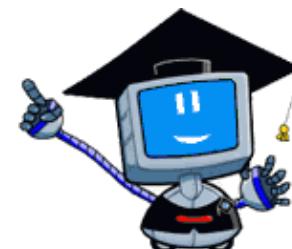


Figura 1.36 - Operação lógica XOR.

Neste capítulo, falamos sobre:

- ✚ Introdução histórica sobre o surgimento dos microprocessadores;
- ✚ Sistemas microprocessados e microcontrolados;
- ✚ Linguagem de máquina, representação numérica;
- ✚ Portas lógicas.



Capítulo 2 – Arquitetura do PIC e Kit de Desenvolvimento

Microcontroladores (Símbologia µC) podem ser definidos como um “pequeno” componente eletrônico, dotado de uma “inteligência” programável, utilizado no controle de processos lógicos (SOUZA, 2005). Para entendermos melhor esta definição, vamos analisá-lo por partes:

- ✓ O controle de processos deve ser entendido como o controle de periféricos, tais como: LEDs, botões, displays de 7-segmentos, displays de cristal líquido (LCD), relés, sensores diversos (pressão, temperatura, etc.) e muitos outros. São chamados de controles lógicos, pois a operação do sistema baseia-se nas ações lógicas que devem ser executadas, dependendo do estado dos periféricos de entrada e/ou saída.
- ✓ O microcontrolador é programável, pois toda a lógica de operação de que acabamos de falar é estruturada na forma de um programa e gravada dentro do componente. Depois disso, toda vez que o microcontrolador for energizado, o programa armazenado internamente será executado.
- ✓ Quanto à “inteligência” do componente, podemos associá-la ao fato de os programas, a partir dos quais o microcontrolador orienta suas ações, serem feitos a partir da inteligência humana. É como se o homem emprestasse um pouco de sua inteligência a uma máquina por meio de algumas linhas de código, permitindo a ela tomar decisões dentro de um universo restrito de eventos. Além disso, a CPU do microcontrolador possui uma Unidade Lógica e Aritmética (ULA), que é capaz de executar todas as operações matemáticas e lógicas requisitadas pelo programa. Quanto mais poderosa a ULA do componente, maior sua capacidade de processar informações.
- ✓ Em nossa definição, o microcontrolador ganhou ainda o adjetivo “pequeno”, pois em uma única pastilha de silício encapsulada (popularmente chamada de CI ou CHIP), temos todos os componentes necessários ao controle de um processo, ou seja, o microcontrolador está provido internamente de memória de programa, memória de dados, portas de entrada e/ou saída, CPU e periféricos diversos como: timers, contadores, comunicação serial e paralela, PWMs, conversores analógico-digitais, entre outros.

Atualmente, muitos equipamentos de nosso uso diário, tais como: eletrodomésticos, videocassete, alarmes, celulares, DVDs, brinquedos, entre outros, utilizam microcontroladores para execução de suas funções básicas. Portanto, pode ser que você nem sabia, mas esses componentes já fazem parte da sua vida há um bom tempo (SOUZA, 2005). Hoje, temos em média 24 microcontroladores em uma casa com 4 (quatro) pessoas.

Os microcontroladores PIC (*Peripheral Interface Controller*), fabricados pela *Microchip Technology*, podem processar dados de até 32-bits, dependendo do tipo e família a que pertencem, com uma extensa variedade de modelos e periféricos internos. Tiveram sua origem em 1965, na divisão eletrônica da General Instruments (GI). No inicio dos anos 70, essa empresa introduziu no mercado um microcontrolador de 16-bits chamado CP1600, contudo, ele era pobre em I/O e lento em processamento. Por volta de 1975, a GI introduziu no mercado uma versão aprimorada do PIC, com memória de programa reprogramável e diversos outros aprimoramentos, sob a sigla PIC 16C50 (WIKIPÉDIA.ORG).

O PIC representa uma família de microcontroladores projetados sob a arquitetura Harvard, usando um conjunto de instruções reduzido (RISC).

Os microcontroladores podem ser projetados segundo duas arquiteturas:

- ✓ Von-Neumann: apenas um barramento interno, pelo qual trafegam tanto instruções quanto dados (Figura 2.1);

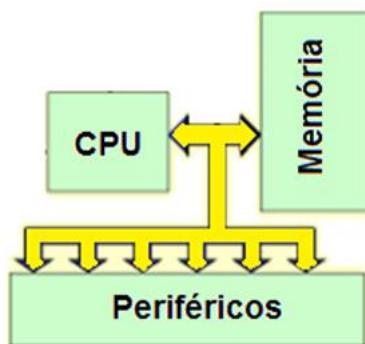


Figura 2.1 - Arquitetura Von-Neumann

- ✓ Harvard: dois barramentos, sendo um exclusivo para tráfego de dados e outro para instruções. No caso dos microcontroladores PIC, o barramento de dados podem ter de 8-bits a 32-bits e o de instruções pode ter 12, 14 ou 16-bits como mostrado na Figura 2.2;

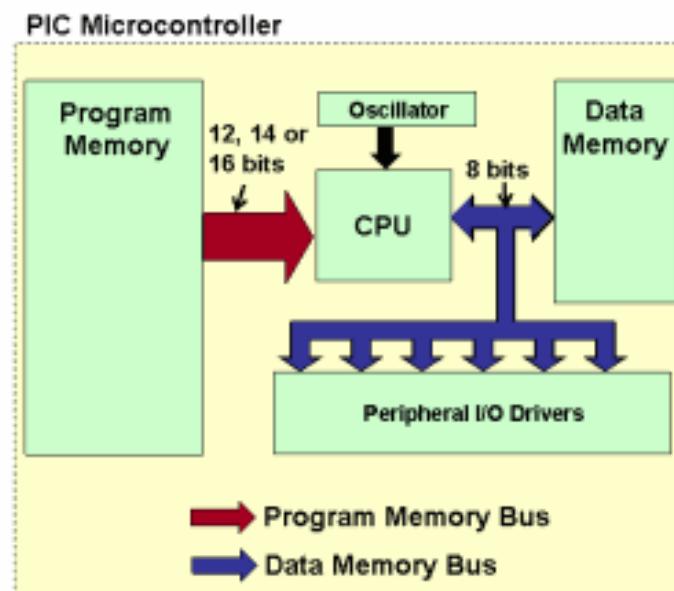


Figura 2.2 - Arquitetura Harvard.

Na arquitetura Harvard pode-se executar uma instrução enquanto outra está sendo carregada. Diferentemente do que ocorre com o Von-Neumann. Além disso, como os barramentos são separados, há a necessidade de a instrução conter o endereço dos dados a serem processados (SOUZA, 2005). Como consequência o número das instruções possíveis fica limitado. Devida a essa limitação foi utilizado CPUs com tecnologia RISC (*Reduced Instruction Set Computer* – Computador com conjunto de instruções reduzido). Existe ainda a tecnologia CISC (*Complex Instruction Set Computer* – Computador com conjunto de instruções complexo), muito utilizada em computadores com tecnologia Von-Neumann.

Os PICs da Microchip são os microcontroladores mais vendidos no mundo. Isso se deve a algumas características, tais como: grande variedade de famílias que permitem adaptar o μ C às mais variadas necessidades existentes nos tempos modernos, ferramentas de desenvolvimento comuns para todas as famílias, grande variedade de funcionalidades embutidas, preços competitivos, bom suporte (datasheet, livros, notas de aplicações, seminários e muita informação disponível na internet) e principalmente pelo fato de a *Microchip* ter investido em suporte para usuários iniciantes e encapsulamento dos CHIPS que permitiam sua montagem em *proto-boards*.

Os μCs PIC possuem famílias com núcleos de processamento de 12-bits, 16-bits e 32-bits trabalham em velocidades de *clock* de até 80MHz, o que pode permitir uma velocidade de processamento de até 120 DMIPS. Possibilita o reconhecimento de interrupções tanto externas quanto provenientes de periféricos internos. Funcionam com tensões de alimentação de 3 a 6V e os modelos possuem encapsulamento de 6 a 100 pinos em diversos formatos (SOT23, DIP, SOIC, TQFP, etc.) (WIKIPÉDIA.ORG).

Os PICs podem ser programados em linguagem mnemônica (*assembly*) ou usando-se compiladores de linguagem de alto nível (Pascal, C, Basic) que geram um código em formato hexadecimal (Intel Hex format ou linguagem de máquina) que podem ser gravados na memória de programa desses microcontroladores. Para tal procedimento, utiliza-se um hardware especial (gravador externo de PIC) acoplado a um PC, ou ainda, processos de autogravação por meio de *bootloader* (ver seção 3.5) (WIKIPÉDIA.ORG).

Abaixo temos as primeiras funcionalidades internas dos PICs. Essas características variam de acordo com o modelo e família do μC PIC:

- ✓ Conversores Analógico-Digitais de 8 a 12-bits;
- ✓ Contadores e timers de 8 e 16-bits;
- ✓ Comparadores Analógicos;
- ✓ USARTs (**Universal Synchronous Asynchronous Receiver Transmitter**, significando Transmissor/Receptor Universal Síncrono e Assíncrono);
- ✓ Controladores de comunicação I2C, SPI, USB;
- ✓ Controladores PWM;
- ✓ Controladores de LCD;
- ✓ Controladores de motores;
- ✓ Gerador de energia de alta potência;
- ✓ Periféricos para LIN, CAN;
- ✓ Controladores Ethernet;
- ✓ Periféricos IRDA;
- ✓ Codificadores para criptografia Keeloq;
- ✓ *Watchdog timer*;
- ✓ Detectores de falha na alimentação;
- ✓ Portas digitais com capacidade de 25mA (fornecer ou drenar) para acionar circuitos externos;
- ✓ Osciladores internos.

2.1 PIC18F4550 E PIC32MX775F256L

Os PIC18F4550 e PIC32MX775F256L são os microcontroladores usados em nossos cursos. A escolha desses μCs se deve a algumas de suas características que os tornam os melhores de suas famílias. No caso do PIC18F4550, podemos citar a facilidade de encontrá-lo no comércio, o seu custo acessível, a facilidade de gravação, possuir encapsulamento PDIP (o que permite sua montagem em *proto-board*) e possuir periféricos que permitem o desenvolvimento das mais modernas soluções, tais como, comunicação USB. O PIC32MX775F256L foi escolhido por apresentar alto desempenho de processamento associado a uma grande variedade de periféricos, o que permite a realização das práticas mais modernas quando o assunto é sistema microcontrolador, como por exemplo, processamento de áudio e vídeo, controle de telas gráficas coloridas dotadas de tecnologia *touch-screen*, dentre outros.

2.1.1. PIC18F4550

A seguir são apresentadas algumas das características do PIC18F4550:

- ✓ Memória FLASH para armazenamento de programa: 32 KBytes;
- ✓ Memória SRAM para armazenamento de dados: 2 KBytes;

- ✓ Memória EEPROM de dados: 256 Bytes;
- ✓ Portas configuráveis como entradas ou saídas digitais: 35;
- ✓ Portas configuráveis como canais de entradas analógicas: 13
- ✓ Módulo CCP (Capture/Compare/PWM)
- ✓ Porta paralela de 8-bits (SPP – *Streaming Parallel Port*).
- ✓ Temporizadores de 8 e 16-bit: 4;
- ✓ *Watchdog Timer*;
- ✓ Frequência de operação de até 48MHz;
- ✓ Múltiplas fontes de interrupção (20);
- ✓ Dois comparadores;
- ✓ Periféricos avançados de comunicação: Porta de comunicação serial, Porta de comunicação USB 2.0;
- ✓ Arquitetura Harvard, tecnologia RISC com um conjunto de 75 instruções;
- ✓ Pilha de 31 níveis.

Ver *datasheet* do PIC18F4550 na página 3.

Funcionalidades	PIC18F2455	PIC18F2550	PIC18F4455	PIC18F4550
Operating Frequency	DC – 48 MHz			
Program Memory (Bytes)	24576	32768	24576	32768
Program Memory (Instructions)	12288	16384	12288	16384
Data Memory (Bytes)	2048	2048	2048	2048
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	19	19	20	20
I/O Ports	Ports A, B, C, (E)	Ports A, B, C, (E)	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	2	2	1	1
Enhanced Capture/Compare/PWM Modules	0	0	1	1
Serial Communications	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART	MSSP, Enhanced USART
Universal Serial Bus (USB) Module	1	1	1	1
Streaming Parallel Port (SPP)	No	No	Yes	Yes
10-bit Analog-to-Digital Module	10 Input Channels	10 Input Channels	13 Input Channels	13 Input Channels
Comparators	2	2	2	2
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT			
Programmable Low-Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions; 83 with Extended Instruction Set enabled			
Packages	28-pin PDIP 28-pin SOIC	28-pin PDIP 28-pin SOIC	40-pin PDIP 44-pin QFN 44-pin TQFP	40-pin PDIP 44-pin QFN 44-pin TQFP

Tabela 2 - Funcionalidades do PIC18F4550 (fonte *datasheet* PIC18F4550).

Na Figura 2.3 é possível visualizar a pinagem do microcontrolador PIC18F4550 com as portas de entrada e saída (RA, RB, RC, RD e RE), que podem ser configuradas por meio do programa, os canais de entradas analógicas (AN), pinos de alimentação (VDD e VSS), pinos de entrada para o oscilador externo (OSC), porta de comunicação serial (RX e TX) e porta de comunicação USB (D+ e D-), entre outros. É possível ver também que vários pinos acumulam mais de uma função (*Datasheet PIC18F4550*).

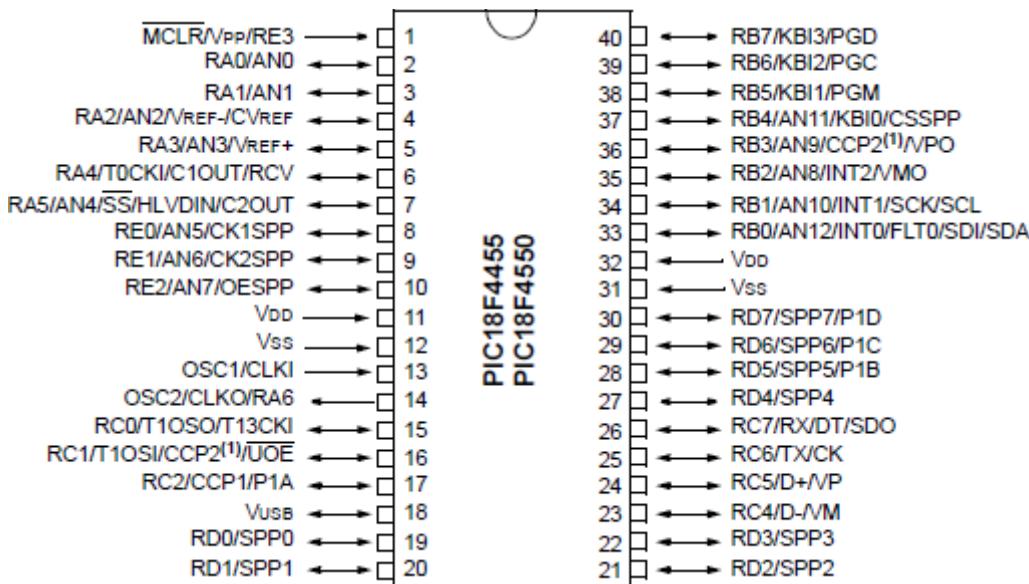


Figura 2.3- Distribuição dos pinos no microcontrolador PIC18F4550 (Fonte: datasheet PIC18F4550).

O PIC18F4550 possui até 35 pinos de I/O configuráveis, que estão agrupados em cinco grupos denominados **PORTE**s. Desta forma, temos a **PORT A**, a **PORT B**, a **PORT C**, a **PORT D** e a **PORT E**.

A maioria desses pinos podem ser configurados como entrada ou saída (input/output), e como já foi dito, alguns deles acumulam outras funções.

Podemos ver mais detalhes sobre cada pino na Tabela 3 (dividida em partes), que segue abaixo (Ver *datasheet* do PIC18F4550 da página 18 à 23):

PIC18F4455/4550 PINOUT I/O DESCRIPTIONS

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	PDIP	QFN	TQFP			
MCLR/VPP/RE3 MCLR	1	18	18	I	ST	Master Clear (input) or programming voltage (input). Master Clear (Reset) input. This pin is an active-low Reset to the device. Programming voltage input. Digital input.
VPP RE3				P I	ST	
OSC1/CLKI OSC1 CLKI	13	32	30	I I	Analog Analog	Oscillator crystal or external clock input. Oscillator crystal input or external clock source input. External clock source input. Always associated with pin function OSC1. (See OSC2/CLKO pins.)
OSC2/CLKO/RA6 OSC2 CLKO RA6	14	33	31	O O	— —	Oscillator crystal or clock output. Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. In RC mode, OSC2 pin outputs CLKO which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate. General purpose I/O pin.

Legend: TTL = TTL compatible input CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels I = Input
O = Output P = Power

Note 1: Alternate assignment for CCP2 when CCP2MX configuration bit is cleared.

2: Default assignment for CCP2 when CCP2MX configuration bit is set.

3: These pins are No Connect unless the ICPRT configuration bit is set. For NC/ICPORTS, the pin is No Connect unless ICPRT is set and the DEBUG configuration bit is cleared.

PIC18F4455/4550 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	PDIP	QFN	TQFP			
RA0/AN0 RA0 AN0	2	19	19	I/O I	TTL Analog	PORTA is a bidirectional I/O port. Digital I/O. Analog input 0.
RA1/AN1 RA1 AN1	3	20	20	I/O I	TTL Analog	Digital I/O. Analog input 1.
RA2/AN2/VREF-/CVREF RA2 AN2 VREF- CVREF	4	21	21	I/O I I O	TTL Analog Analog Analog	Digital I/O. Analog input 2. A/D reference voltage (low) input. Analog comparator reference output.
RA3/AN3/VREF+ RA3 AN3 VREF+	5	22	22	I/O I I	TTL Analog Analog	Digital I/O. Analog input 3. A/D reference voltage (high) input.
RA4/T0CKI/C1OUT/RCV RA4 T0CKI C1OUT RCV	6	23	23	I/O I O I	ST ST — TTL	Digital I/O. Timer0 external clock input. Comparator 1 output. External USB transceiver RCV input.
RA5/AN4/SS/ HLVDIN/C2OUT RA5 AN4 SS HLVDIN C2OUT	7	24	24	I/O I I O	TTL Analog TTL Analog —	Digital I/O. Analog input 4. SPI™ slave select input. High/Low-Voltage Detect input. Comparator 2 output.
RA6	—	—	—	—	—	See the OSC2/CLK0/RA6 pin.

Legend: TTL = TTL compatible input

CMOS = CMOS compatible input or output

ST = Schmitt Trigger input with CMOS levels

I = Input

O = Output

P = Power

Note 1: Alternate assignment for CCP2 when CCP2MX configuration bit is cleared.

2: Default assignment for CCP2 when CCP2MX configuration bit is set.

3: These pins are No Connect unless the ICPRT configuration bit is set. For NC/ICPORTS, the pin is No Connect unless ICPRT is set and the DEBUG configuration bit is cleared.

PIC18F4455/4550 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	PDIP	QFN	TQFP			
RB0/AN12/INT0/ FLT0/SDI/SDA	33	9	8	I/O	TTL	PORTB is a bidirectional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs.
RB0				I	Analog	Digital I/O.
AN12				I	ST	Analog Input 12.
INT0				I	ST	External Interrupt 0.
FLT0				I	ST	Enhanced PWM Fault Input (ECCP1 module).
SDI				I	ST	SPI™ data In.
SDA				I/O	ST	I²C™ data I/O.
RB1/AN10/INT1/SCK/ SCL	34	10	9	I/O	TTL	Digital I/O.
RB1				I	Analog	Analog Input 10.
AN10				I	ST	External Interrupt 1.
INT1				I/O	ST	Synchronous serial clock Input/output for SPI mode.
SCK				I/O	ST	Synchronous serial clock Input/output for I²C mode.
RB2/AN8/INT2/VMO	35	11	10	I/O	TTL	Digital I/O.
RB2				I	Analog	Analog Input 8.
AN8				I	ST	External Interrupt 2.
INT2				O	—	External USB transceiver VMO output.
RB3/AN9/CCP2/VPO	36	12	11	I/O	TTL	Digital I/O.
RB3				I	Analog	Analog Input 9.
AN9				I/O	ST	Capture 2 Input/Compare 2 output/PWM 2 output.
CCP2 ⁽¹⁾				O	—	External USB transceiver VPO output.
RB4/AN11/KBI0/CSSPP	37	14	14	I/O	TTL	Digital I/O.
RB4				I	Analog	Analog Input 11.
AN11				I	TTL	Interrupt-on-change pin.
KBI0				O	—	SPP chip select control output.
CSSPP				—	—	—
RB5/KBI1/PGM	38	15	15	I/O	TTL	Digital I/O.
RB5				I	TTL	Interrupt-on-change pin.
KBI1				I/O	ST	Low-Voltage ICSP™ Programming enable pin.
PGM				—	—	—
RB6/KBI2/PGC	39	16	16	I/O	TTL	Digital I/O.
RB6				I	TTL	Interrupt-on-change pin.
KBI2				I/O	ST	In-Circuit Debugger and ICSP programming clock pin.
PGC				—	—	—
RB7/KBI3/PGD	40	17	17	I/O	TTL	Digital I/O.
RB7				I	TTL	Interrupt-on-change pin.
KBI3				I/O	ST	In-Circuit Debugger and ICSP programming data pin.
PGD				—	—	—

Legend: TTL = TTL compatible input
 ST = Schmitt Trigger Input with CMOS levels
 O = Output

CMOS = CMOS compatible Input or output
 I = Input
 P = Power

Note 1: Alternate assignment for CCP2 when CCP2MX configuration bit is cleared.

2: Default assignment for CCP2 when CCP2MX configuration bit is set.

3: These pins are No Connect unless the ICPRT configuration bit is set. For NC/ICPORTS, the pin is No Connect unless ICPRT is set and the DEBUG configuration bit is cleared.

PIC18F4455/4550 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	PDIP	QFN	TQFP			
RC0/T1OS0/T13CKI RC0 T1OS0 T13CKI	15	34	32	I/O O I	ST — ST	PORTC is a bidirectional I/O port. Digital I/O. Timer1 oscillator output. Timer1/Timer3 external clock input.
RC1/T1OS1/CCP2/ UOE RC1 T1OS1 CCP2 ⁽²⁾ UOE	16	35	35	I/O I I/O O	ST CMOS ST —	Digital I/O. Timer1 oscillator input. Capture 2 input/Compare 2 output/PWM 2 output. External USB transceiver OE output.
RC2/CCP1/P1A RC2 CCP1 P1A	17	36	36	I/O I/O O	ST ST TTL	Digital I/O. Capture 1 input/Compare 1 output/PWM 1 output. Enhanced CCP1 PWM output, channel A.
RC4/D-/VM RC4 D- VM	23	42	42	I I/O I	TTL — TTL	Digital input. USB differential minus line (input/output). External USB transceiver VM input.
RC5/D+/VP RC5 D+ VP	24	43	43	I I/O I	TTL — TTL	Digital input. USB differential plus line (input/output). External USB transceiver VP input.
RC6/TX/CK RC6 TX CK	25	44	44	I/O O I/O	ST — ST	Digital I/O. EUSART asynchronous transmit. EUSART synchronous clock (see RX/DT).
RC7/RX/DT/SDO RC7 RX DT SDO	26	1	1	I/O I I/O O	ST ST ST —	Digital I/O. EUSART asynchronous receive. EUSART synchronous data (see TX/CK). SPI™ data out.

Legend: TTL = TTL compatible input CMOS = CMOS compatible input or output
 ST = Schmitt Trigger input with CMOS levels I = Input
 O = Output P = Power

Note 1: Alternate assignment for CCP2 when CCP2MX configuration bit is cleared.

2: Default assignment for CCP2 when CCP2MX configuration bit is set.

3: These pins are No Connect unless the ICPRT configuration bit is set. For NC/ICPORTS, the pin is No Connect unless ICPRT is set and the DEBUG configuration bit is cleared.

PIC18F4455/4550 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	PDIP	QFN	TQFP			
RD0/SPP0 RD0 SPP0	19	38	38	I/O	ST	PORTD is a bidirectional I/O port or a Streaming Parallel Port (SPP). These pins have TTL input buffers when the SPP module is enabled.
RD1/SPP1 RD1 SPP1	20	39	39	I/O	ST	Digital I/O. Streaming Parallel Port data.
RD2/SPP2 RD2 SPP2	21	40	40	I/O	ST	Digital I/O. Streaming Parallel Port data.
RD3/SPP3 RD3 SPP3	22	41	41	I/O	ST	Digital I/O. Streaming Parallel Port data.
RD4/SPP4 RD4 SPP4	27	2	2	I/O	ST	Digital I/O. Streaming Parallel Port data.
RD5/SPP5/P1B RD5 SPP5 P1B	28	3	3	I/O	ST	Digital I/O. Streaming Parallel Port data.
RD6/SPP6/P1C RD6 SPP6 P1C	29	4	4	I/O	ST	Enhanced CCP1 PWM output, channel B.
RD7/SPP7/P1D RD7 SPP7 P1D	30	5	5	I/O	ST	Digital I/O. Streaming Parallel Port data.
				I/O	TTL	Enhanced CCP1 PWM output, channel C.
				I/O	TTL	Enhanced CCP1 PWM output, channel D.

Legend: TTL = TTL compatible input CMOS = CMOS compatible input or output

ST = Schmitt Trigger input with CMOS levels

O = Output

I = Input

P = Power

Note 1: Alternate assignment for CCP2 when CCP2MX configuration bit is cleared.

2: Default assignment for CCP2 when CCP2MX configuration bit is set.

3: These pins are No Connect unless the ICPRT configuration bit is set. For NC/ICPORTS, the pin is No Connect unless ICPRT is set and the DEBUG configuration bit is cleared.

PIC18F4455/4550 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	PDIP	QFN	TQFP			
RE0/AN5/CK1SPP RE0 AN5 CK1SPP	8	25	25	I/O I O	ST Analog —	PORTE is a bidirectional I/O port. Digital I/O. Analog input 5. SPP clock 1 output.
RE1/AN6/CK2SPP RE1 AN6 CK2SPP	9	26	26	I/O I O	ST Analog —	Digital I/O. Analog input 6. SPP clock 2 output.
RE2/AN7/OESPP RE2 AN7 OESPP	10	27	27	I/O I O	ST Analog —	Digital I/O. Analog input 7. SPP output enable output.
RE3	—	—	—	—	—	See MCLR/VPP/RE3 pin.
V _{ss}	12, 31	6, 30, 31	6, 29	P	—	Ground reference for logic and I/O pins.
V _{dd}	11, 32	7, 8, 28, 29	7, 28	P	—	Positive supply for logic and I/O pins.
V _{usb}	18	37	37	O	—	Internal USB 3.3V voltage regulator output.
NC/ICCK/ICPGC ICCK ICPGC	—	—	12	I/O I/O	ST ST	No Connect or dedicated ICD/ICSP™ port clock. ⁽³⁾ In-Circuit Debugger clock. ICSP programming clock.
NC/ICDT/ICPGD ICDT ICPGD	—	—	13	I/O I/O	ST ST	No Connect or dedicated ICD/ICSP port clock. ⁽³⁾ In-Circuit Debugger data. ICSP programming data.
NC/ICRST/ICVPP ICRST ICVPP	—	—	33	I P	—	No Connect or dedicated ICD/ICSP port Reset. ⁽³⁾ Master Clear (Reset) input. Programming voltage input.
NC/ICPORTS ICPORTS	—	—	34	P	—	No Connect or 28-pin device emulation. ⁽³⁾ Enable 28-pin device emulation when connected to V _{ss} .
NC	—	13	—	—	—	No Connect.

Legend: TTL = TTL compatible input

CMOS = CMOS compatible input or output

ST = Schmitt Trigger input with CMOS levels

I = Input

O = Output

P = Power

Note 1: Alternate assignment for CCP2 when CCP2MX configuration bit is cleared.

2: Default assignment for CCP2 when CCP2MX configuration bit is set.

3: These pins are No Connect unless the ICPRT configuration bit is set. For NC/ICPORTS, the pin is No Connect unless ICPRT is set and the DEBUG configuration bit is cleared.

Tabela 3 - Pinos do PIC18F4550 (Fonte: datasheet do PIC18F4550).

ARQUITETURA INTERNA DO PIC18F4550

No diagrama da Figura 2.4, podem ser visualizadas as diversas partes que compõem o microcontrolador PIC 18F4550. Observe a ULA (Unidade Lógica Aritmética) ligada ao registrador W (*Work-register*). No canto superior esquerdo, abaixo da tabela de ponteiros, temos a memória de programa (32 KBytes), e saindo desse bloco um barramento de instruções com 16-bits. No lado superior direito é possível visualizar a memória de dados (2 KBytes), que possui um barramento de dados de 8-bits, conforme explicado na definição da arquitetura *Harvard*.

Do lado direito podemos visualizar as portas com todos os seus pinos de entrada/saída (I/O). Na parte inferior é possível visualizar os periféricos, tais como, a memória EEPROM, os temporizadores (Timer0, Timer1, Timer2, Timer3), o comparador interno, o módulo CCP (Capture, Compare e PWM), porta serial (EUSART), porta USB, conversor A/D de 10-bits.

Um pouco mais ao centro da Figura 2.4, estão representados os osciladores internos, o regulador de tensão da porta USB, *Power-up Timer* e *Watchdog Timer*. Na parte superior central, temos o contador de linha de programa (*Program Counter*) e a pilha (*Stack*) com 31 níveis.

Mais informações estão disponíveis no *datasheet* do microcontrolador PIC18F4550, disponível no site no DVD didático.

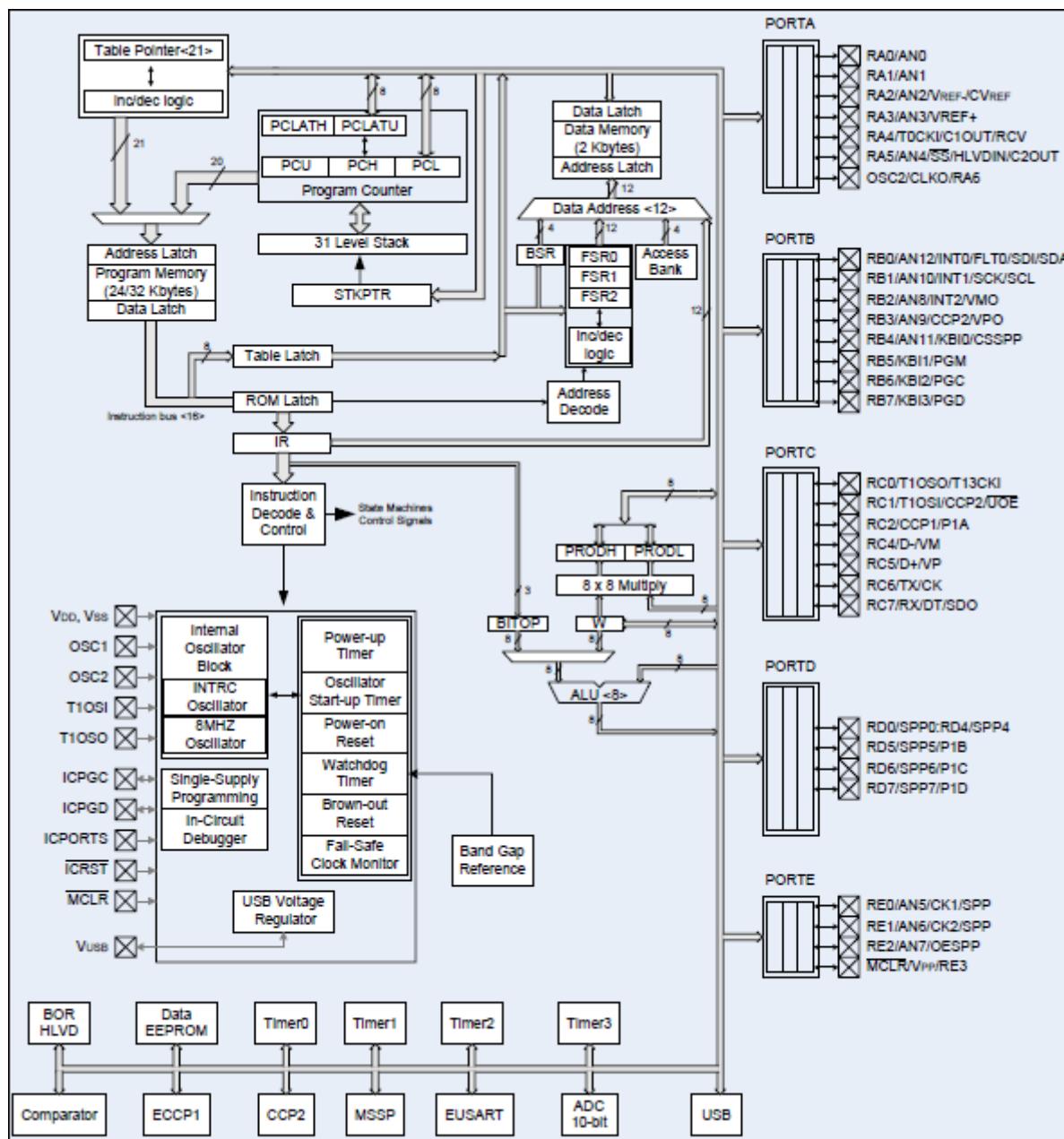


Figura 2.4 - Estrutura interna do microcontrolador PIC18F4550 (Fonte: datasheet PIC18F4550).

Como vimos, a mostra na Figura 2.4, por meio de diagrama de blocos, as principais partes de um sistema microprocessado padrão. Na Figura 2.5 é possível visualizar como estão distribuídas cada uma dessas partes na arquitetura interna do PIC18F4550.

Memória de Programa (Flash)

PIC18F4550

Memória de Dados (SRAM)

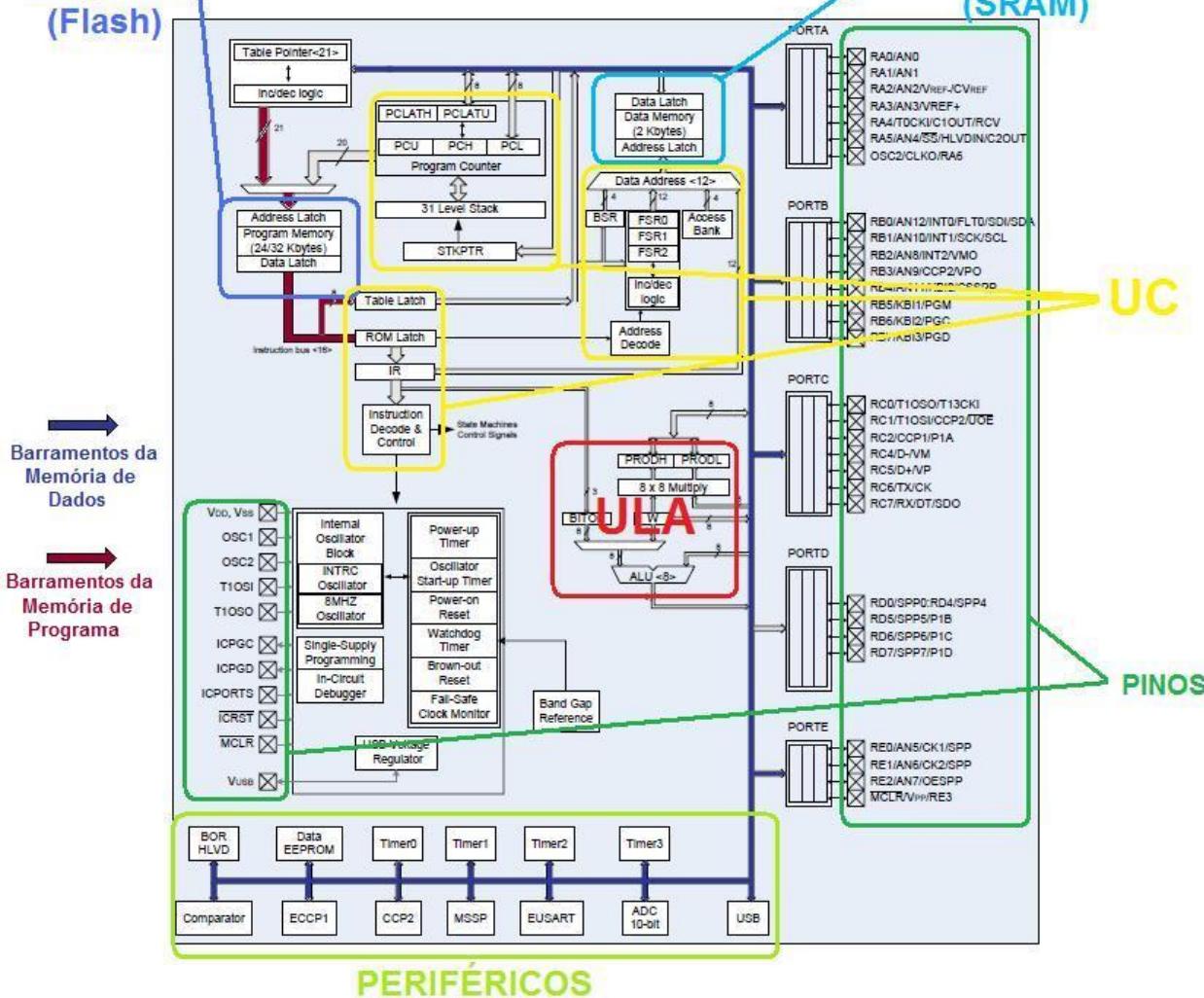


Figura 2.5 - Arquitetura do PIC18F4550 destacando as partes de um sistema microprocessado.

ORGANIZAÇÃO DA MEMÓRIA DO PIC18F4550

Segue abaixo detalhes da organização da memória do PIC18F4550:

Memória de programa: memória interna *flash* de 32.768 Bytes (32 KBytes).

- ⊕ Armazena instruções, constantes e dados;
- ⊕ Pode ser escrita/lida de acordo com o programador externo (*Off-board*), *In-circuit* ou durante a execução do programa através de ponteiros.

Memória RAM de dados: memória SRAM interna de 2048 Bytes (2 KBytes) e nela estão incluídos os registradores de função especial (SFR).

- ⊕ Armazena dados de forma temporária (Memória volátil) durante a execução do programa;
- ⊕ Pode ser escrita/lida em tempo de execução do programa através de diversas instruções.

Memória EEPROM de dados: memória não volátil de 256 Bytes.

- ⊕ Armazena dados que devem ser conservados na ausência de tensão de alimentação;

- ⊕ Pode ser escrita/lida em tempo de execução do programa através de registradores, observe que para serem lidos ou escritos os dados devem primeiro passar pela memória SRAM.

Memória de configuração: memória que contém bits de configuração (12 Bytes de memória *flash*) e os registradores de identificação (2 Bytes de memória de apenas leitura).

A memória de configuração se trata de um bloco de memória situado a partir da posição 300000H de memória de programa (muito além do espaço de memória do programa para o usuário).

Nesta memória de configuração incluem:

- ✓ Bits de configuração: contidos em 12 Bytes de memória *flash* permitindo a configuração de algumas opções do μC como:
 - ⊕ Opções de oscilador;
 - ⊕ Opções de reset;
 - ⊕ Opções de watchdog;
 - ⊕ Opções para depuração e programação do circuito;
 - ⊕ Opções da proteção contra escrita da memória do programa e de dados da memória EEPROM.

Estes bits são configurados geralmente durante a programação do μC, mas podem ser lidos e modificados durante a execução do programa.

- ✓ Registradores de identificação: trata-se de registradores situados nos endereços 3FFFFEH e 3FFFFFFH que contém informações do modelo e versão do dispositivo. Os registradores são apenas de leitura e não podem ser alterados pelo usuário.

ARQUITETURA HARVARD

O PIC18F4550 dispõe de barramentos diferentes para o acesso a memória de programa e a memória de dados (arquitetura Harvard). O barramento da memória de programa tem 21 linhas de endereçamento e 16/8 linhas de dados (16 linhas para instruções e 8 linhas para dados). O barramento da memória de dados tem 12 linhas de endereçamento e 8 linhas para dados.

Isto permite acessar simultaneamente a memória de programa e a memória de dados. Significa que ele pode executar uma instrução (o que geralmente exige a acesso aos dados de memória), enquanto busca a próxima instrução da memória de programa para ser executada em seguida (processo conhecido como **pipeline**) (Figura 2.6).

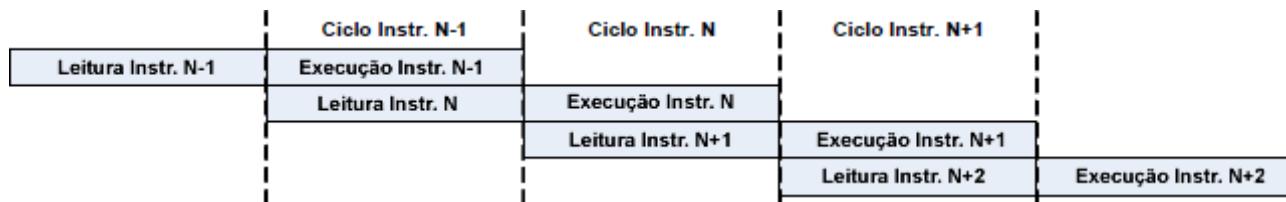


Figura 2.6 - Processo Pipeline.

Portanto, a execução completa de 1 instrução (Leitura da instrução + execução) se faz em um ciclo de instrução ($4 T_{osc}$ ¹). EXCEÇÃO: as instruções que modificam o conteúdo do PC (Program Counter) requerem 2 ciclos de instruções, algumas conhecidas como instruções de salto. Ex: Chamadas de função.

MEMÓRIA DE PROGRAMA (FLASH)

O µC PIC18F4550 dispõe de uma memória de programa de 32.768 Bytes (0000H-7FFFH). As instruções ocupam 2 Bytes (exceto de instruções em assembly: CALL, MOVFF, GOTO e LSFR que ocupam 4). Assim, a memória de programa pode armazenar até 16.384 instruções.

A operação de leitura da memória na posição acima de 7FFFH resulta com '0' (equivalente à instrução NOP).

Podemos ainda citar endereços especiais na memória de programa. O endereço do **vetor reset** é 0000H, o do **vetor de interrupções de alta prioridade** é 0008H e o do **vetor de interrupções de baixa prioridade** é 0018H. (Figura 2.7)

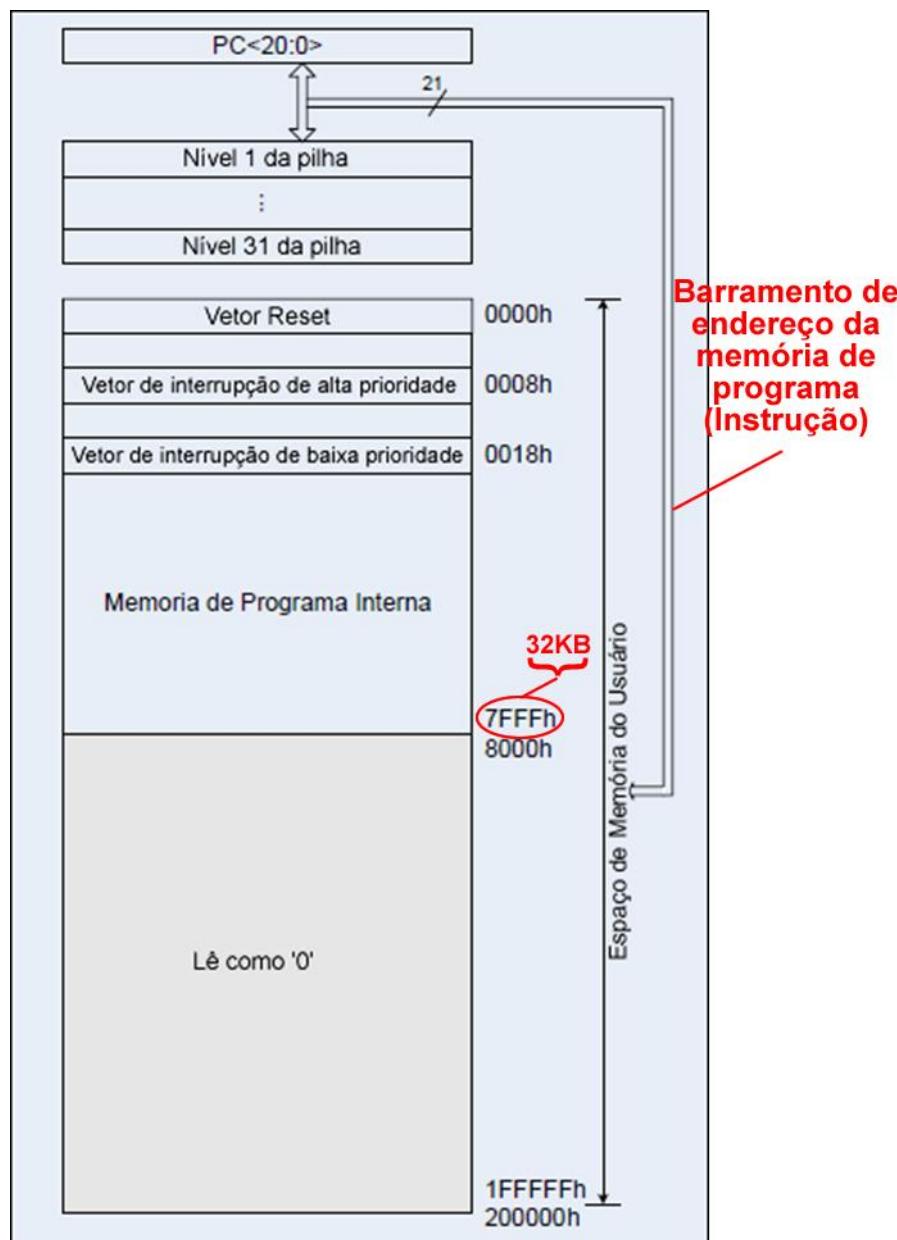


Figura 2.7 - Detalhes da memória FLASH e da Pilha.

O vetor Reset trata-se do primeiro endereço de memória de programa que será executado quando o PIC começar a rodar (após a alimentação ou um reset). As rotinas de interrupção serão armazenadas na área de programação, juntamente com todo o resto do programa. No entanto, existe um endereço que é reservado para o início do tratamento de todas as interrupções. A pilha é um local, totalmente separado da memória de programação, em que serão armazenados os endereços de retorno quando utilizarmos instruções de chamadas de rotinas (SOUZA, 2005).

MEMÓRIA RAM DE DADOS

A memória de dados é utilizada para guardar todas as variáveis e registradores utilizados pelo programa. Essa memória armazena dados de 8 bits e é volátil, ou seja, quando o PIC é desligado, ela é automaticamente perdida. Ela é dividida em 16 bancos de memória, dos quais apenas 8 são utilizados pelo PIC18F4550.

O PIC18F4550 tem 2048 Bytes de memória RAM (Volátil) divididos em 8 bancos de 256 Bytes. Dos quais, 160 Bytes são reservados aos registradores de funções especiais (SFR) localizado na parte mais alta do banco 15 da memória (endereço F60h até FFFh). Uma informação importante a ser destacada é a diferença de um barramento de endereço do PIC poder endereçar uma determinada quantidade de memória (esse valor está associado ao tamanho do barramento de endereço) e ter o que endereçar (tamanho real da memória de dados). Por exemplo: no PIC18F4550 é possível endereçar até 4 KBytes, pois o barramento de endereços tem 12 bits, $2^{12} = 4096$ Bytes = 4 KBytes, porém só temos 2 KBytes de memória SRAM física disponível no hardware.

A memória de dados contém os registradores especiais de função (SFRs) e os registradores de propósito geral (GPRs). Os SFRs já foram citados na seção 1.3.4, os GPRs são usados para armazenamento temporário de dados e resultados de operações do programa do usuário. Qualquer leitura em um local não implementado é lido como 0 (zero). Quando o modo USB é habilitado, os bancos 4, 5, 6 e 7 são mapeados para bufferização da porta USB. Quando o modo USB é desabilitado, os GPRs nesse banco podem ser usados como qualquer outro GPR no espaço da memória de dados. (Figura 2.8)

Grandes áreas da memória de dados requerem um esquema de endereçamento eficiente para fazer o rápido acesso para qualquer dos endereços possíveis. Idealmente, isso significa que todos os endereços possíveis não precisam ser fornecidos para cada operação de leitura ou escrita. Por isso que no PIC18F4550 existe o esquema de Bancos de Memória, conforme citado acima. O acesso a um Byte da memória SRAM (memória de dados do PIC) é feito pelo Registrador de Seleção do Banco (BSR). Esse SFR ocupa os 4 mais significativos bits da localização do endereço, sendo que a instrução ocupa os 8 menos significativos bits (4-bit + 8-bit = 12-bit de endereçamento).

Existe também uma forma de acesso rápido para as 96 posições da parte inferior do banco e os 160 Bytes mais altos do banco 15, os SFRs. (Figura 2.8)

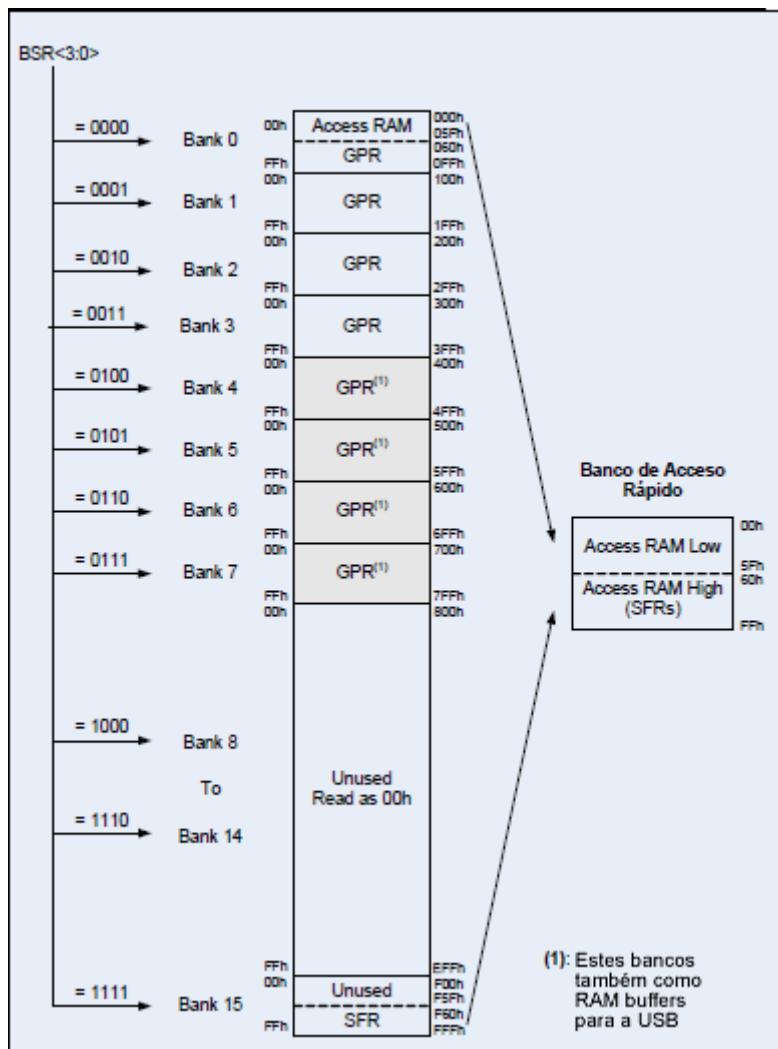


Figura 2.8 - Mapa da memória de dados.

O endereço FFFh equivale a 4095 Bytes (capacidade máxima de endereçamento com 12-bit).

REGISTRADORES DE FUNÇÕES ESPECIAIS NO PIC18F4550

Os registradores de funções especiais são usados pela CPU e pelos módulos periféricos para controle de operações desejadas do dispositivo, servem para guardar a configuração e o estado atual da máquina. Os SFRs (o PIC 18F4550 possui cerca de 140 SFRs) são os registradores através dos quais monitoramos/controlamos o funcionamento da CPU e das unidades funcionais do µC.

O conjunto dos SFRs classifica-se em:

- ✓ SFRs associados a funcionalidades do µC:
 - CPU: WREG, STATUS, BSR, etc...
 - Interrupções: INTCON, PIE1 PIR1 IPR1, etc...
 - Reset: RCON.
- ✓ SFRs relativos a operações das funcionalidades dos periféricos:
 - Timers: T0CON, TMR1H, TMR1L, T1CON, etc...
 - Conversor A/D: ADRESH, ADRESL, ADCON0, ADCON1, etc...
 - USART: TXREG, TXSTA, RCSTA, etc...
 - CCP: CCP1H, CCP1L, CCP1CON, etc...
 - MSSP: SSPSTAT, SSPDATA, SSPCFG, etc...
 - Portas de I/O: TRISA, PORTA, TRISB, PORTB, etc...

REGISTRADOR STATUS

O registrador STATUS é utilizado para armazenamento de flags (sinalizadores) matemáticos e de estado da CPU, esses flags sinalizam o status aritmético da ULA. (Figura 2.9)

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC	C
bit 7						bit 0	

Figura 2.9 - Registrador Status.

Ele possui 8 bits, sendo 5 importantes e descritos abaixo:

- ✓ **N:** esse bit indica quando um resultado é negativo. Bit usado para operações com sinal;
 - **N='0':** resultado da última operação foi positivo;
 - **N='1':** resultado da última operação foi negativo;
 - ✓ **OV:** bit de overflow. Bit utilizado para operações com sinal. Indica se houve overflow do 7º bit;
 - **OV='0':** não houve overflow na operação aritmética;
 - **OV='1':** houve overflow na operação aritmética;
 - ✓ **Z:** bit de zero. Indica se o resultado de uma operação foi zero.
 - **Z='0':** o resultado de uma operação aritmética ou lógica foi diferente de '0';
 - **Z='1':** o resultado de uma operação aritmética ou lógica foi igual de '0';
 - ✓ **DC:** bit de transporte de bit entre os nibbles. Se houve transporte de bits do 4º para o 5º bit.
 - **DC='0':** não houve transporte do 4º para o 5º bit;
 - **DC='1':** houve transporte do 4º para o 5º bit;
 - ✓ **C:** bit de transporte ou estouro. Se a operação ultrapassou os 8 bits de dados.
 - **C='0':** não houve transporte ou estouro;
 - **C='1':** houve transporte ou estouro;

Legenda:

L = Bit de Leitura **E** = Bit de Escrita **U** = Bit não implementado, lido como "0"

1 = Bit é setado (nível lógico 5V) 0 = Bit é zerado (nível lógico 0V)

$x \equiv$ Valor do bit é desconhecido

Para informações dos outros registradores, consulte o *datasheet* do PIC18F4550 disponível no site no DVD didático.

MEMÓRIA EEPROM DE DADOS

A EEPROM é uma memória não-volátil separada das memórias de dados RAM e de programa. Que é usada para armazenar dados do programa. No PIC18F4550, a memória EEPROM de dados possui 256 Bytes. Apenas quatro registradores são usados para leitura e escrita de dados na EEPROM. São eles: EECON1, EECON2, EEDATA, EEADR.

Esta memória permite até 1.000.000 de ciclos de leitura e escrita. Pode-se ler/escrever de forma individual cada uma das 256 posições de memória.

2.2 KIT DE DESENVOLVIMENTO COM PIC18F4550

O KIT de desenvolvimento baseado no PIC18F4550, fabricado pela PICMinas, pode ser dividido em quatro blocos: circuitos atuadores, chaves/teclas, circuitos sensores e displays.

2.2.1. CIRCUITOS ATUADORES

Os circuitos atuadores são compostos por: 3 (três) LEDs (Vermelho, Verde e Amarelo), Relé de 5V e o Buzzer (Sirene). A Figura 2.10, Figura 2.11 e Figura 2.12 mostram as ligações de cada um desses atuadores no PIC18F4550.

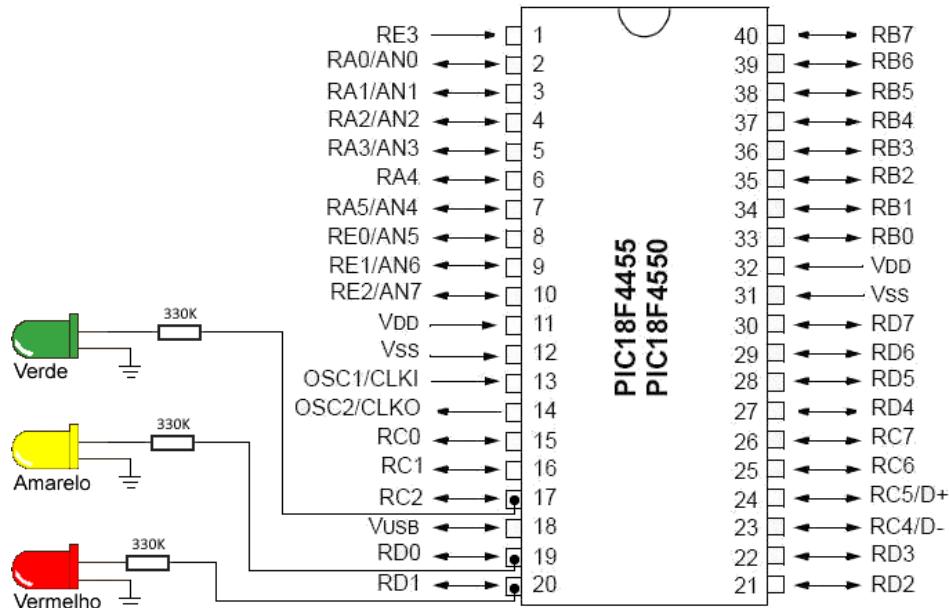


Figura 2.10 - Ligação dos led's no PIC.

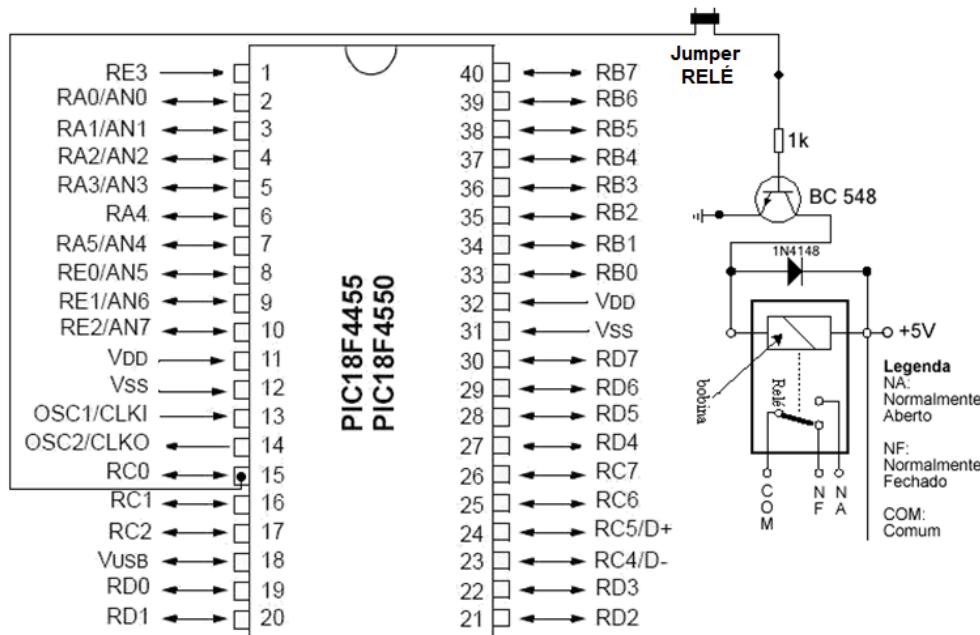


Figura 2.11 - Ligação do Relé no PIC.

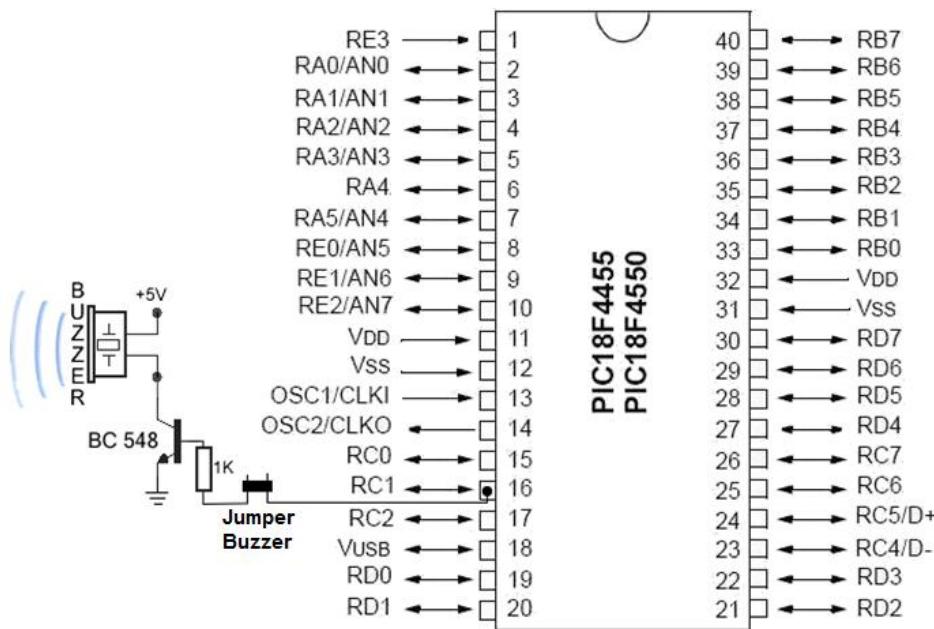


Figura 2.12 - Ligação do Buzzer (Sirene) no PIC.

2.2.2. CHAVES/TECLAS

O KIT possui quatro chaves téteis (*Push button*): **Botão 1**, **Botão 2**, **RESET** e **BOOT**. A chave RESET é de uso específico e não pode ser utilizada pelos programas desenvolvidos pelo usuário. Para se utilizar a chave tátil BOOT é necessário que o **Jumper “BOOT/RB4”** esteja selecionado na “**posição BOOT**”, como mostrado na Figura 2.13. Caso contrário, o botão não possui nenhuma funcionalidade e o pino RB4 (onde a chave BOOT está conectada) ficará disponível no conector do Display de 7-Segmentos, tanto para acionamento do próprio display, quanto para a utilização de dispositivos externos ao KIT.

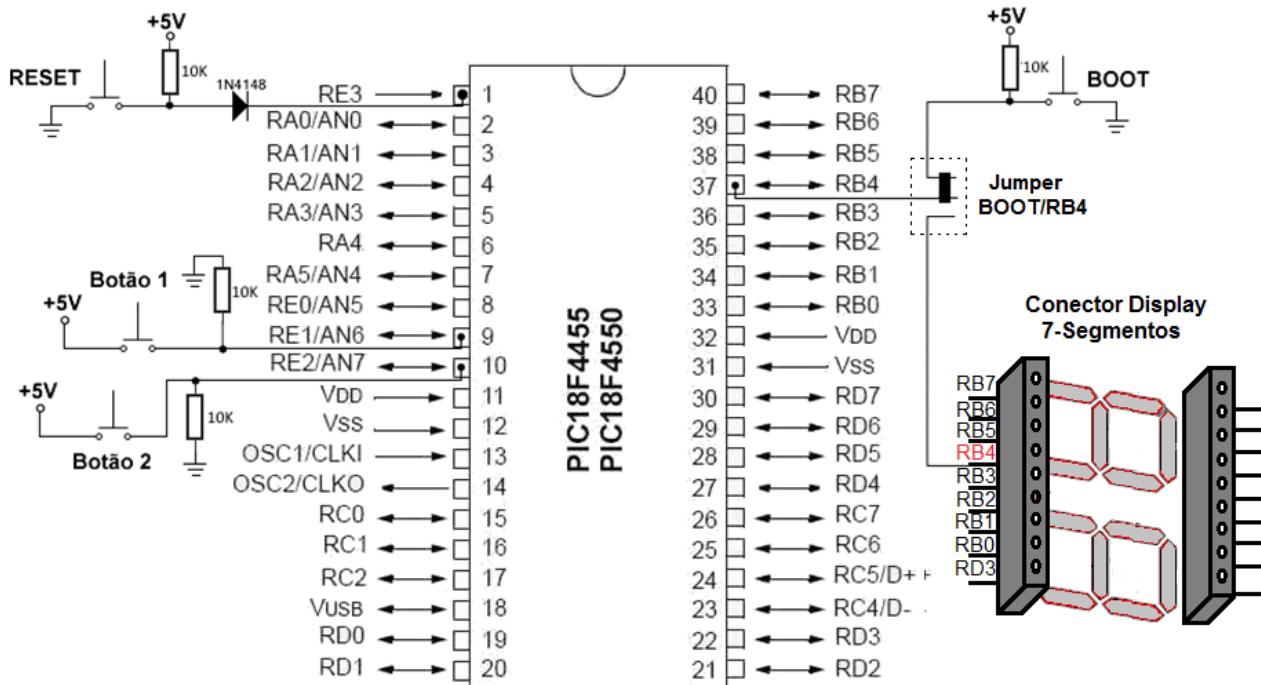


Figura 2.13 - Ligação da Chave Push Button (Chaves Táteis) no PIC.

A Figura 2.14 mostra as ligações das chaves *Dip-Switch* no KIT. Observe a existência de um *Jumper* chamado “CHAVES”. Com ele é possível habilitar ou desabilitar os *pull-ups* presentes em cada uma das chaves. Os mesmos pinos que estão conectados às chaves (RA1/AN1, RA2/AN2, RA3/AN3 e RA4) também são disponibilizados em um conector de expansão. Os *pull-ups* devem ser desabilitados sempre que o usuário desejar acionar outros dispositivos externos por meio do conector de expansão.

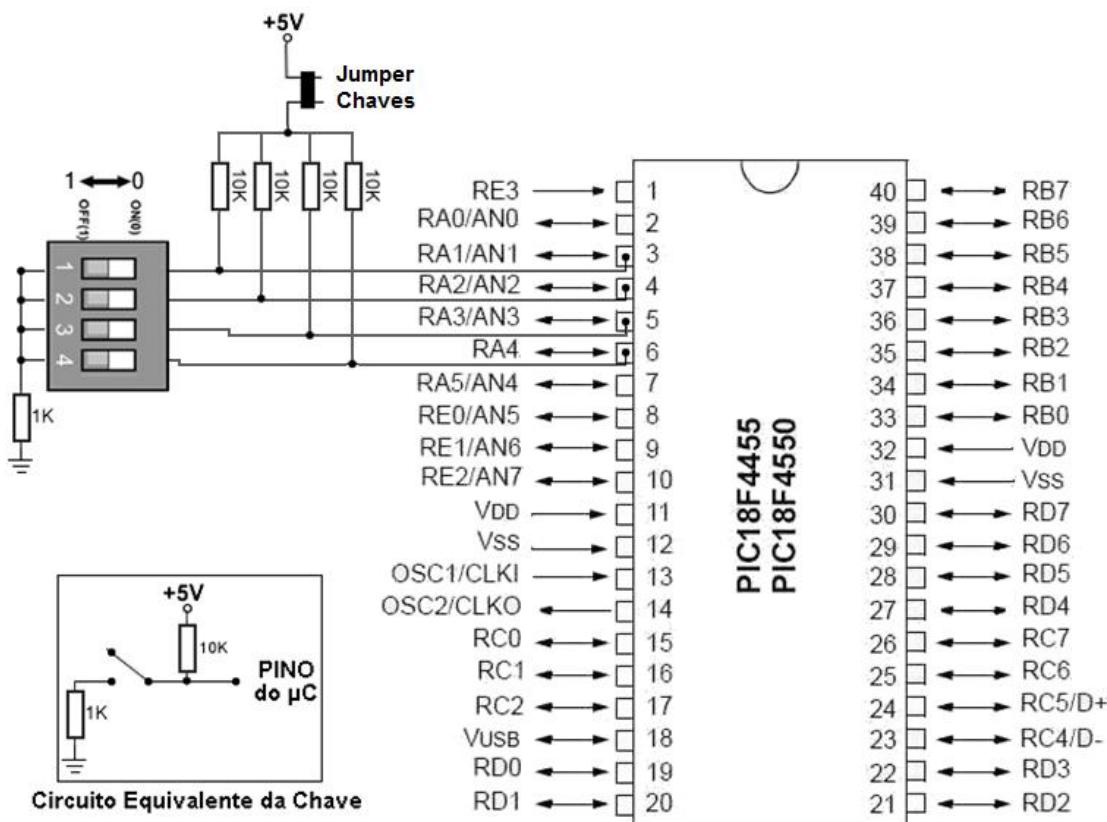


Figura 2.14 - Ligação da Chave Dip switch no PIC.

2.2.3. CIRCUITOS SENsoRES

Os circuitos sensores são compostos por: um LDR, um sensor de temperatura e um potenciômetro. A Figura 2.15, Figura 2.16 e Figura 2.17 mostram as ligações de cada um desses sensores no KIT.

O LDR (*Light Dependent Resistor*) é um transdutor resistivo sensível as variações da intensidade luminosa. A sua resistência é inversamente proporcional à intensidade de luz a qual está submetido, ou seja, quanto maior a intensidade de luz aplicada sobre ele, menor será o valor de sua resistência elétrica. Desta forma, fica fácil verificar pela Figura 2.15 que à medida que a resistência do LDR diminui a tensão elétrica aplicada na entrada analógica AN5 também irá diminuir. Conclui-se assim, que quanto maior a intensidade luminosa aplicada ao LDR, menor será a tensão elétrica presente na entrada analógica AN5.

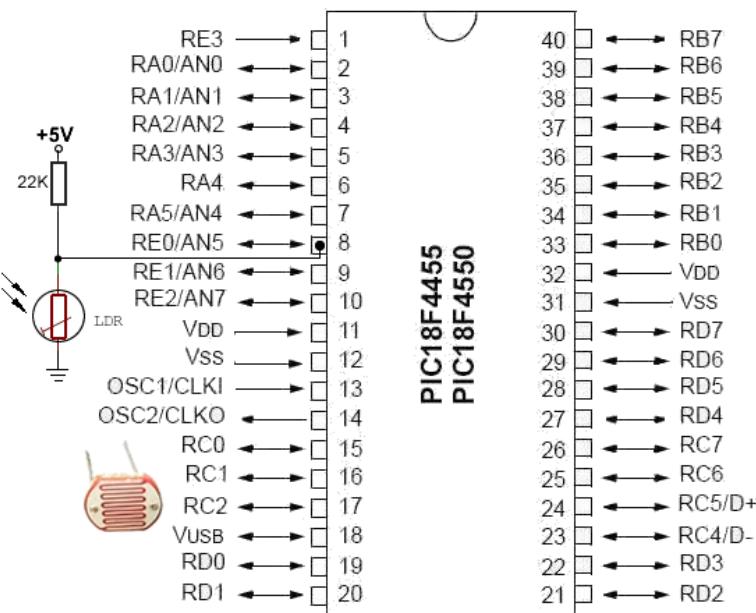


Figura 2.15 - Ligação do LDR no PIC.

O sensor de temperatura utilizado é o MCP9700A, que está conectado ao PIC como mostrado na Figura 2.16. Este sensor está acoplado ao KIT por meio de um conector, podendo assim ser removido. Isso permite a conexão de outros sensores à entrada analógica AN0, desde que estes possuam características elétricas compatíveis com as entradas analógicas do PIC (ver datasheet – Conversor Analógico Digital).

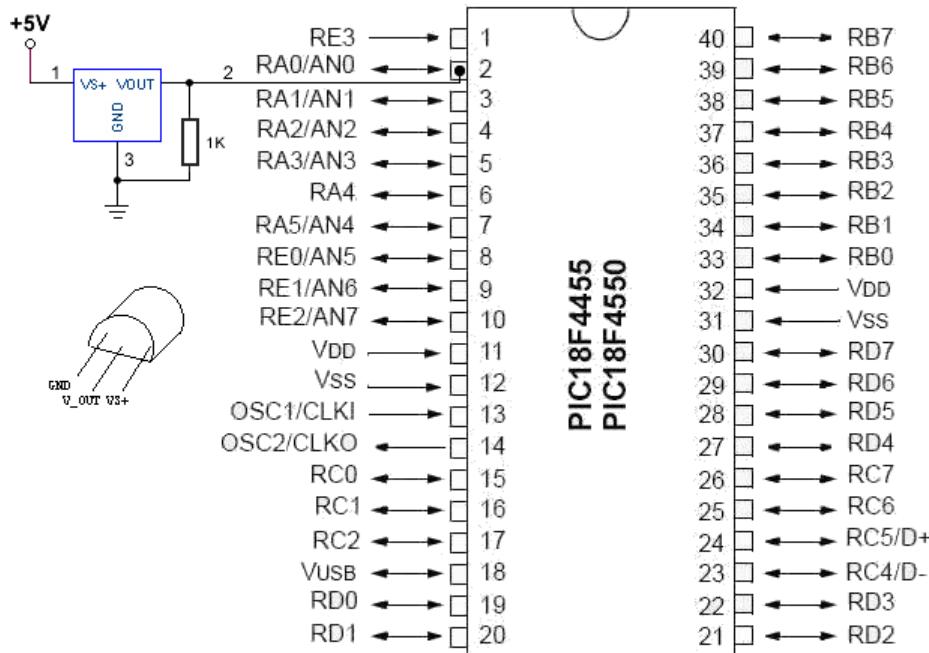


Figura 2.16 - Ligação do sensor de temperatura no PIC.

O potenciômetro funciona como um circuito divisor de tensão. A tensão elétrica aplicada ao pino AN4 do PIC18F4550 varia de acordo com a posição de seu parafuso. (Figura 2.17)

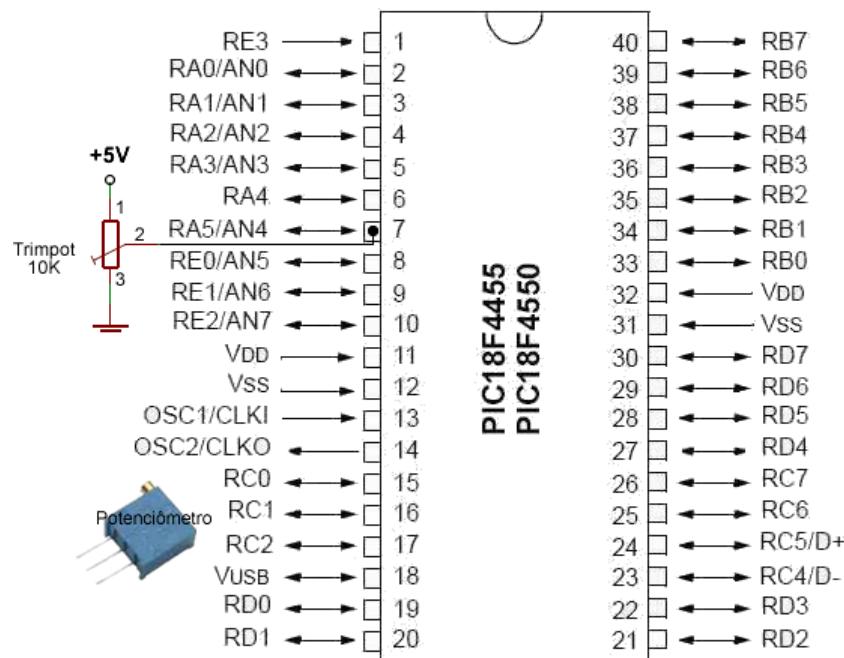


Figura 2.17 - Ligação do potenciômetro no PIC.

2.2.4. DISPLAYS

O KIT possui dois conectores para displays: um conector para display duplo de 7-segmentos (anodo comum) e um conector compatível com a maioria dos displays de LCD Alfanuméricos disponíveis no mercado (displays 8x1, 8x2, 16x1, 16x2, 16x4, 32x2, etc.). As Figura 2.18 e Figura 2.19 mostram as ligações de cada um desses conectores. Observe que para habilitar o funcionamento do ponto entre os dois dígitos, o Jumper “BOOT/RB4” deve estar selecionado na posição “RB4”, como mostrado na Figura 2.18.

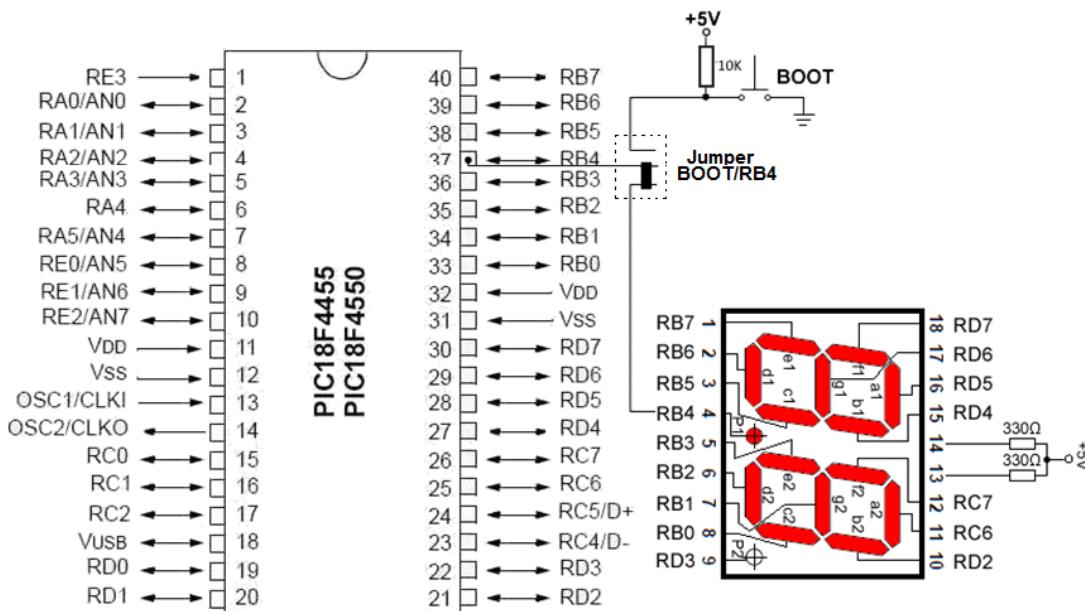


Figura 2.18 - Ligação do display duplo de 7 segmentos no PIC.

Observação: o display de 7 (sete) segmentos é do tipo Anodo Comum, isso quer dizer que seus LED's ($a_1, b_1, c_1, d_1 \dots$) acenderão quando as portas conectadas a eles (RB0, RB1, RB2, ...) estiverem em lógica binária “0” (zero) ou “0” (zero) Volts.

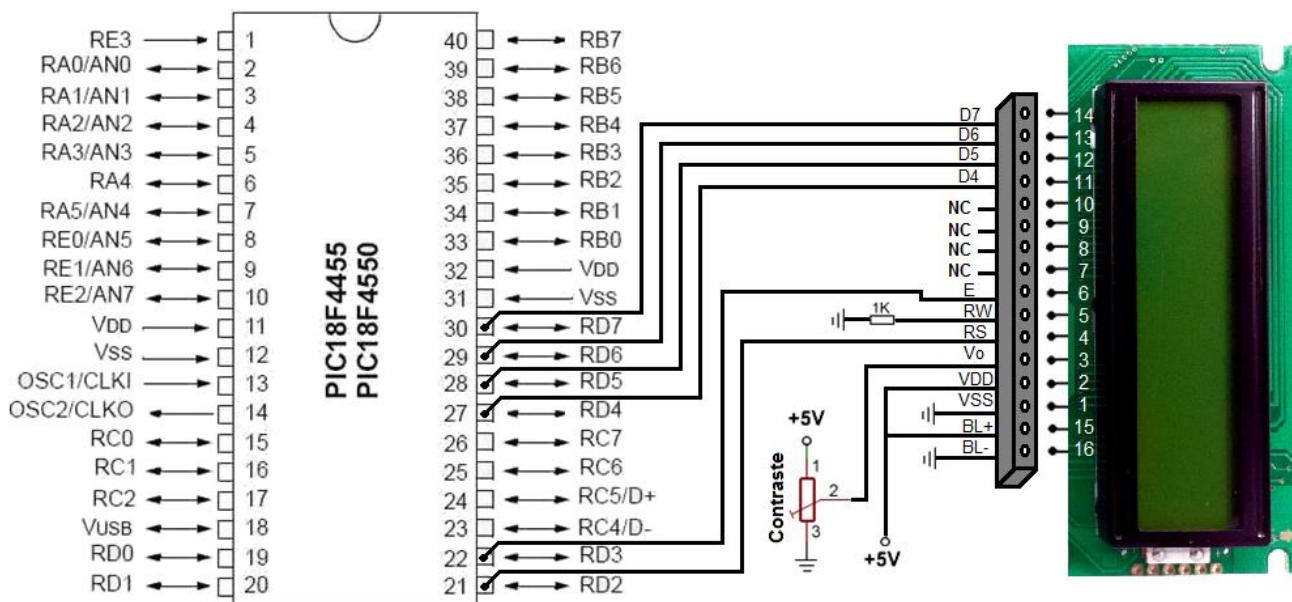


Figura 2.19 - Ligação do display de LCD no PIC.

Abaixo segue a Tabela 4 com as informações úteis do LCD:

Nome no LCD	Pino do LCD	Pino do PIC18F4550	Nome do Pino no PIC18F4550	Descrição
VSS	1	GND		Pino de Terra do LCD
VDD	2	5V		Pino de Alimentação
VO	3	-	Não conectado	Pino de ajuste do contraste ligado ao potenciômetro.
RS	4	P21	RD2	Pino de Seleção de envio de Comandos ou Dados .
R/W	5	GND		Pino de Seleção Leitura ou Escrita de dados. No KIT PICMINAS está habilitada apenas escrita.
E	6	P22	RD3	Pino de habilitação de recepção de sinal do LCD
D0	7	-	Não conectado	Pino para envio de Dados (Não utilizado no KIT)
D1	8	-	Não conectado	Pino para envio de Dados (Não utilizado no KIT)
D2	9	-	Não conectado	Pino para envio de Dados (Não utilizado no KIT)
D3	10	-	Não conectado	Pino para envio de Dados (Não utilizado no KIT)
D4	11	P34	RD4	Pino para envio de Dados
D5	12	P25	RD5	Pino para envio de Dados
D6	13	P33	RD6	Pino para envio de Dados
D7	14	P26	RD7	Pino para envio de Dados
BL+	15	5V		Polo positivo da alimentação do <i>Back Light</i> do LCD
BL-	16	GND		Polo negativo da alimentação do <i>Back Light</i> do LCD

Tabela 4 - Informações de ligação do LCD.

2.2.5. CIRCUITOS DE GRAVAÇÃO IN-CIRCUIT (ICSP)

O KIT PICMINAS possui um conector padrão usado para gravação *in-circuit*, o ICSP (*In-Circuito Serial Programming*). Desta forma, o kit pode ser conectado às gravadoras disponíveis no mercado que possuam esse tipo de conexão (ICD2, PICKIT2, dentre outras), tanto para carregar programas, quanto para debug e emulação de firmwares, veja a Figura 2.20.

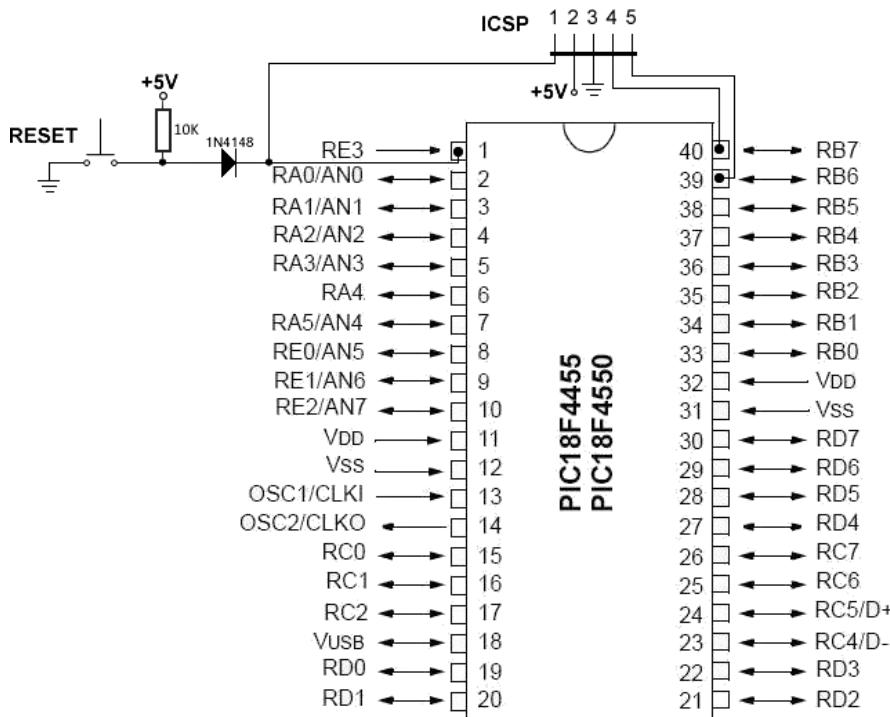


Figura 2.20 – Conector padrão MICROCHIP ICSP.

RECOMENDAÇÕES PARA USO DO CONECTOR ICSP DO KIT PICMINAS:

- Desconectar qualquer periférico ligado nos pinos RB6 e RB7, como por exemplo: o display duplo de 7 segmentos e o LCD.
- Usar a alimentação da placa do KIT PICMINAS pela USB ou por uma fonte externa. Não é recomendada a utilização da alimentação da própria gravadora.

2.2.6. Jumpers do KIT PICMINAS

Jumper é uma ligação móvel entre dois pontos de um circuito eletrônico. É, geralmente, uma pequena peça plástica que contém um metal em seu interior, responsável pela condução de eletricidade entre dois ou mais pontos de um circuito. São responsáveis por desviar o fluxo elétrico permitindo configurações por meio físico do hardware em questão. O KIT PICMINAS dispõe de diversos *jumpers* que permitem habilitar e desabilitar dispositivos, como por exemplo, buzina e relé, ou ainda, possibilitam ao usuário escolher entre duas opções de configuração. Os *Jumpers* são identificados por meio de nomes escritos na placa do KIT PICMINAS. Abaixo segue uma lista com todos os nomes e a função de cada um deles:

- ✚ *Jumper CHAVES*: quando removido desabilita os *pull-ups* das 4 chaves do *Dip-Switch*.
- ✚ *Jumper BUZZER*: quando removido desabilita o funcionamento do Buzzer.
- ✚ *Jumper RELÉ*: quando removido desabilita o funcionamento do Relé.
- ✚ *Jumper BOOT | RB4*: com este *jumper* é possível escolher aonde o pino RB4 será conectado. Quando selecionado na posição “BOOT”, o pino RB4 estará conectado à chave tátil BOOT. Já se selecionado na posição “RB4”, a chave BOOT para de funcionar e o pino RB4 fica disponível no conector do display de 7-Segmentos.
- ✚ *Jumper VUSB | VBAT*: permite ao usuário escolher qual será a fonte de alimentação do KIT, se proveniente do cabo USB (posição “VUSB”) ou da alimentação externa (posição “VBAT”).

A Figura 2. 21 mostra como os cinco *Jumpers* estão conectados aos respectivos circuitos do KIT PICMinas.

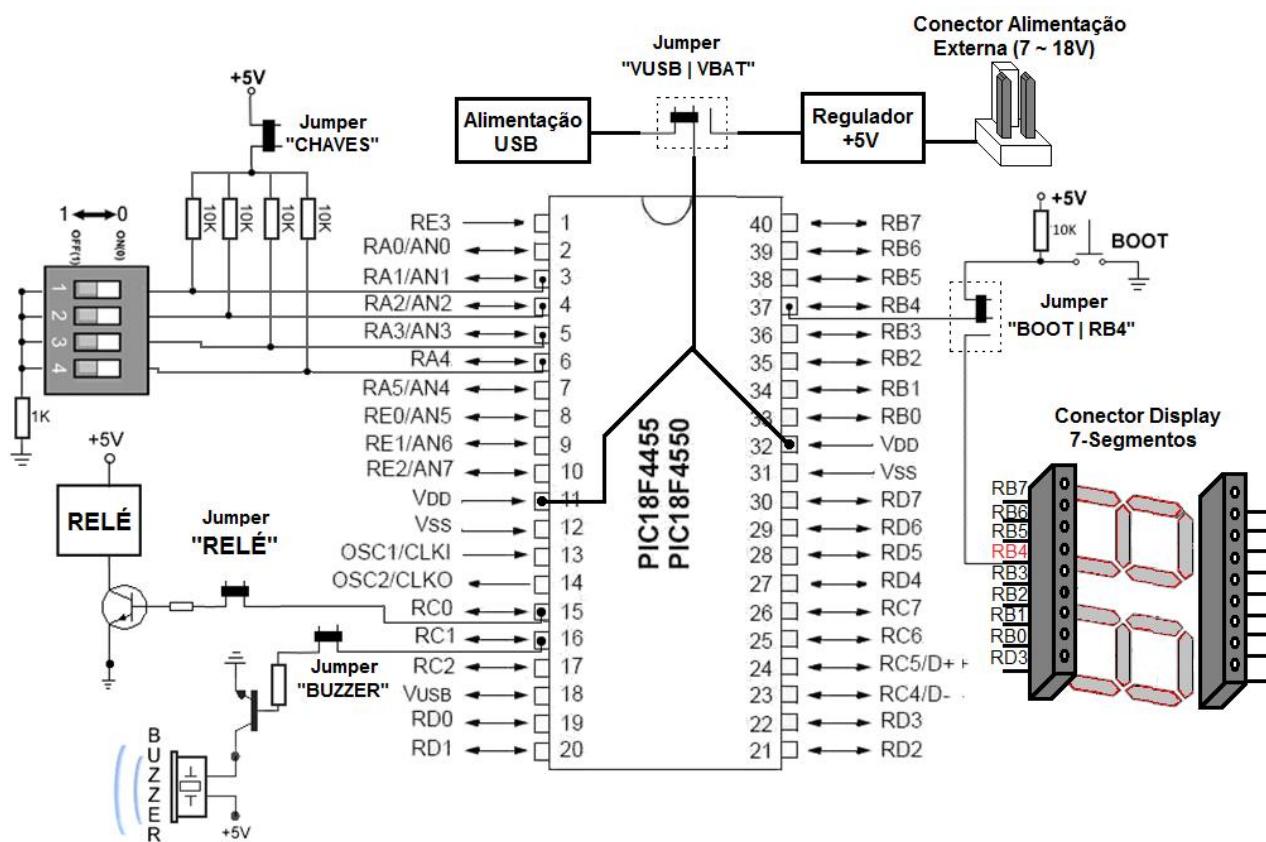


Figura 2. 21 - Jumpers do KIT PICMinas.

2.3 COMPONENTES DO KIT DIDÁTICO PIC18

A Figura 2.22 abaixo detalha os componentes principais do kit de desenvolvimento da PICMINAS.

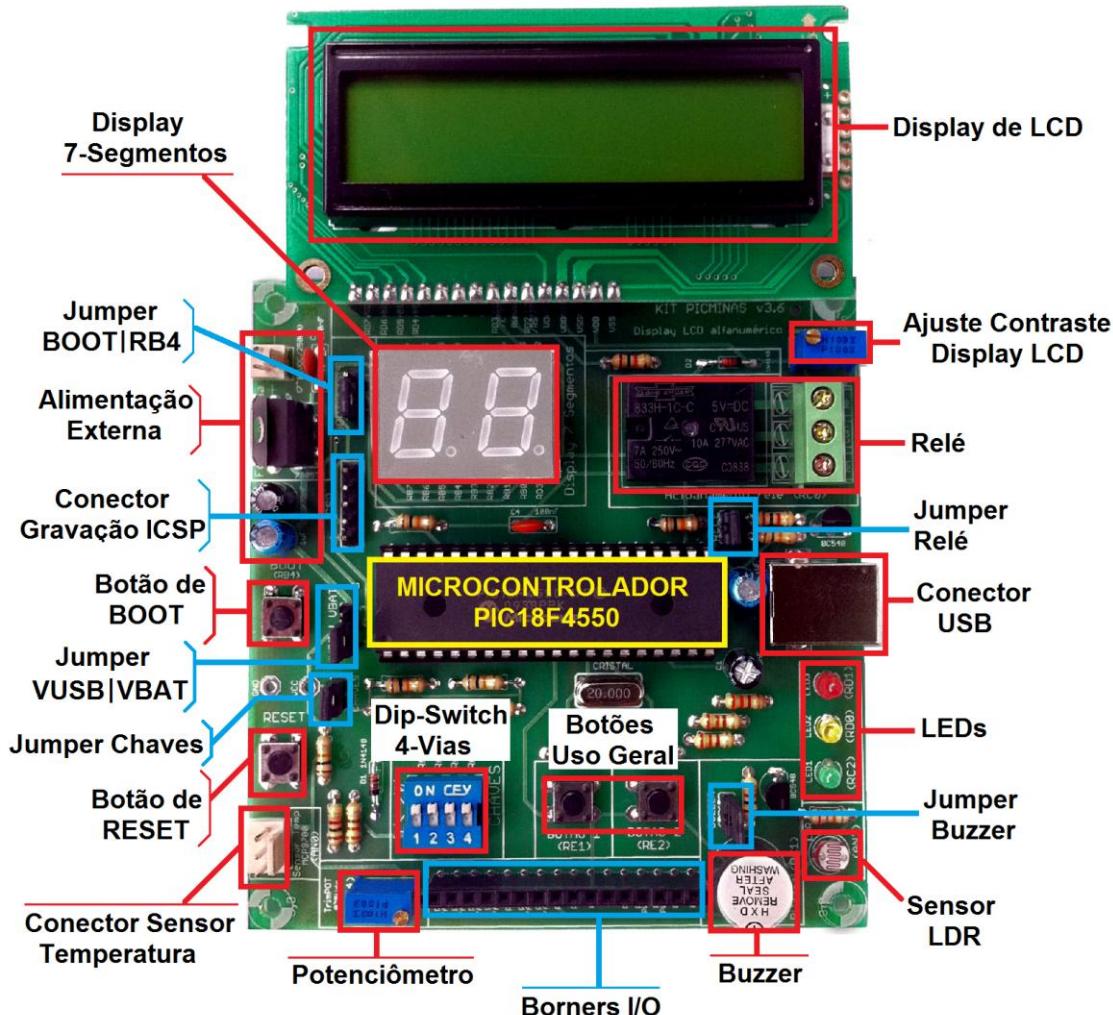


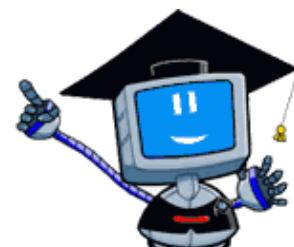
Figura 2.22 - Componentes principais do kit de desenvolvimento.

Referências:

- 1- MARTINS, H. R. **Sistema Para O Estudo Do Limiar De Percepção De Corrente Elétrica Com Forma De Onda Arbitrária.**, 2008. 120 p. Dissertação (Mestrado em Engenharia) - Universidade Federal de Minas Gerais, PPGEE, Belo Horizonte.
- 2- PEREIRA, Fabio. **Microcontroladores PIC Programação em C.** 7º ed. São Paulo: Editora Érica Ltda, 2008.
- 3- SOUZA, David José. **Desbravando o PIC – Ampliado e Atualizado para PIC16F628A.** 5º ed. São Paulo: Editora Érica Ltda, 2005.

Neste capítulo, falamos sobre:

- A arquitetura do microcontrolador PIC18F4550;
- Detalhes sobre o kit de desenvolvimento da PICMINAS.



Capítulo 3 – Ferramentas de Desenvolvimento

Antes de iniciarmos o desenvolvimento de projetos com sistemas microcontrolados, faz-se necessária a instalação de todas as ferramentas (softwares) utilizadas em sua programação. Todos os softwares necessários possuem versões gratuitas disponíveis no site da Microchip ou em nosso DVD Didático.

O DVD Didático traz opção vários vídeos explicativos que iram guiá-lo por todo o processo de instalação desses programas. Esses vídeos substituem em parte as informações contidas nesse capítulo.

Este capítulo trás algumas dicas para orientá-lo no manuseio do DVD Didático referente ao processo de instalação das ferramentas de desenvolvimento.

Para desenvolver programas em linguagem C nos microcontroladores PIC, são utilizados os seguintes softwares: compilador C18 (para programar microcontroladores da família PIC18), compilador C32 (para programar microcontroladores da família PIC32) e o ambiente de desenvolvimento MPLAB. Para a gravação, no microcontrolador, dos códigos desenvolvidos via USB, utilizando o Bootloader, sem a necessidade de um gravador externo, você pode utilizar o DVD Didático, no ícone “Gravar PIC”.

É importante seguir a ordem de instalação proposta neste capítulo, para garantir o bom funcionamento de todos os programas.

3.1 AMBIENTE DE DESENVOLVIMENTO – MPLAB

O MPLAB é um pacote de programas fornecido gratuitamente pela Microchip (www.microchip.com), fabricante dos microcontroladores PIC, para gerar os códigos de programação que serão convertidos em linguagem de máquina (tipos “.hex” e “.cof”) para serem gravados na memória dos microcontroladores. O MPLAB integra num único ambiente o gerenciador de projetos, o editor de programa fonte, o compilador, o simulador, o emulador e quando conectado às ferramentas da Microchip também integra o gravador do PIC, facilitando assim o trabalho do programador. É um programa para PC, que roda sobre uma plataforma Windows.

O Programa fonte, ou simplesmente fonte do programa é uma sequência de texto, escrita numa linguagem de programação que será convertida em código de máquina para ser gravado na memória de programa do PIC. O Compilador é o programa que converte o código fonte, desenvolvido pelo usuário em alguma linguagem de programação, em código de máquina. O Simulador é o programa que, como o próprio nome diz, simula o funcionamento do µC (CPU e periféricos), tornando possível a verificação de programas fonte em desenvolvimento, sem a necessidade de um hardware.

A linguagem de programação utilizada pelo MPLAB para gerar os códigos é o *Assembly*, mas podem ser instalados outros compiladores para trabalharem em conjunto com o MPLAB, como por exemplo, o PICC Lite da Hi-Tech, PCWH CSS, C18, C32, entre outros. Alguns desses compiladores utilizam a linguagem C como interface com os usuários, porém cada compilador possui suas particularidades na forma de configurar os registradores e acessar os periféricos do microcontrolador.

O MPLAB trabalha com projetos, que são constituídos por arquivos gerados pelo usuário e/ou arquivos de bibliotecas que agrupam funções desenvolvidas pela própria Microchip.

Nesta seção, serão mostrados os passos envolvidos na instalação do MPLAB IDE a partir do DVD Didático. Este DVD possui a versão mais atual da época de sua gravação. Você pode obter a versão mais atual, gratuitamente, no site da Microchip.

Procedimento

- 1) Para instalar o MPLAB utilizando o DVD, vá à aba “Instaladores” e em MPLAB IDE clique no botão “Instalar” (ver Figura 3. 1). O instalador abrirá automaticamente. Clique em “Next” (ver Figura 3.2).

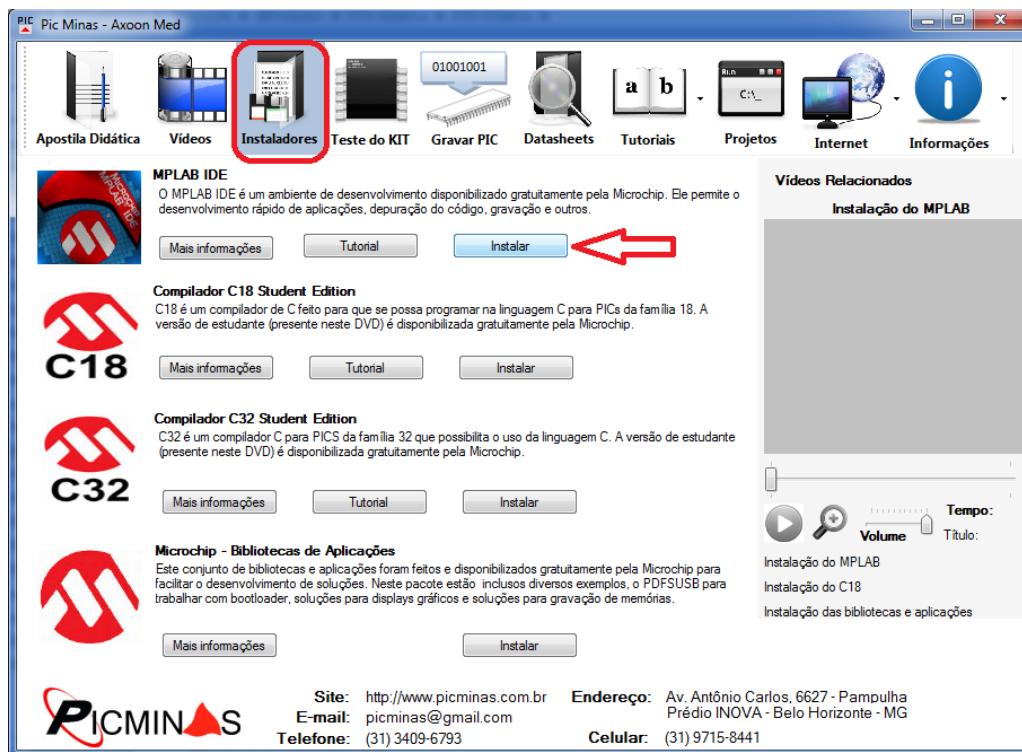


Figura 3. 1 - Ícone Instaladores do DVD Didático. Instalação do MPLAB.



Figura 3.2 - Tela de boas vindas do instalador do MPLAB.

- 2) A tela seguinte contém o termo de licença do MPLAB. Leia atentamente e, caso você concorde, clique em “I accept the terms of the license agreement”, em seguida, clique em “Next” (ver Figura 3.3).

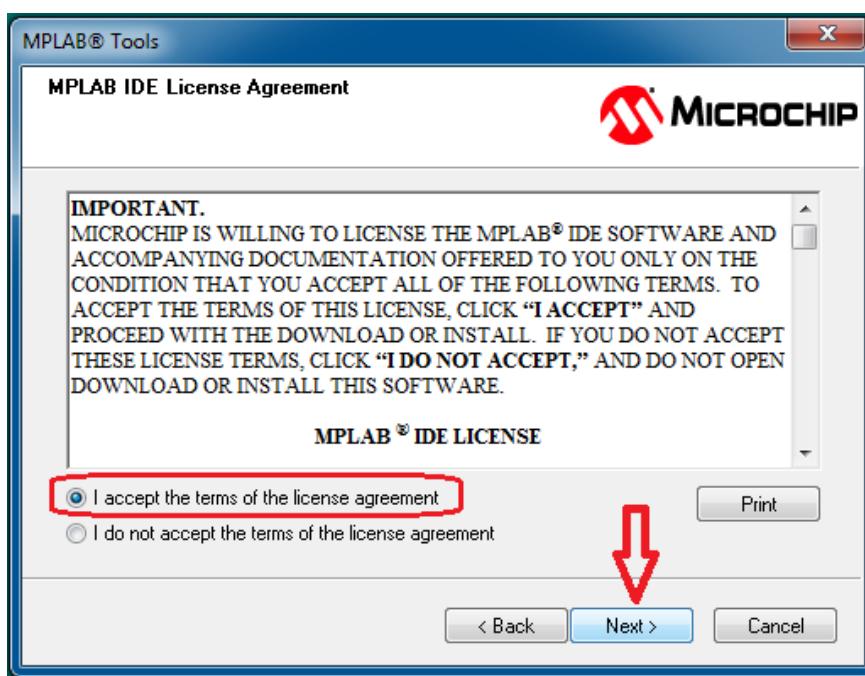


Figura 3.3 - Termo de licença.

- 3) Nesta etapa o instalador nos dará a opção de instalar o programa completo, com todos os seus recursos, ou realizar uma instalação customizada. Utilizaremos todos os recursos do MPLAB. Para isso, clique em “complete” e depois em “Next” (ver Figura 3.4).

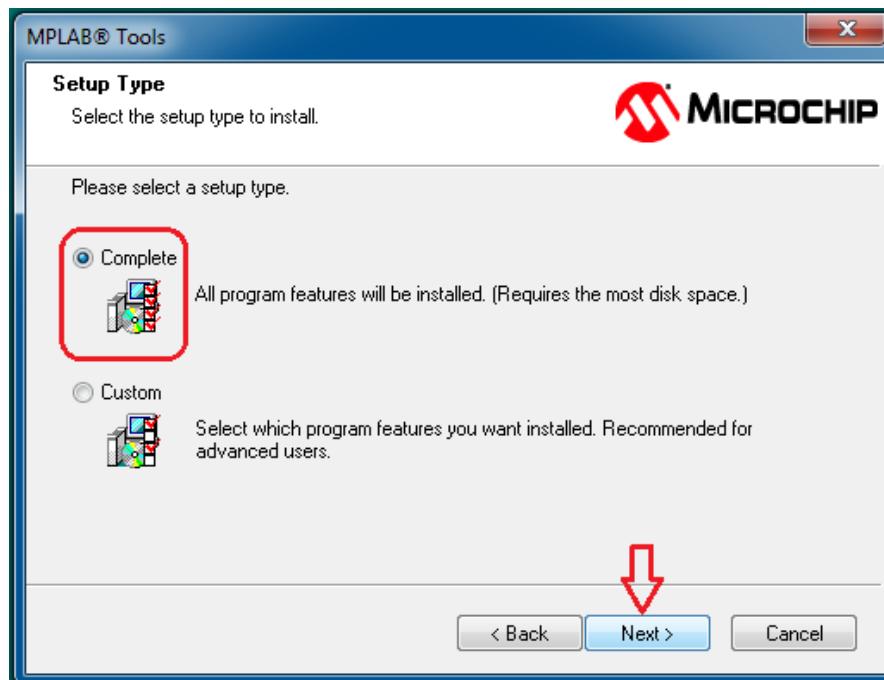


Figura 3.4 - Tipo de instalação. Escolha a opção "Completa".

- 4) Neste passo iremos escolher o diretório onde o MPLAB será instalado. É muito importante que o diretório seja C:\Microchip. Isso evita erros e padroniza a instalação. Para mudar para o diretório C:\Microchip clique em “browse”. Apague a frase “program files” ou “arquivo de programas” deixando apenas c:\Microchip. Clique em “OK”. Confira se o diretório é C:\Microchip. Se o diretório estiver correto clique em “Next” (ver Figura 3.5).

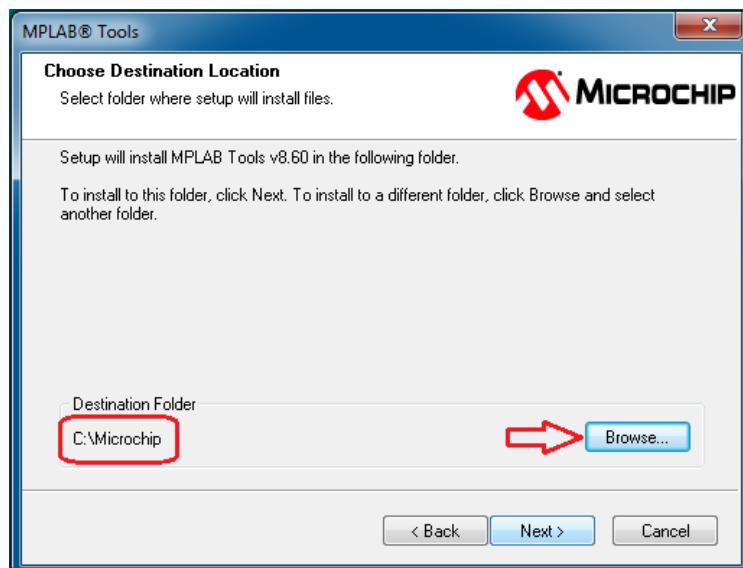


Figura 3.5 - Escolha do diretório onde o MPLAB será instalado.

- 5) Nesta etapa temos o termo de licença da aplicação “Maestro”. Este é uma ferramenta para configurar e incorporar uma série de módulos de firmware pré-escritos em sua aplicação. Ele é completamente compatível com o MPLAB IDE e foi desenvolvido pela própria Microchip. Leia com atenção toda a licença e, caso você concorde, clique em “**I accept the terms of the license agreement**”, em seguida clique em “Next”.
- 6) Em seguida aparecerá a opção para a instalação do compilador MPLAB C32. O C32 é um compilador para PICs da família 32. Leia com atenção toda a licença e, caso você concorde, clique em “**I accept the terms of the license agreement**” e depois clique em “Next”.
- 7) No passo seguinte aparecerá uma tela com um resumo do que será instalado. Confira se o diretório de instalação é C:\Microchip e clique em “Next” (ver Figura 3.6). Isso levará alguns minutos.

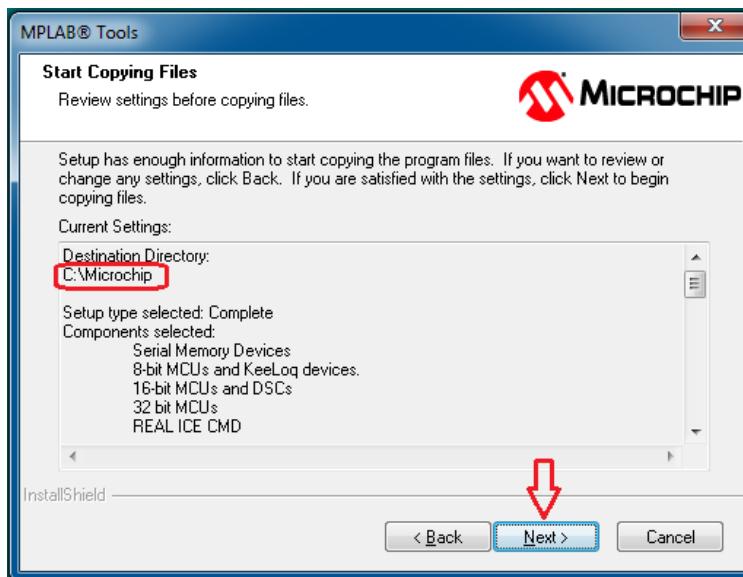


Figura 3.6 - Resumo do processo de instalação.

- 8) No final do processo de instalação irá aparecer uma mensagem pedindo para que o programa HI-TEC C *installer* seja instalado. Esse programa é um conjunto de compiladores para várias famílias de PIC, ferramentas de desenvolvimento para sistemas embarcados e uma IDE baseada em Eclipse (HI-TIDE™) para as famílias de 8, 16, e 32-bits do microcontrolador. Nós não utilizaremos esta ferramenta, desta forma, é desnecessária sua instalação. Clique em “Não” (ver Figura 3.7).

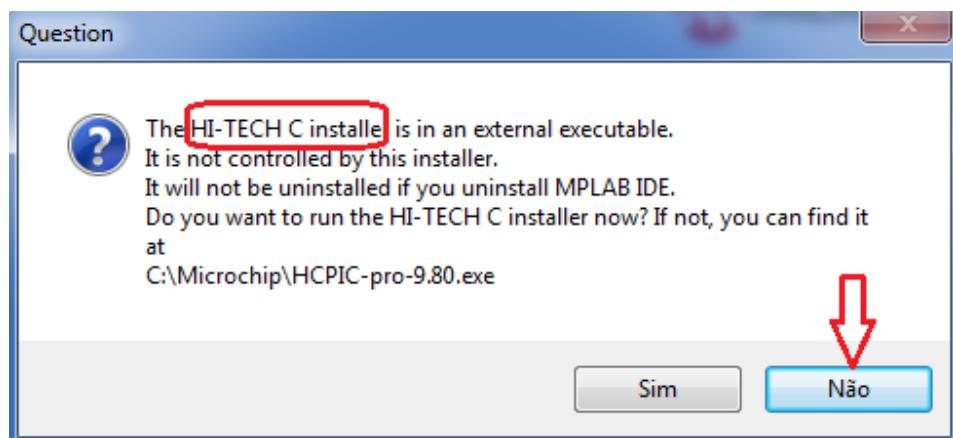


Figura 3.7 - Instalador do programa HI-TECH.

- 9) A Figura 3.8 traz uma tela informativa que indica que o processo de instalação foi concluído com sucesso. Clique em “Finish” para fechá-la.

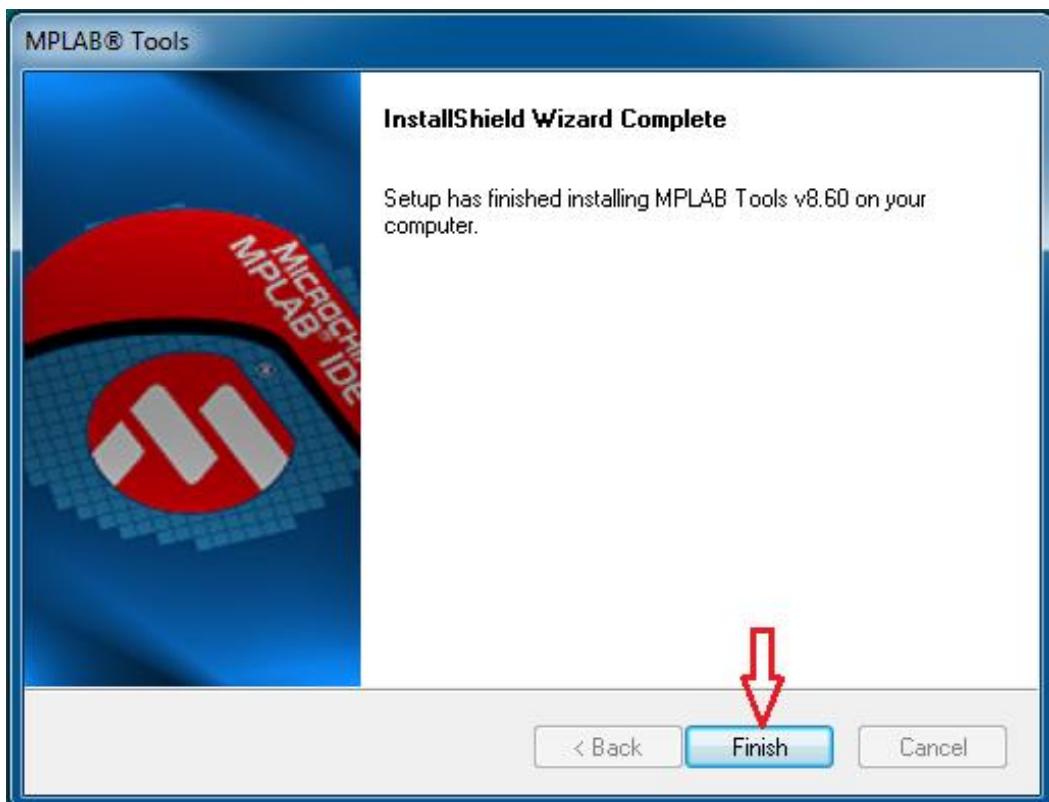


Figura 3.8 - Finalização da instalação.

- 10) Por fim, aparecerá uma janela informativa mostrando uma lista de documentos e notas de versões e instalações de drives referentes aos programas instalados. Você pode fechar essa janela.

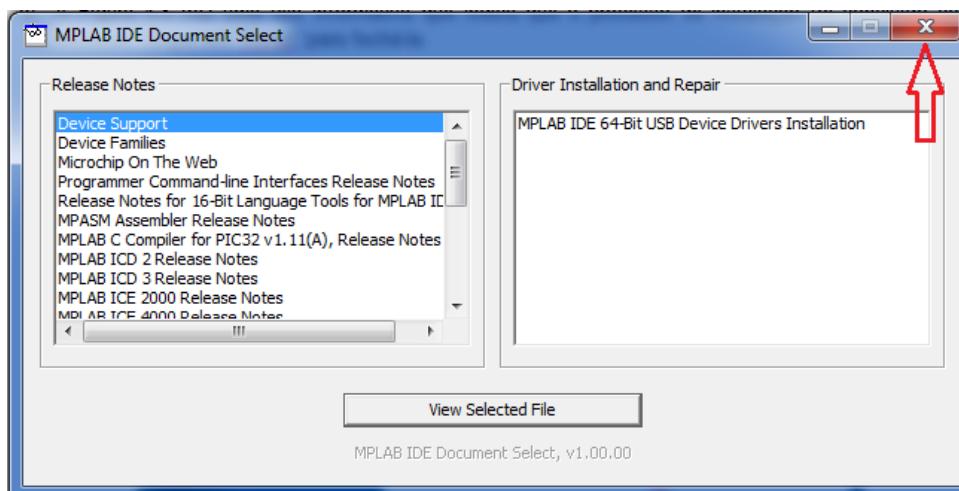


Figura 3.9 - Tela meramente informativa.

Com isso terminamos a instalação do MPLAB. Confira se um ícone do MPLAB foi criado no Desktop (Área de Trabalho) de seu computador.

Caso apareça um pedido para instalação do *Driver da Microchip Custom USB Device*, clique em “Instalar”.

3.2 COMPILADORES C18 E C32 DA MICROCHIP

A maneira de comandar um computador é através de um programa, sendo a linguagem de máquina a única linguagem entendida por ele. Assim é preciso traduzir os programas, que estão em uma linguagem de alto nível (mais próxima da linguagem humana), para linguagem de máquina (Figura 3.10), isso é feito por um tradutor chamado compilador.

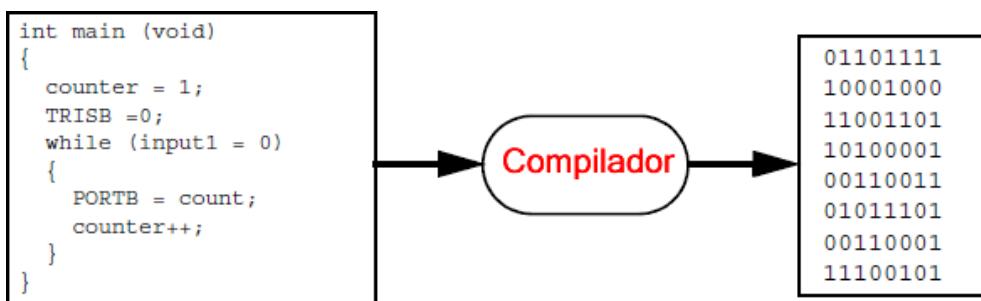


Figura 3.10 - Função do compilador.

Para a programação dos PICs (PIC18 e PIC32) utilizados em nossos cursos, são utilizados os compiladores C18 e C32 da própria Microchip. Eles permitem que o programador escreva códigos em linguagem C e, posteriormente, os “convertam” para a linguagem de máquina apropriada ao µC PIC utilizado.

Os compiladores C18 e C32 possuem duas versões: uma versão comercial que pode ser comprada no site da Microchip, e uma versão de estudante gratuita, disponível neste DVD, que diminui seu

poder de compilação após um período de avaliação (60 dias). No entanto, mesmo após expirar o tempo de avaliação, este compilador continuará atendendo as necessidades de desenvolvimento de projetos não comerciais.

Um projeto criado no MPLAB, é gerenciado por informações gravadas num arquivo de extensão .mcp (Projeto do MPLAB). A compilação de um projeto gera importantes arquivos, conforme mostrado na Figura 3.11.

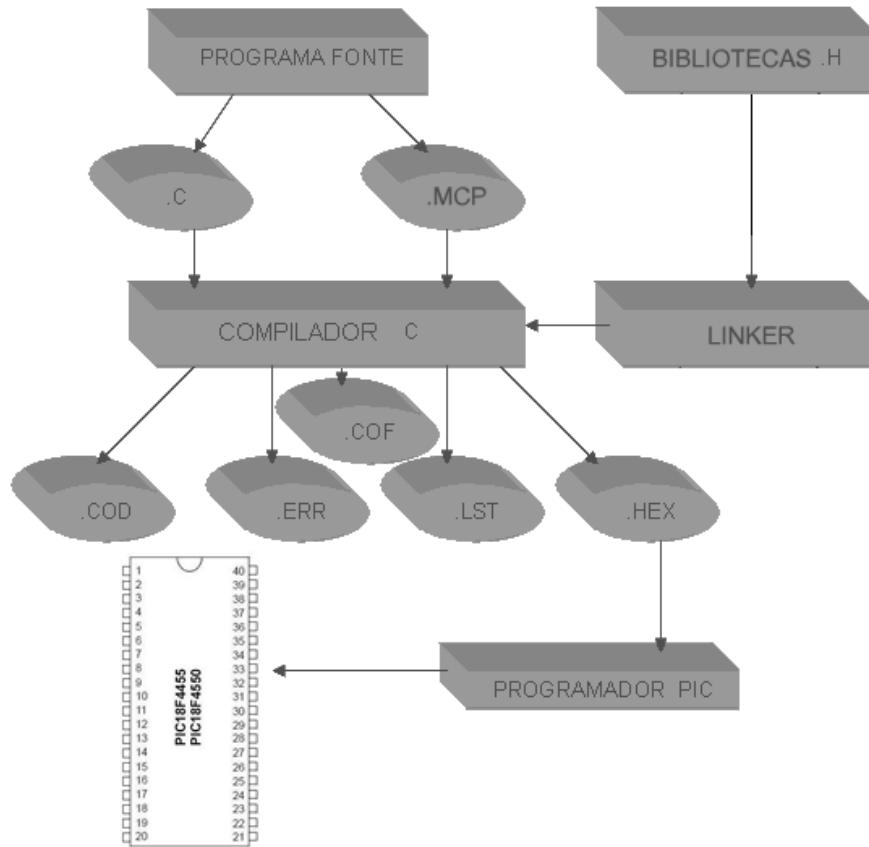


Figura 3.11 - Processo de compilação do código.

O programador escreve o código em C no MPLAB e inclui as bibliotecas que precisam ser usadas. O MPLAB utiliza o compilador para converter a **linguagem de programação**, também conhecida como **linguagem de alto nível** (neste caso linguagem C) para linguagem de máquina (**linguagem de baixo nível**). O compilador utiliza outro software chamado Linker (MPLINK) que irá instanciar as **bibliotecas** necessárias para o funcionamento do código em desenvolvimento. Finalmente, depois da “linkagem” e da compilação, temos como resultado o código em linguagem de máquina (**.HEX**).

O principal arquivo resultante no processo de compilação é o arquivo **HEX** (.HEX), que contém o código de máquina em formato hexadecimal, usado pelos simuladores (MPLAB SIM), emuladores (MPLAB ICE 2000 and PICMASTER) e programadores (PRO MATE II e PICSTART Plus). **OBS.:** para maiores informações sobre o MPLAB, consulte o MPLAB USER'S GUIDE (DS51519C), disponível no DVD didático, na seção “Datasheets”, em Manuais do MAPLAB – Guia do Usuário.

Alguns dos outros arquivos gerados são:

- ⊕ **Code file (.cod)** – Arquivo de depuração usado pelo MPLAB IDE, contém a informação simbólica e o código objeto;
- ⊕ **Listing file (.lst)** – Código fonte original, passo-a-passo com o código final em binário;
- ⊕ **Error file (.err)** – Arquivo de erros gerados pelo compilador;
- ⊕ **Configuration file (.cof)** – Arquivo de configuração do dispositivo usado, contém toda a configuração de portas de I/O, registradores e outros.

3.2.1. INSTALAÇÃO DO COMPILADOR C18

CARACTERÍSTICAS DO COMPILADOR C18

O compilador MPLAB C18 é livre apenas por 60 dias e é uma adaptação do compilador ANSI C para microcontroladores PIC18.

O compilador MPLAB C18 tem as seguintes características:

- ✚ Compatível com ANSI '89;
- ✚ Integração com o MPLAB IDE para ser fácil de usar gerenciando o projeto e o código no nível de depuração;
- ✚ Geração de módulo de realocação de objetos aumentando a reutilização do código;
- ✚ Compatibilidade com módulos de objetos gerados pelo MPASM assembler, permitindo a total liberdade em misturar programas em assembly e C em um único projeto;
- ✚ Acesso transparente de leitura/escrita para memória externa;
- ✚ Eficiente gerador de código com otimização em multi-nível;
- ✚ Extenso suporte as bibliotecas, incluindo PWM, SPI™, I2C™, UART, USART, manipulação de string e bibliotecas matemáticas.
- ✚ Total controle no nível usuário sobre alocação de dados.

Para instalação do compilador MPLAB C18 COMPILER siga o procedimento abaixo:

Procedimento

- 1) Para instalar o C18 a partir deste DVD, vá à aba “instaladores” e em “Compilador C18 Student Edition” clique no botão “Instalar”. O instalador abrirá automaticamente (ver Figura 3.12).



Figura 3.12 - Ícone Instaladores do DVD Didático - Instalador do Compilador C18.

- 2) No passo seguinte aparecerá uma tela de boas vindas à instalação. **É importante que você já tenha instalado o MPLAB em sua máquina e que este não esteja aberto.** Clique em “Next” para continuar (ver Figura 3.13).

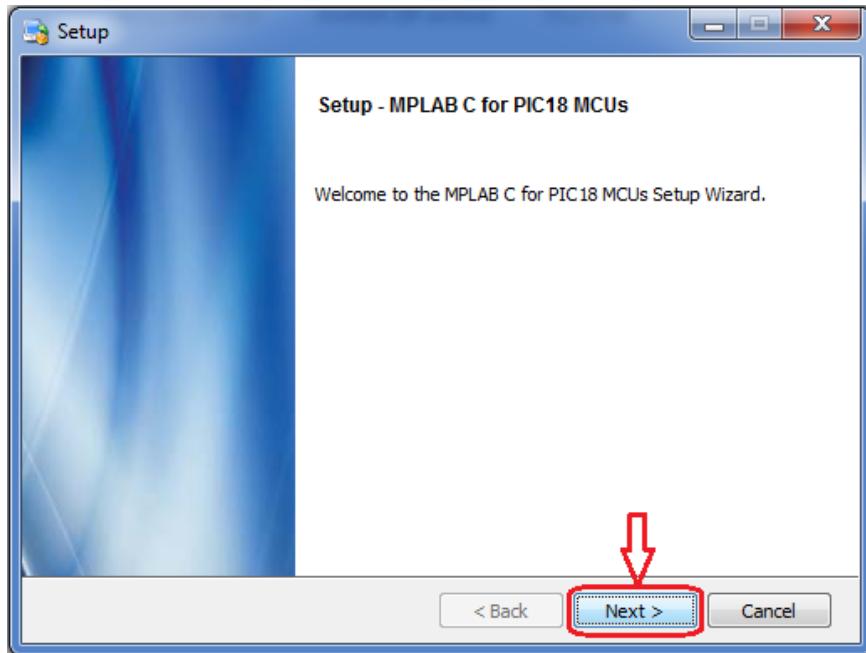


Figura 3.13 - Tela de boas vindas do instalador do compilador C18.

- 3) Na janela seguinte temos o termo de licença do C18. Leia atentamente e, caso você concorde, clique em “I accept the agreement” e, em seguida, clique em “Next” (ver Figura 3. 14).



Figura 3. 14 - Licença do compilador C18.

- 4) No próximo passo deve-se escolher o diretório onde o compilador C18 será instalado. É muito importante que o diretório seja C:\mplabc18. Mude o caminho padrão presente na instalação para o C:\mplabc18. Isso evita erros e padroniza a instalação. Clique em “Next” (ver Figura 3.15).

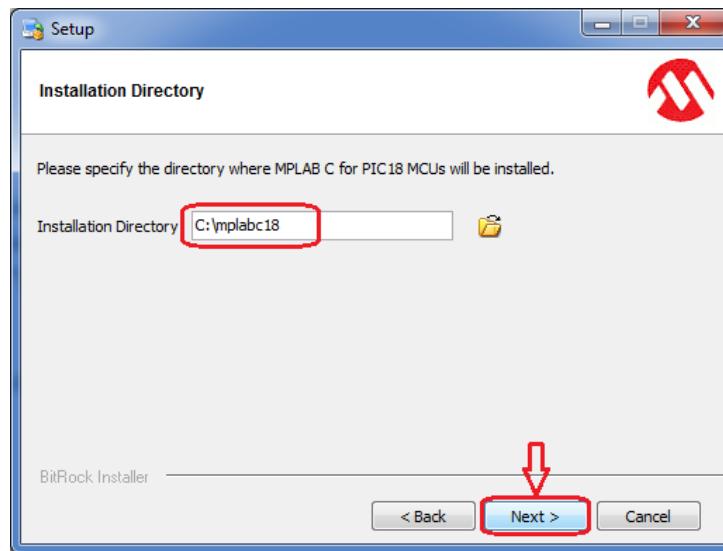


Figura 3.15 - Escolha do diretório onde será instalado o C18.

- 5) Em seguida, uma janela irá informar que o programa está pronto para iniciar a instalação do C18, clique em “Next” e aguarde. O processo de instalação levará alguns minutos.
- 6) Ao final do processo de instalação aparecerá uma janela informando que a instalação foi concluída com sucesso. Pressione “Finish” para fechá-la (ver Figura 3.16).

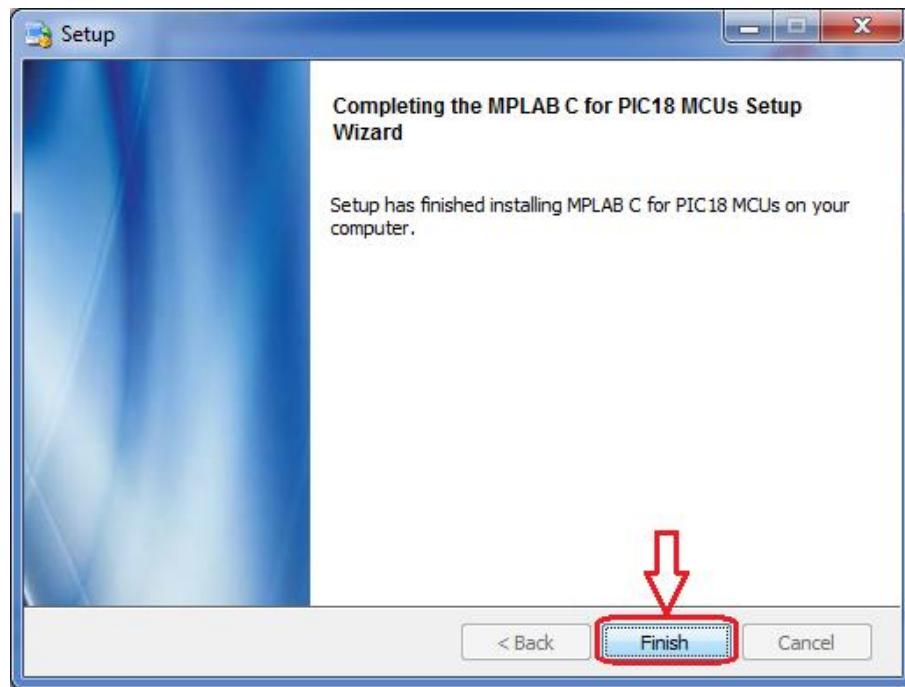


Figura 3.16 - Final do processo de instalação do compilador C18.

Pronto! O compilador C18 já foi instalado em seu computador.

RÁPIDO TOUR NOS DIRETÓRIOS

O diretório de instalação MPLAB C18 contém o arquivo leia-me para o compilador (readme.c18) e o arquivo para o linker (readme.lkr). E ainda, inúmeros subdiretórios estão também presentes. Na Tabela 5, temos a descrição detalhada desses subdiretórios:

Diretório	Descrição
bin	Contém os arquivos executáveis para o compilador e o linker. Eles são descritos mais detalhadamente no capítulo 4, seção 4.2 “Características do compilador”.
cpp	Contém o código fonte para o pré-processador MPLAB C18.
doc	Contém a documentação eletrônica do compilador MPLAB C18. Esses documentos devem ser consultados quando houver dúvidas relativas ao compilador MPLAB C18.
example	Contém aplicações exemplos para auxiliar o usuário a começar a usar o MPLAB C18.
h	Contém o arquivos cabeçalho (header) para as bibliotecas padrão C e bibliotecas específicas do microcontrolador suportados pelo compilador C18. São utilizados em C para definir variáveis, tipos, símbolos e funções úteis ao programa.
lib	Contém as bibliotecas padrão do C (clib.lib ou clib_e.lib), bibliotecas específicas do microcontrolador (p18xxxx.lib or p18xxxx_e.lib, onde xxxx é o número específico do dispositivo) e os módulos de inicialização (c018.o, c018_e.o, c018i.o, c018i_e.o, c018iz.o, c018iz_e.o).
lkr	Contém os arquivos linker script para os µC suportados no C18.
mpasm	Contém a versão da linha de comando para o assembler MPASM, os arquivos de cabeçalho assembly para os dispositivos suportados pelo MPLAB C18 (p18xxxx.inc) e os arquivos cabeçalhos assembly usados pelas bibliotecas.
Scr	Contém o código fonte, em formato C e arquivos assembly, para a biblioteca padrão C, bibliotecas específicas do processador e módulos de inicialização.

Tabela 5 - Descrição dos subdiretórios do MCC18.

3.2.2. INSTALAÇÃO DO COMPILADOR C32

Para instalação do compilador MPLAB C32 COMPILER siga o procedimento abaixo:

Procedimento

- 1) Para instalar o C32 a partir deste DVD, vá à aba “instaladores” e em “Compilador C32 Student Edition” clique no botão “Instalar”. O instalador abrirá automaticamente (ver Figura 3.17).



Figura 3.17 - Ícone Instaladores do DVD Didático - Instalador do Compilador C32.

- 2) No passo seguinte aparecerá uma tela de boas vindas à instalação. **É importante que você já tenha instalado o MPLAB em sua máquina e que este não esteja aberto.** Clique em “Next” para continuar (ver Figura 3.18).

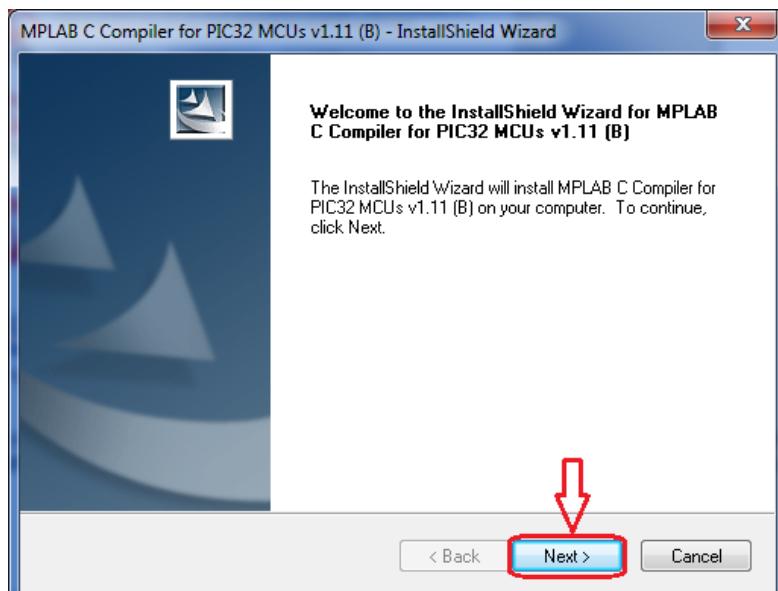


Figura 3.18 - Janela de boas vindas do instalador do compilador C32.

- 3) Na janela seguinte temos o termo de licença do C32. Leia atentamente e, caso você concorde, clique em "I accept the agreement" e, em seguida, clique em "Next" (ver Figura 3.19).

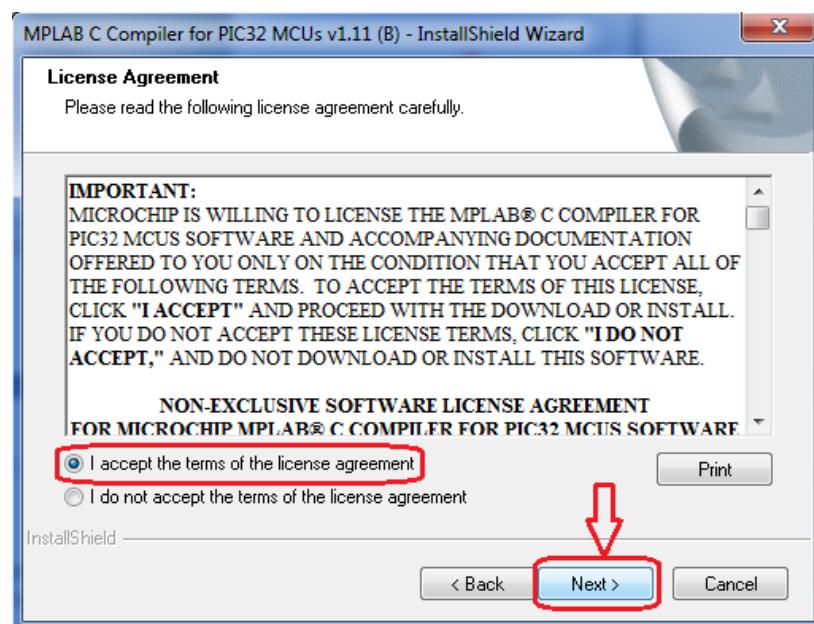


Figura 3.19 - Termo de licença do compilador C32.

- 4) No próximo passo deve-se escolher o diretório onde o compilador C32 será instalado. É muito importante que o diretório seja C:\MPLAB C32. Mude o caminho padrão presente na instalação para o C:\ MPLAB C32. Isso evita erros e padroniza a instalação. Clique em "Next" (ver Figura 3.20).

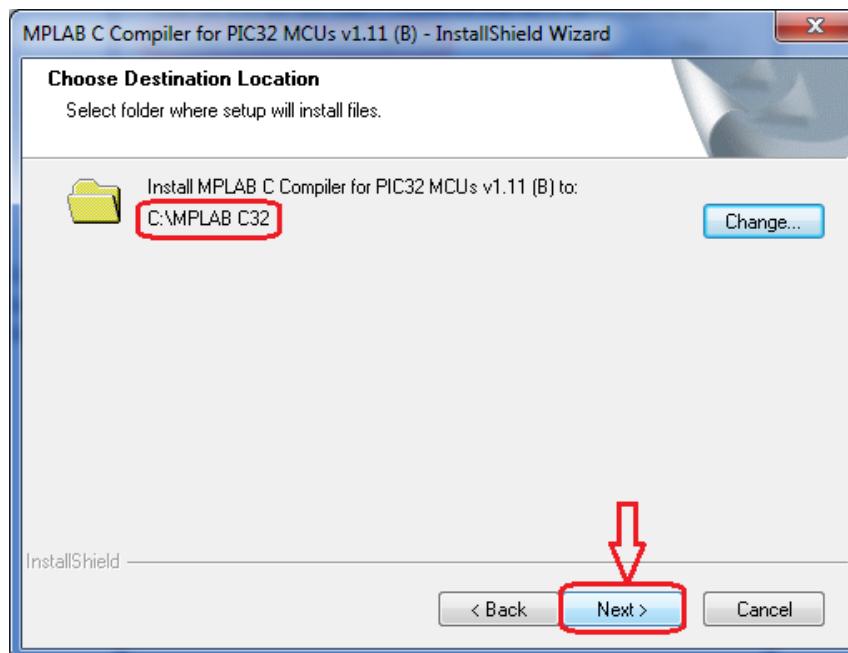


Figura 3.20 - Escolha do diretório onde será instalado o C32.

- 5) Em seguida, uma janela irá informar que o programa está pronto para iniciar a instalação do C32, clique em "Install" e aguarde. O processo de instalação levará alguns minutos.

- 6) Ao final do processo de instalação aparecerá uma janela informando que a instalação foi concluída com sucesso. Pressione “*finish*” para fechá-la (ver Figura 3.21).

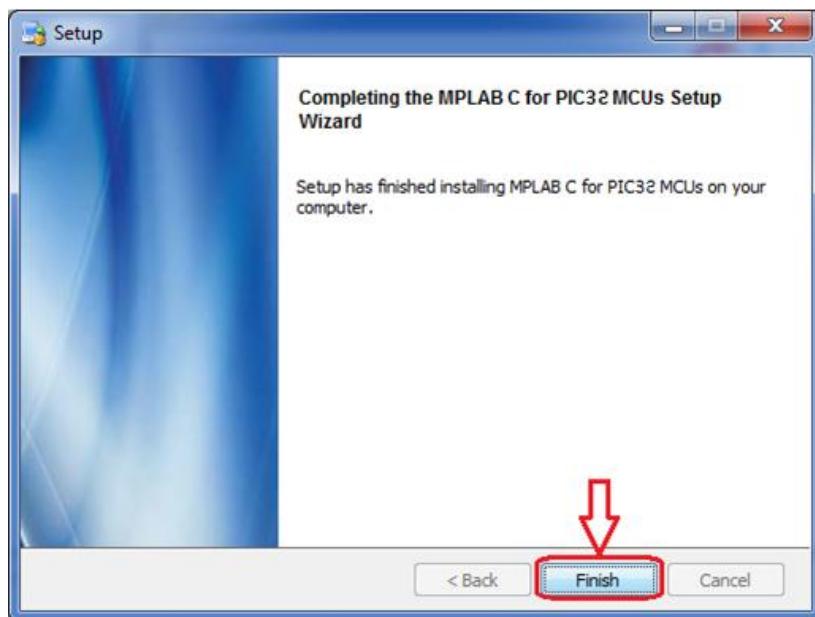


Figura 3.21 - Final do processo de instalação do compilador C32.

Pronto! O compilador C32 já foi instalado em seu computador.

3.3 FIRMWARE BOOTLOADER

Para falarmos sobre *Bootloader* devemos primeiro entender a diferença entre um *firmware* e um *software*. *Firmware* é um programa que fica gravado dentro de um sistema embarcado (em sua memória de programa: ROM, EEPROM e FLASH), e nele estão contidas todas as instruções de funcionamento, reconhecimento e operação de um determinado equipamento eletrônico (como por exemplo, máquinas de lavar, micro-ondas, celulares, vídeos-game, câmeras fotográficas, entre outros), em suma, é um programa desenvolvido para um *hardware* específico. Já o *software* é um programa desenvolvido para executar em computadores de modo geral (respeitando alguns requisitos mínimos para o seu funcionamento), não estando vinculado a um *hardware* específico. Ou seja, um *software* pode ser desenvolvido para funcionar em um computador independente de sua marca (Apple, HP, Dell, Toshiba, etc.) ou modelo.

Baseado neste raciocínio, você saberia responder se o programa *Word* da *Microsoft* seria um *software* ou um *firmware*? Se o *Word* fosse um *firmware*, deveriam existir inúmeras versões do mesmo, uma para cada marca e modelo de computador existente no mercado. Desta forma, fica fácil concluir que o programa *Word*, assim como o próprio Sistema Operacional *Windows*, são exemplos de *softwares*.

Outro exemplo seria um programa para controlar as funções de um forno de micro-ondas. Este seria um *software* ou *firmware*? Para respondermos a esta pergunta vamos nos fazer outra pergunta: seria possível gravar o programa responsável pelo funcionamento de um forno de micro-ondas da marca LG em outro forno de micro-ondas da marca Consul e este último continuar funcionando? Provavelmente não, pois esses programas são desenvolvidos para a montagem específica de cada modelo de micro-ondas, pensando em seus dispositivos eletrônicos como, por exemplo, tipo de teclado, número de funcionalidades, tipo de display (LCD, 7 segmentos, touch-screen, etc), características elétricas do motor que gira o prato e até mesmo do microcontrolador utilizado para controlar todos esses periféricos. Desta forma, fica fácil concluir que o programa gravado em um modelo específico de forno de micro-ondas da marca LG foi desenvolvido especificamente para aquele equipamento, não sendo possível gravá-lo nem em outros

modelos de micro-ondas da própria LG. Isso nos permite afirmar que este programa trata-se de um *firmware* e não de um *software*, uma vez que o mesmo foi feito para um *hardware* específico.

Firmware	Software
<ul style="list-style-type: none"> Desenvolvido especificamente para um determinado dispositivo eletrônico (Microcontrolador + circuito específico com os componentes ligados ao µC). 	<ul style="list-style-type: none"> Desenvolvido para rodar em qualquer computador com o mesmo sistema operacional (Windows, Linux, OS e outros) independente da configuração do <i>hardware</i>.
<ul style="list-style-type: none"> Depende da estrutura externa ligada ao µC, uma vez mudada a estrutura deve mudar o <i>firmware</i>. Por exemplo: um programa desenvolvido para executar numa TV Toshiba não funciona numa TV LG. 	<ul style="list-style-type: none"> Não depende de todos os dispositivos (placa de vídeo, som, modem) ligados no computador. Funciona apenas com a configuração mínima.
<ul style="list-style-type: none"> Exemplos de equipamentos que usam <i>firmware</i>: micro-ondas, DVD player, TV, controle remoto, celular, videogame, relógio digital e etc. 	<ul style="list-style-type: none"> Exemplos de equipamentos que usam <i>software</i>: computadores pessoais, celulares modernos com sistema operacional embarcado (Windows Mobile, OS, Linux).

Outro conceito importante que devemos abordar antes de falarmos do *Bootloader* são os tipos de gravação que um microcontrolador pode ter. Ou seja, as possíveis formas de se armazenar (carregar) o programa desenvolvido para um determinado microcontrolador dentro de sua memória de programa. Este programa irá controlar todas as funcionalidades do microcontrolador para uma aplicação específica. Iremos chamá-lo de **Firmware de Aplicação**.

O modo mais comum de gravar o **Firmware de Aplicação** é chamado de gravação *off-board*. Nesse método é necessário retirar o microcontrolador da sua plataforma de aplicação (placa de circuito impresso que liga os periféricos do microcontrolador aos dispositivos externos que ele irá controlar, como por exemplo: botões, relés, LEDs, display, etc.) e colocá-lo em outro dispositivo, chamado **Gravadora**, que irá gerenciar a transferência do **Firmware de Aplicação** de dentro do computador para a memória de programa do microcontrolador. Este método será melhor detalhado na seção 3.6. A Figura 3.22 ilustra esse tipo de gravação.

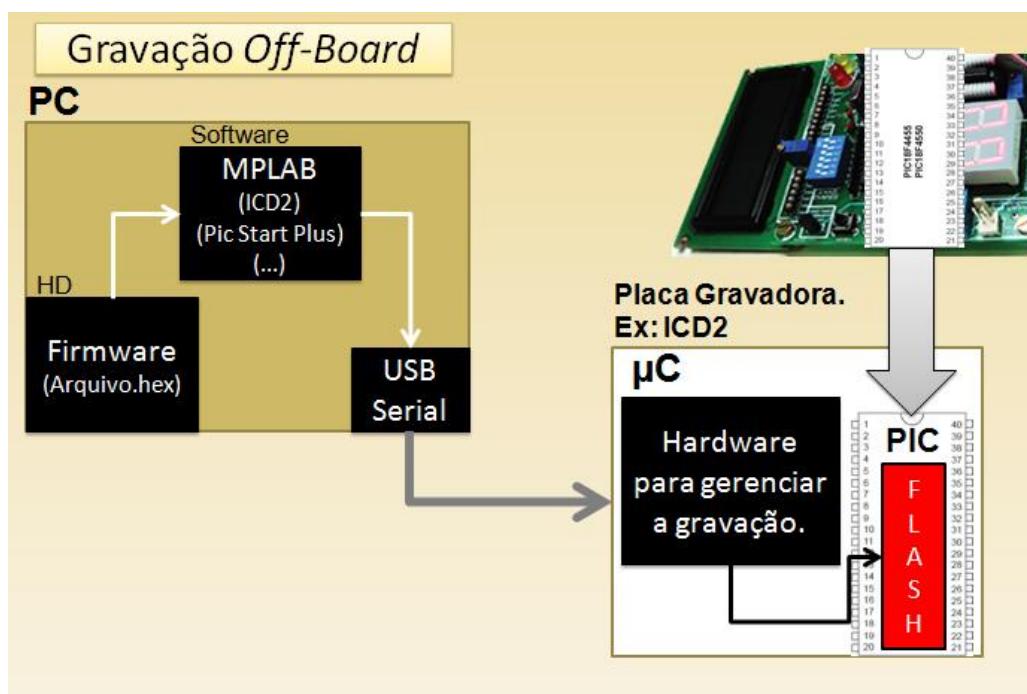


Figura 3.22 - Método de Gravação Off-Board.

Existem ainda outros dois métodos de gravação: *In-circuit* via Hardware e *In-circuit* via Bootloader. Nesses métodos, não há a necessidade de retirar o microcontrolador da placa de aplicação para realizar a gravação do firmware em sua memória de programa (*In-circuit* – no circuito).

In-circuit via Hardware: Neste modo, apesar de não precisar retirar o PIC do KIT, ainda é necessário o uso de uma gravadora. Além disso, a placa de aplicação (por exemplo o KIT Didático) deve possuir alguma interface de comunicação com a gravadora, pois será a gravadora que gerenciará o processo de transferência e armazenamento na memória de programa do microcontrolador. Para os PICs um método de conexão entre gravadora e placa de aplicação é o ICSP (*In-Circuit Serial Programmer*), compatível com a maioria das gravadoras de PIC existentes no mercado. Perceba na Figura 3.23 que o KIT Didático está conectado à gravadora por meio de seu conector ICSP (ver detalhes do KIT na seção 2.3 desta apostila). Assim como no método off-board, neste método é necessário conectar a gravadora ao MPLAB (via conexão serial ou USB). O MPLAB busca o arquivo.hex (firmware compilado) no HD do computador e o envia para a gravadora via porta USB ou serial. A gravadora acessa a memória de programa do PIC (memória FLASH) via conexão ICSP e armazena o todo o arquivo.hex. (ver Figura 3.23)

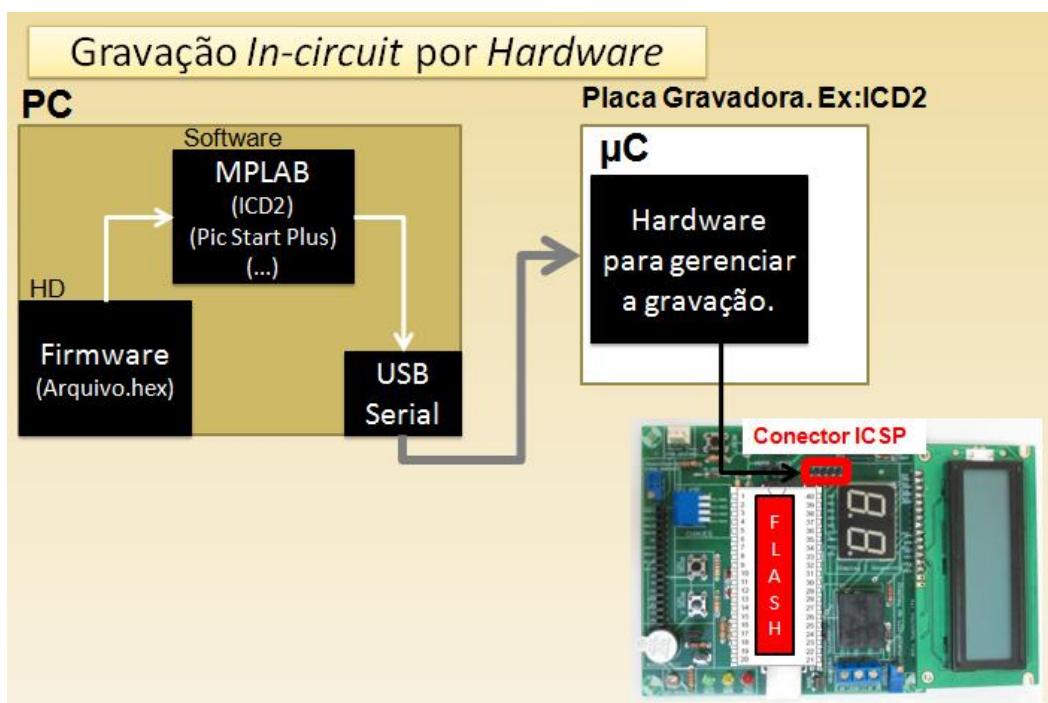


Figura 3.23 - Modo de gravação *in-circuit* via hardware.

***In-circuit* via Hardware:** Este é um método de autogravação, ou seja, o próprio microcontrolador onde se deseja carregar o arquivo.hex irá gerenciar todo o processo de gravação. Isso é possível porque em uma pequena parte de sua memória de programa (FLASH) foi previamente gravado outro firmware, que irá coordenar a transferência do **Firmware de Aplicação** (arquivo.hex) do computador para a memória a própria memória de programa do microcontrolador. Esse firmware é chamado de **Bootloader**, expressão em inglês que significa **Rotina de Partida**.

O **Firmware Bootloader** (previamente gravado em uma pequena parte da memória de programa do microcontrolador por um dos outros dois métodos mencionados anteriormente) é o programa que inicia a gravação (o carregamento) de **Firmwares de Aplicação** no microcontrolador pelo método *in-circuit* via Bootloader. No caso do PIC18F4550 isso é feito através da porta USB. O Bootloader irá comunicar com o software de gravação de nosso DVD Didático, presente no ícone “Gravar PIC” (ver Figura 3.24). Maiores detalhes sobre este procedimento são vistos na seção 3.4 ou nos vídeos de nosso DVD Didático.

A maior vantagem de sistemas que possuem a funcionalidade de autogravação é não precisar retirar o microcontrolador de sua plataforma de aplicação e de não fazer uso de uma gravadora.

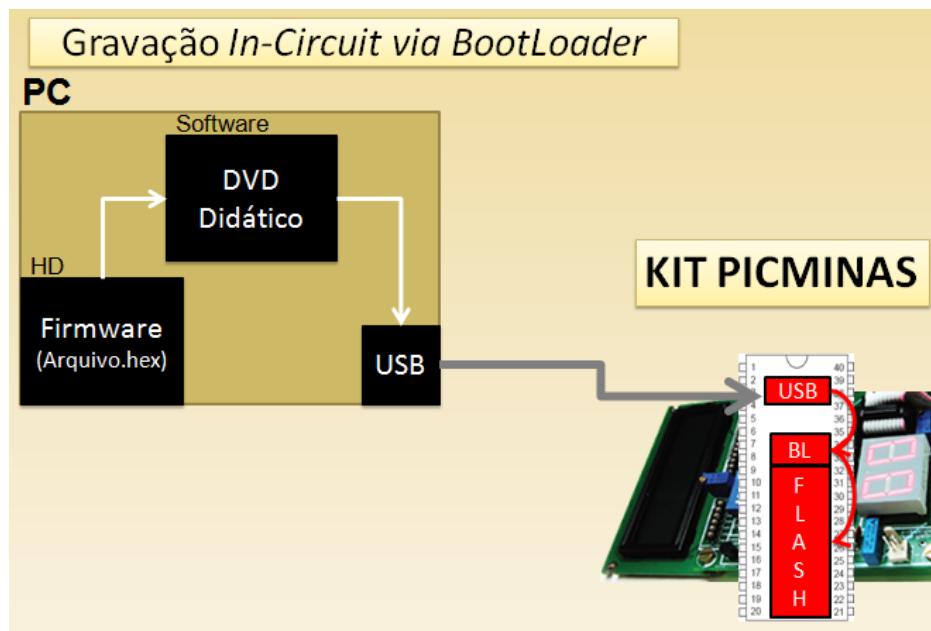


Figura 3.24 - Método de Autogravação.

O *Bootloader* no PIC18F4550 usa a memória do endereço 0x000h até o endereço 0x800h. Este é o único programa que deve ser gravado na memória interna do PIC, por meio de um circuito de gravação externo (uma gravadora PIC).

3.4 COMO CRIAR UM PROJETO NO MPLAB

O primeiro passo para desenvolver uma solução utilizando um microcontrolador da Microchip é a criação de um projeto. O projeto contém as informações do microcontrolador e do compilador utilizado, os diretórios onde estão os arquivos fonte e as bibliotecas, configurações da *workspace*, entre outros. O MPLAB disponibiliza uma ferramenta para facilitar a criação de projetos chamada “Project Wizard” que será utilizada nesta seção.

Para criar um projeto siga os seguintes passos:

Procedimento

- Crie uma pasta em seu computador com o nome do seu projeto. Abra o DVD didático e clique no ícone “Projetos”. No menu “Projetos” (à esquerda da tela), em “Arquivos Modelos”, clique sobre a família do PIC desejado (PIC18 ou PIC32). Em seguida, clique no botão “Salvar Modelo” e escolha o diretório onde deseja salvar os arquivos. Os arquivos modelo devem ser salvos dentro da pasta de projeto que você acabou de criar (ver Figura 3.25).

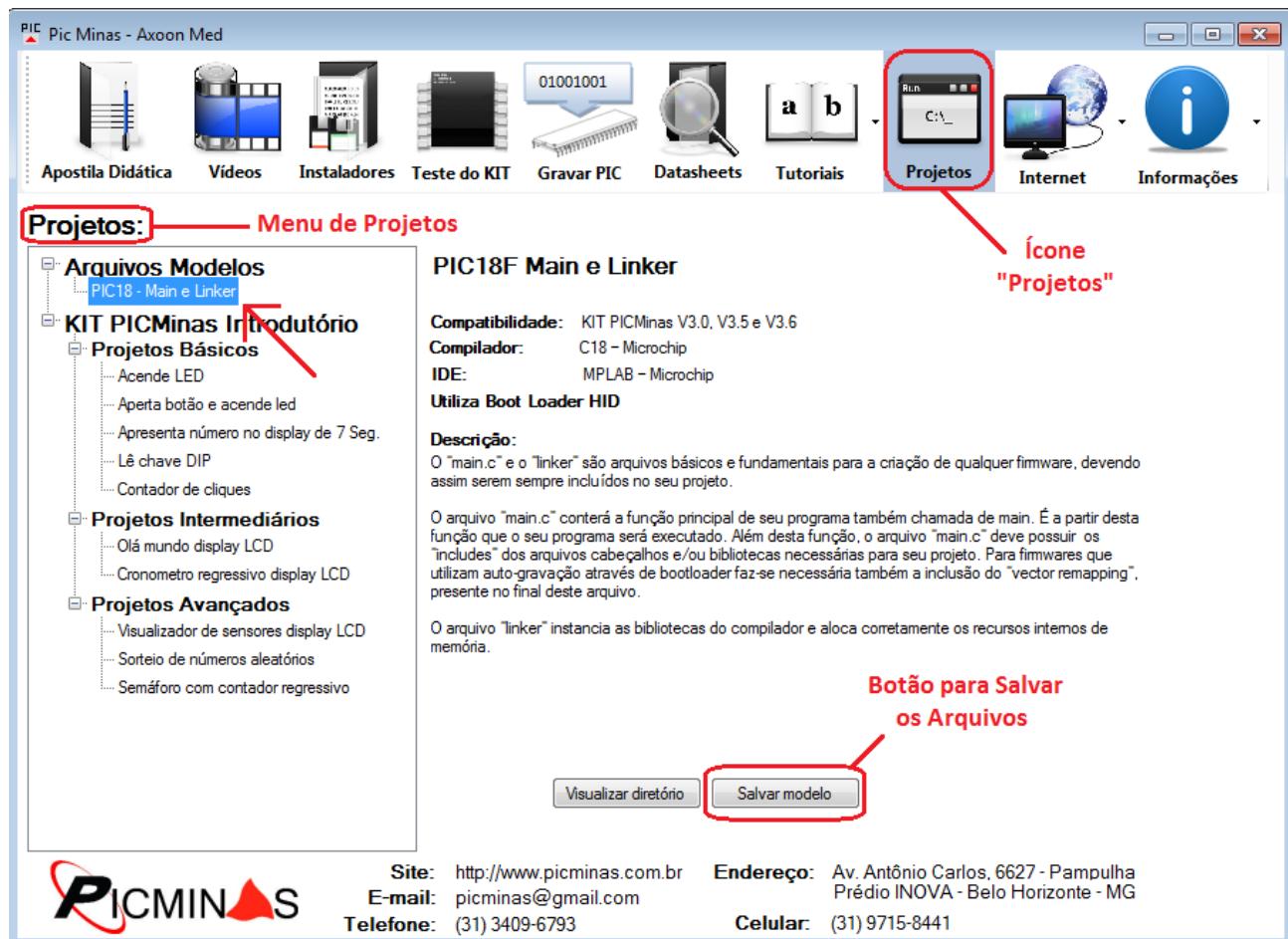


Figura 3.25 - Arquivos modelo no DVD didático.

- 2) Abra o MPLAB. Clique no menu “Project” e em seguida em “Project Wizard” (ver Figura 3.26)

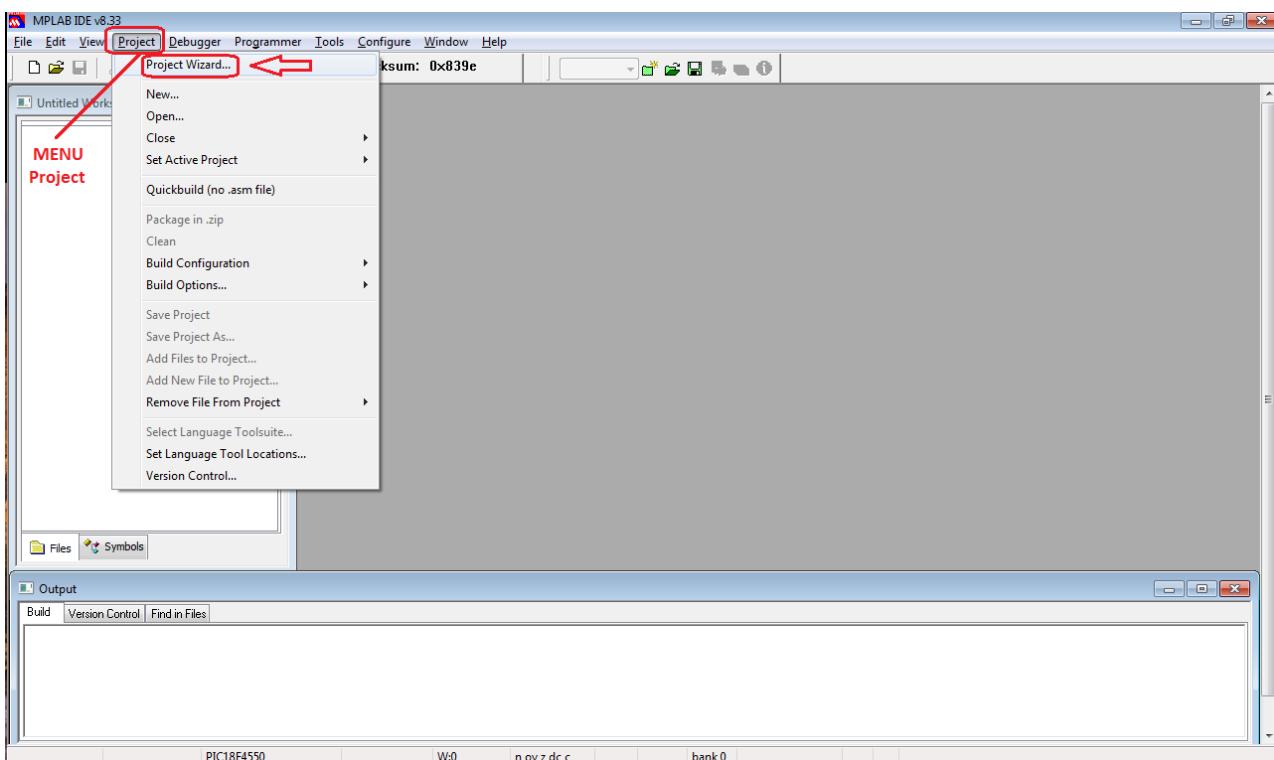


Figura 3.26 - Janela do MPLAB - Acessando o "Project Wizard".

- 3) Abrirá uma janela de boas vindas ao assistente de criação de projetos. Clique em avançar (ver Figura 3.27)

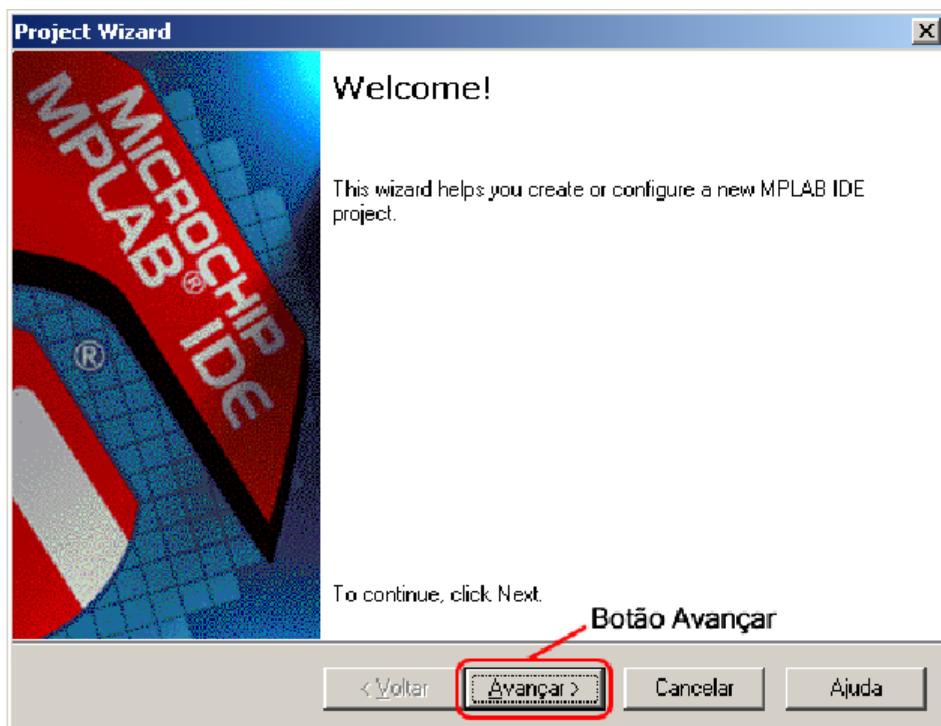


Figura 3.27 - Janela de Boas Vindas.

- 4) A tela seguinte permite a seleção do dispositivo a ser utilizado. Em nossos KITS utilizamos o PIC18F4550 ou o PIC32MX775F256L. Selecione o microcontrolador desejado e, em seguida, clique no botão avançar (ver Figura 3.28).

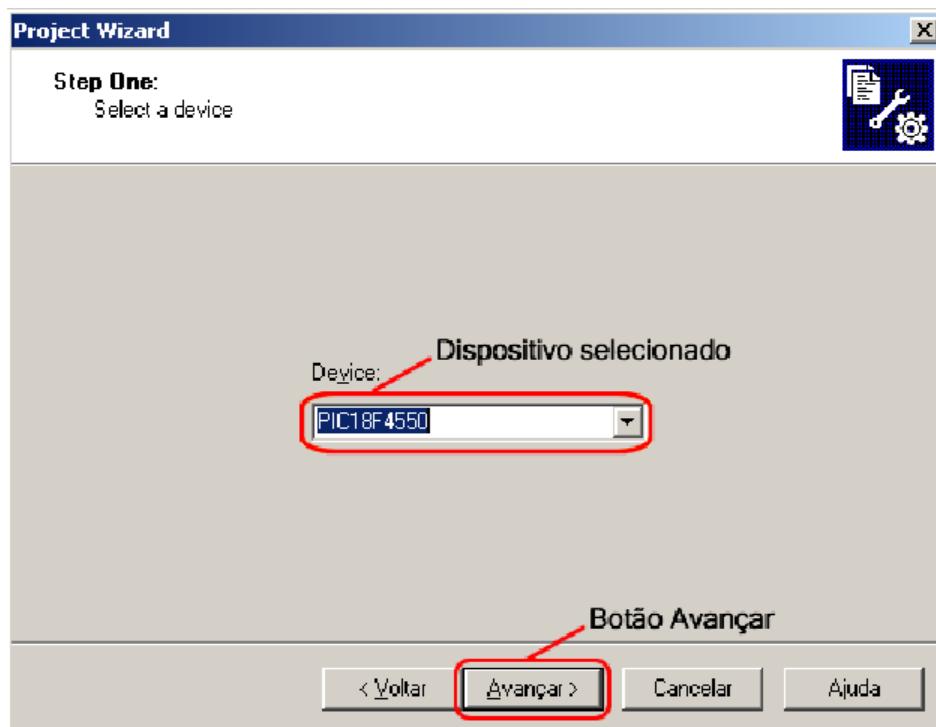


Figura 3.28 - Escolha do microcontrolador.

- 5) Na próxima tela, é necessário selecionar qual o compilador a ser utilizado no projeto. Isto é feito através da opção “Active Toolsuite”. Caso esteja utilizando o PIC18F4550 escolha a opção “Microchip C18 Toolsuite”. Caso esteja utilizando nosso KIT avançado com o PIC32, escolha a opção “Microchip PIC32 C-Compiler Toolsuite”. Em seguida, clique em avançar (ver Figura 3.29).

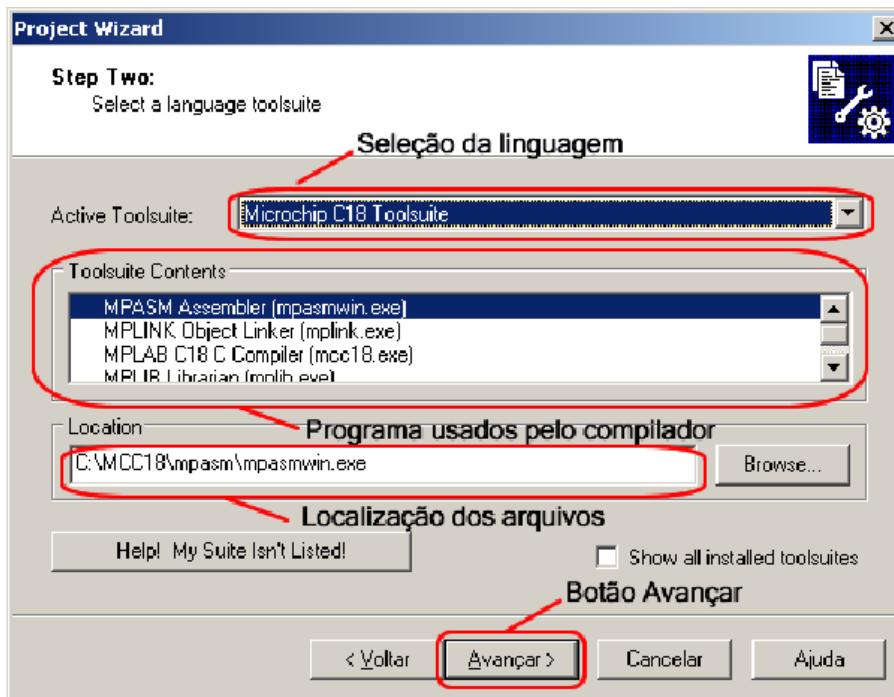


Figura 3.29 - Escolha da "Active Toolsuite".

- 6) A próxima tela permite ao usuário escolher o diretório onde o projeto será salvo, bem como o nome do mesmo. Para tanto, clique em “Browser” e escolha o diretório e o nome do projeto. Em seguida, clique em avançar.

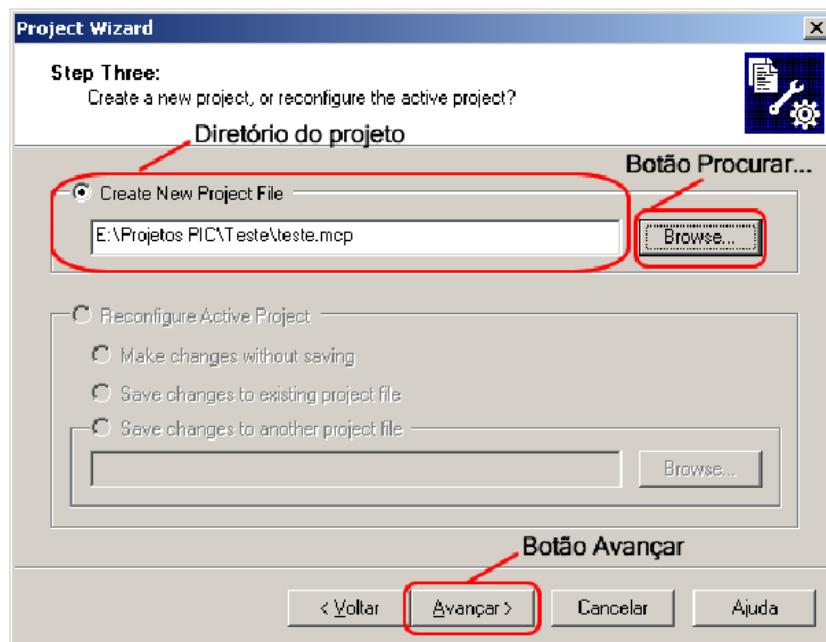


Figura 3.30 - Escolha do diretório.

- 7) A próxima tela permite adicionar arquivos ao seu projeto. Nós não recomendamos adicionar arquivos através desta janela devido à grande complexidade dos modos de importação de arquivos. Apenas clique em “avançar”. (ver Figura 3. 31)

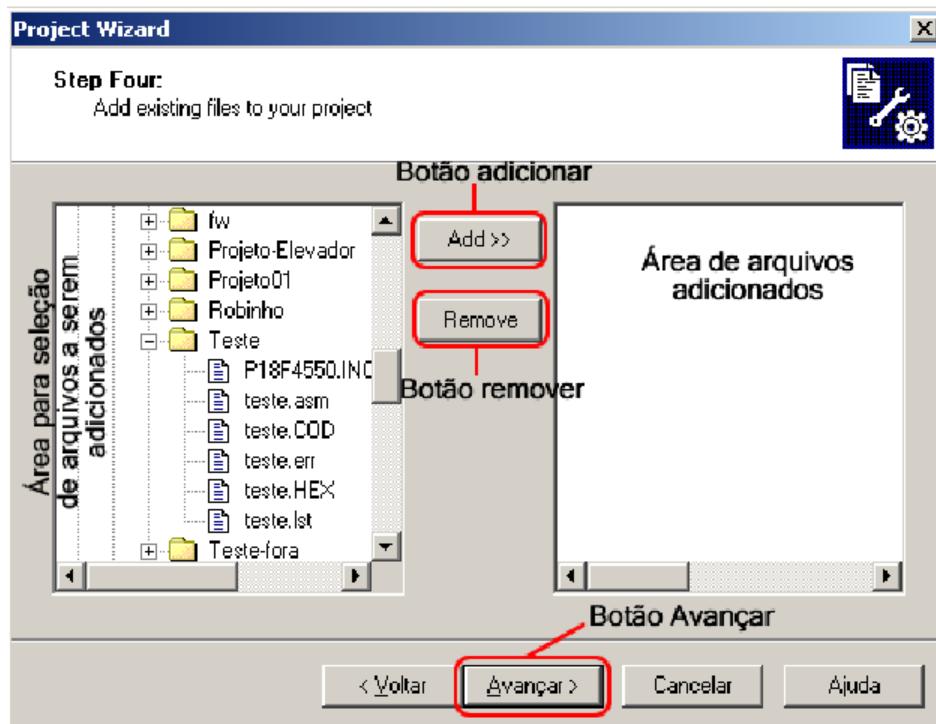


Figura 3. 31 - Inserir arquivos.

- 8) A próxima janela mostra um resumo dos dados do seu projeto. Confira o microcontrolador escolhido, o compilador, o caminho do projeto. Se todas as informações estiverem corretas clique em “concluir” (ver Figura 3.32).

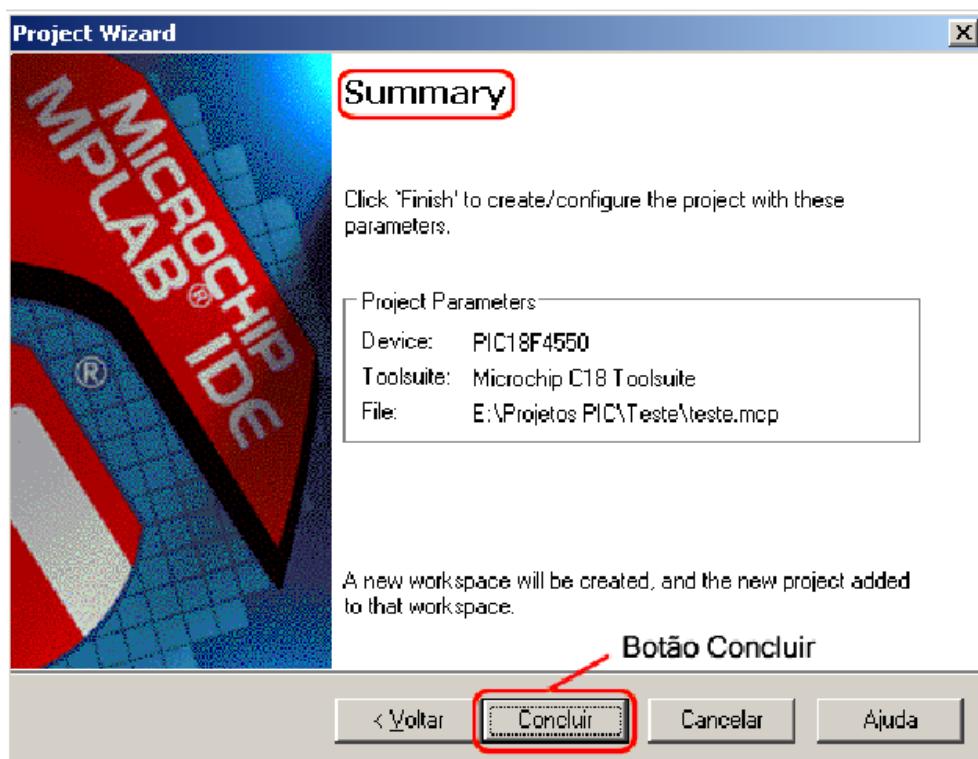


Figura 3.32 - Resumo do projeto.

- 9) O próximo passo é adicionar os arquivos main.c e linker ao seu projeto. Recomendamos que utilizem os arquivos disponíveis em nosso DVD. Para adicionar arquivos ao seu projeto, utilize à janela do Projeto (Ver Figura 3.33). Nesta janela você pode adicionar arquivos fontes (Source Files), arquivos cabeçalhos (Header Files), arquivos objetos (Object Files), arquivos de biblioteca (Library Files).

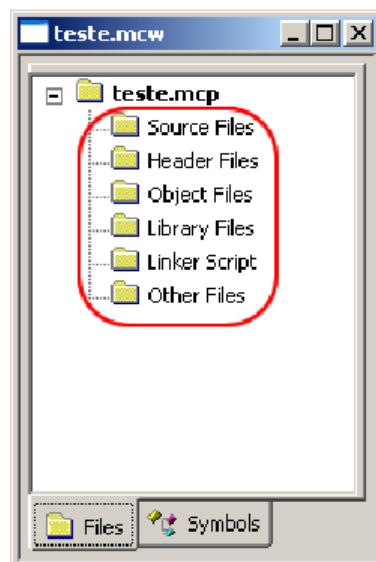


Figura 3.33 - Janela "Project".

- 10) Para adicionar o arquivo main.c em seu projeto, clique com o botão direito do mouse sobre a pasta “Source Files”, e em seguida clique em “Add Files” (ver Figura 3.34).

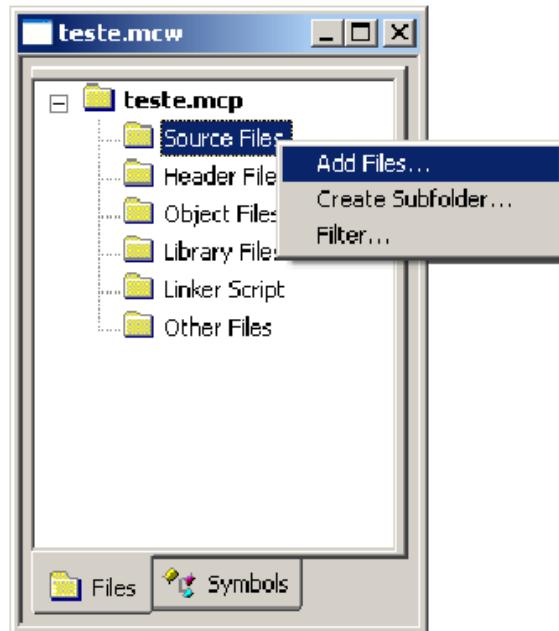


Figura 3.34 - Adicionando arquivos ao projeto.

- 11) Abrirá uma janela de busca. Localize seu arquivo (previamente copiado em seu computador, dentro do diretório do seu projeto) e, em seguida, clique em “abrir” (ver Figura 3.35).

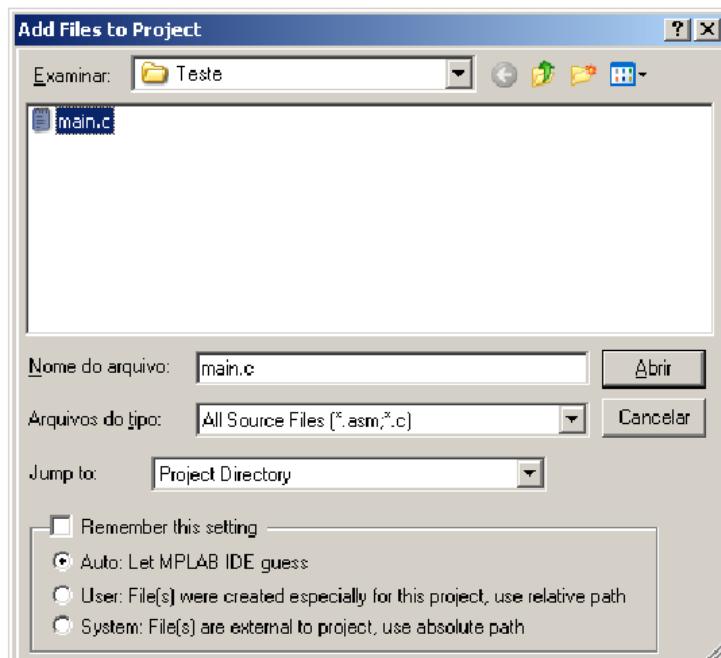


Figura 3.35 - Adicionando arquivos ao projeto.

- 12) Para incluir o arquivo linker (.lkr) siga as mesmas instruções dos passos 10 e 11. No entanto, este arquivo deve ser incluído na pasta “Linker Script”. Para isso, clique com o botão direito na pasta “Linker Script” e, em seguida, clique em “Add Files”. Encontre o arquivo de linker e clique em “abrir”.

- 13) Seu projeto está pronto. O passo seguinte será abrir o arquivo main.c e começar a desenvolver o seu código. Para abrir o arquivo main.c, vá na janela “Project”, na pasta “Source Files” e dê um duplo clique em main.c (ver Figura 3.36).

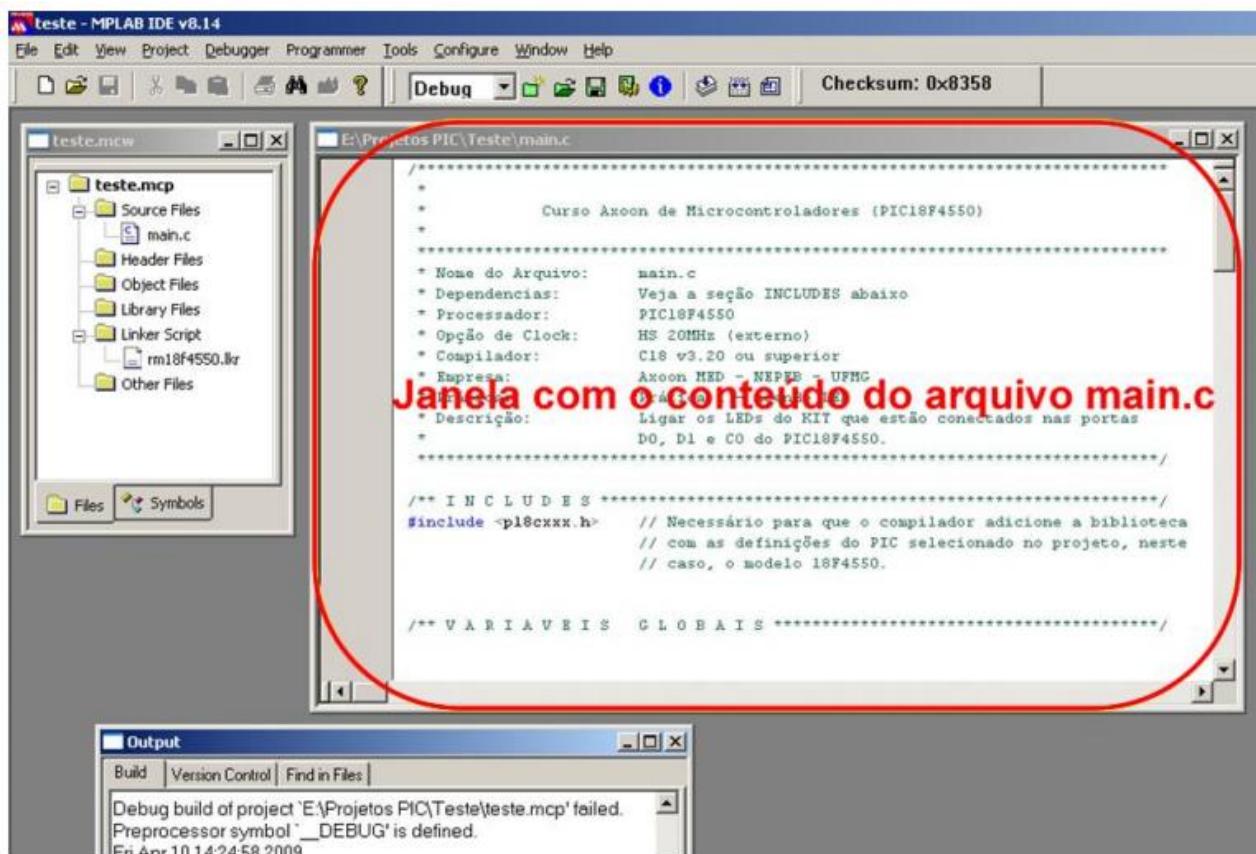


Figura 3.36 - Janela do MPLAB mostrando o arquivo main.c do projeto teste.mcp aberto.

3.5 COMO COMPIRAR E GRAVAR UM FIRMWARE NO KIT DIDÁTICO UTILIZANDO BOOTLOADER

Após a implementação de um programa no MPLAB os próximos passos seriam a compilação e gravação do código no KIT Didático. Compilar é transformar o código escrito em linguagem C, para linguagem de máquina, hexadecimal (para maiores detalhes veja a seção 3.2).

O processo de compilação de seu projeto é muito simples. Vá à barra ferramentas e clique sobre o botão “build all”. Observe na janela “Output” as mensagens do compilador. Caso seu projeto não possua erros, irá aparecer à mensagem “BUILD SUCCEEDED” indicando que a compilação foi efetuada com sucesso (ver Figura 3.37). Você também pode pressionar o atalho “ctrl+F10” para iniciar a compilação.

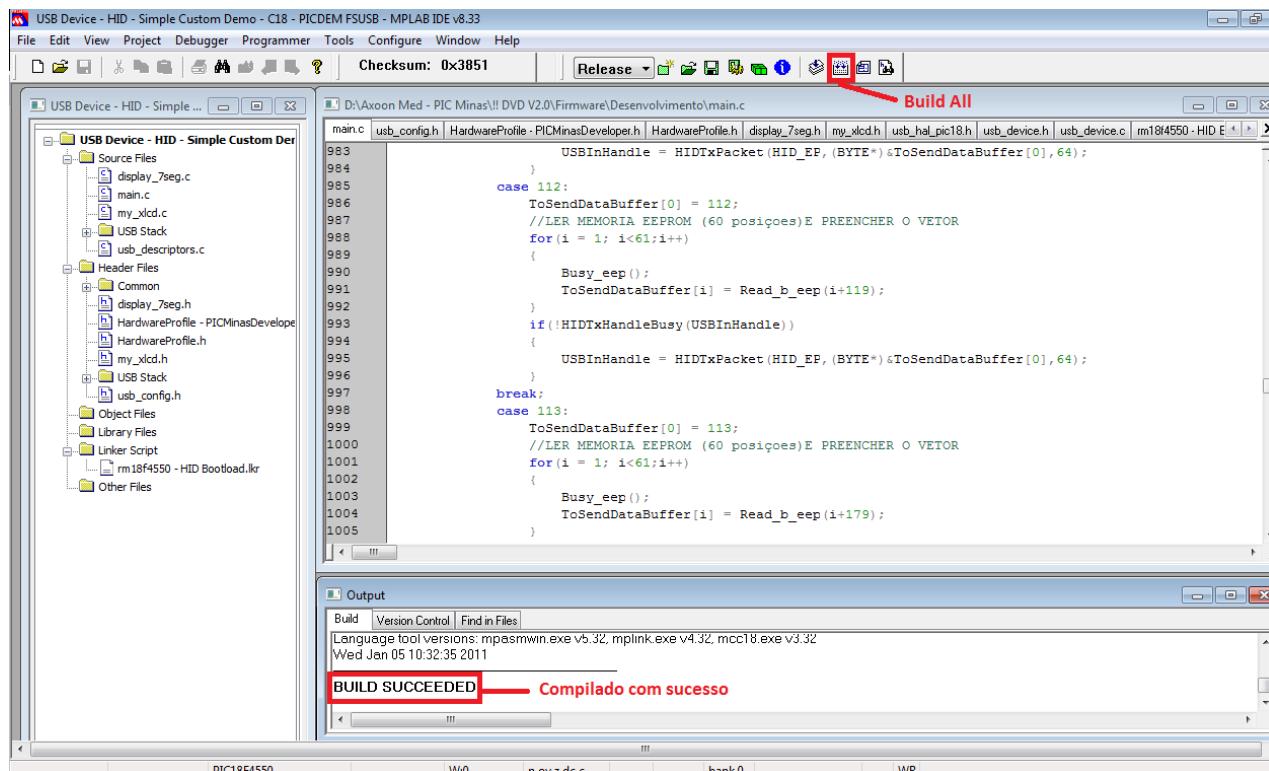


Figura 3.37 - Botão de compilação - Build All.

Após a compilação de seu projeto será gerado um arquivo com o mesmo nome do projeto com extensão **.hex** dentro da pasta do projeto. Esse é o seu firmware compilado. Os nossos KITs Didáticos possuem o Bootloader gravado na memória de programa de seus PICs. Isso permite que ele realize uma Autogravação (ver seção 3.3 desta apostila). A seguir serão mostrados os passos necessários para se carregar um firmware na memória de programa de um PIC utilizando nosso KIT Didático PIC18 e o nosso DVD Didático.

Procedimento para gravação de um firmware via bootloader

- 1) Abra o DVD Didático e clique no ícone “Gravar PIC”. Conecte o seu KIT Didático ao seu computador e coloque-o em modo de gravação conforme mostrado na animação do DVD (segure o botão de BOOT e dê um clique no botão RESET). Observe que os LEDs vermelho e amarelo começarão a piscar. Na caixa de texto “Status” irá aparecer a mensagem “Kit didático conectado”. Observe também que alguns dos botões serão habilitados após a conexão (ver Figura 3.38).

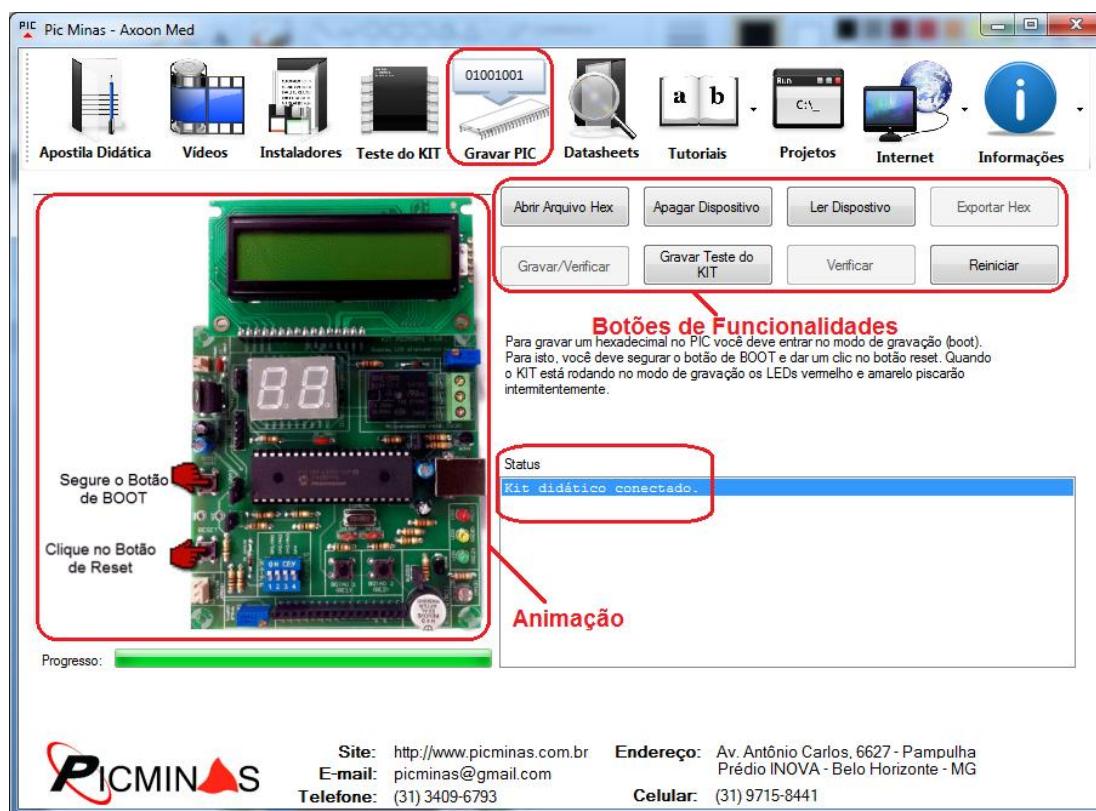


Figura 3.38 - DVD Didático – ícone “Gravar PIC”.

- O próximo passo é informar o diretório onde está localizado o firmware que deseja gravar no PIC. Isso é feito através do botão de funcionalidade “Abrir Arquivo Hex”. Usaremos um projeto exemplo do DVD Didático. Clique no botão **Load HEX File** e vá ao diretório “E:\DVD Didático\Arquivos\Projetos\Botao LED”. Selecione o arquivo hexadecimal (.hex) disponível nessa pasta (neste exemplo o arquivo será **Botão LED.hex**) e, em seguida, clique em “Abrir” (ver Figura 3.39).

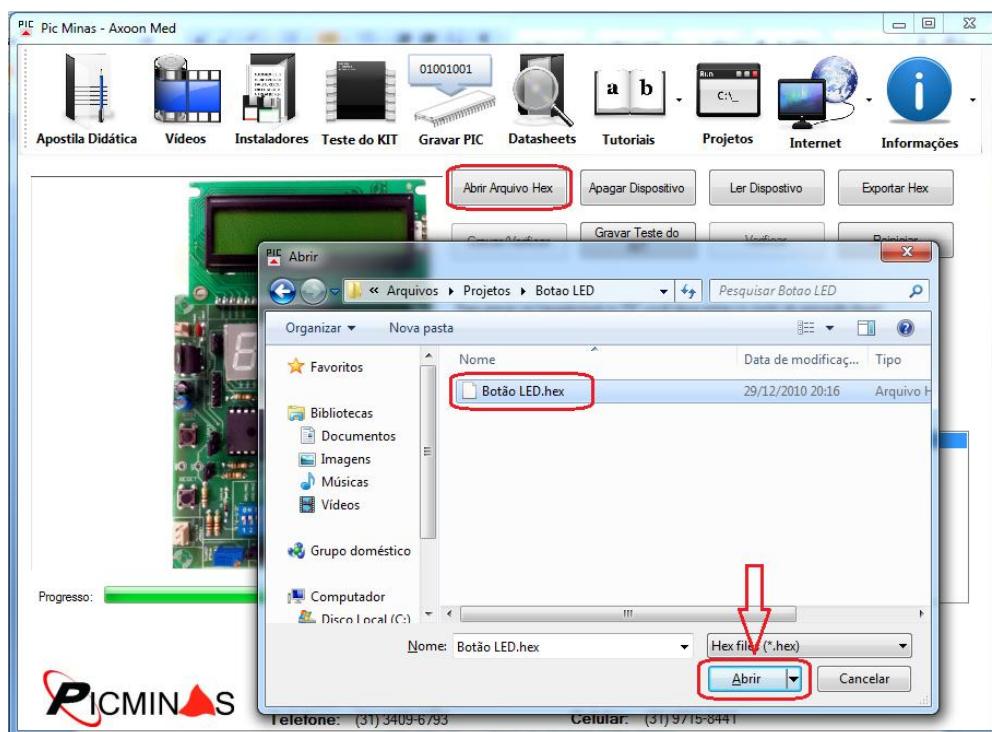


Figura 3.39 - Escolha do arquivo.hex que se deseja gravar no PIC.

- 3) Em seguida clique no botão “**Gravar / Verificar**” e aguarde alguns instantes. Observe que aparecerão algumas mensagens na caixa de texto “**Status**” (ver Figura 3.40).



Figura 3.40 - Processo de gravação do PIC.

- 4) A gravação foi concluída com sucesso. Clique no botão “Reiniciar” do programa ou no botão de reset na placa para verificar o funcionamento do seu programa no KIT.

3.6 COMO GRAVAR UM FIRMWARE NO PIC UTILIZANDO UMA GRAVADORA (ICD2 MICROCHIP)

Além do método de Autogravação, utilizando *bootloader* (visto na seção anterior), existem ainda métodos de gravação que utilizam dispositivos específicos, chamados de GRAVADORAS. Para demonstrar como é feito um processo de gravação utilizando gravadoras, esta seção trás um exemplo de como carregar um código na memória de programa do PIC utilizando a gravadora MPLAB ICD2^{BR} da Microchip (ver Figura 3.41). Neste método não é necessário que o microcontrolador possua o firmware bootloader previamente gravado em sua memória de programa, e ainda permite a realização de depuração de códigos via hardware (emulação).

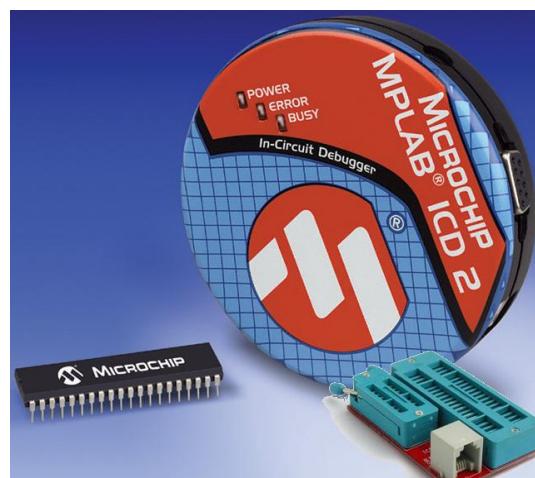


Figura 3.41 - Módulo de gravação ICD2 da Microchip.

Os procedimentos para gravar um firmware utilizando uma gravadora ICD2 são:

Procedimento de Instalação do Driver

- 1) Abra o MPLAB e conecte a gravadora em seu computador.
- 2) Aparecerá uma janela de instalação do *driver* da gravadora. O assistente de instalação do driver pergunte se o Windows pode se conectar ao site do Windows Update para procurar o software, selecione a opção “**Não, não agora**” e clique no botão “**Avançar**” (Figura 3.42).

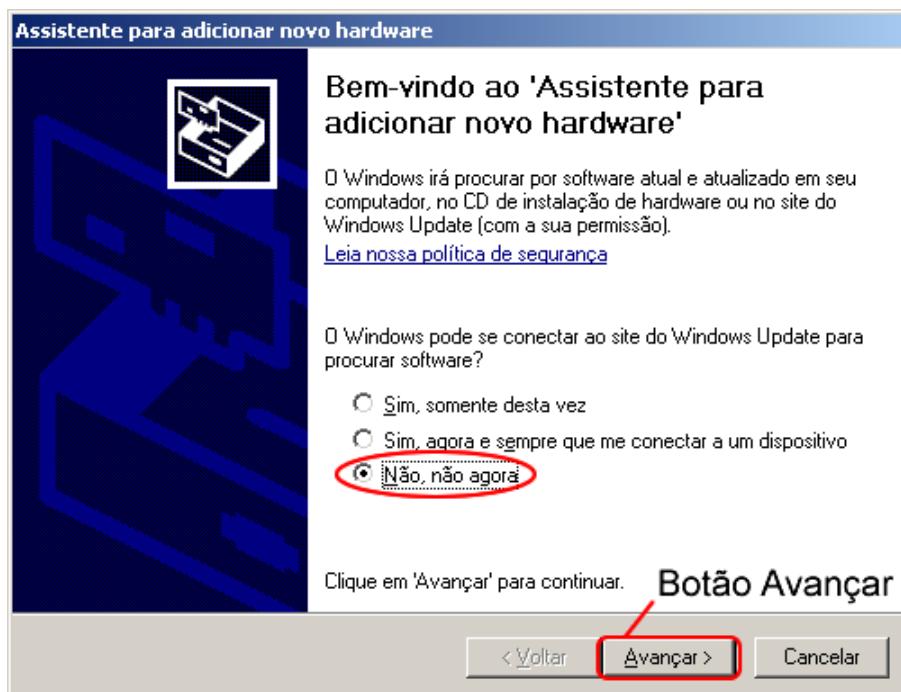


Figura 3.42 - Janela de reconhecimento do dispositivo.

- 3) Na janela seguinte, deixe marcada a opção “**Instalar o software automaticamente (recomendável)**” e clique no botão “**Avançar**” (Figura 3. 43).

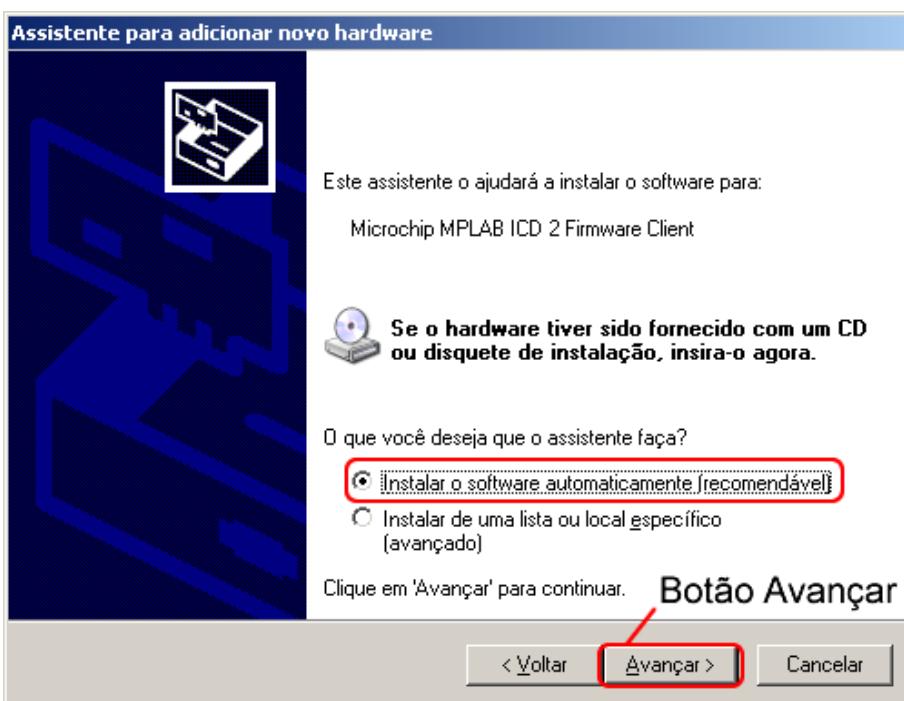


Figura 3. 43 - Janela de instalação do driver.

- 4) Feito isso, espere a instalação do *driver* e, em seguida, clique em “**Concluir**”. (Figura 3.44)

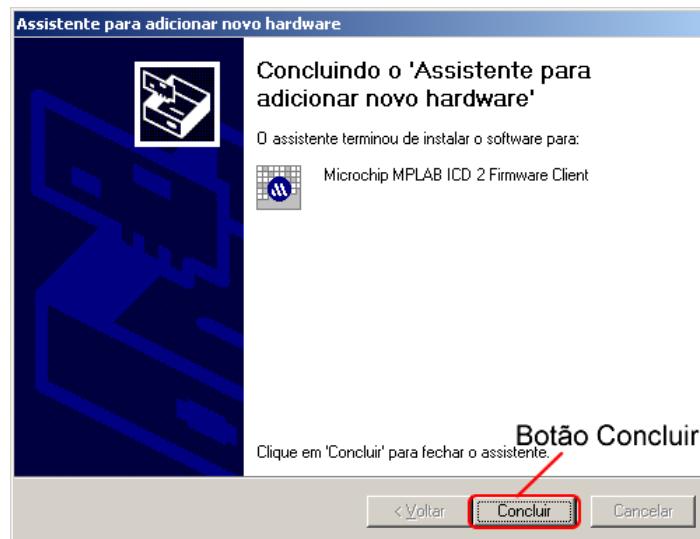


Figura 3.44 - Janela de conclusão.

Observação: Caso você conecte o módulo de gravação em outra porta de comunicação USB, deverá repetir todo o procedimento de instalação do *driver*.

Como exemplo de gravação de um *firmware* no microcontrolador, gravaremos o firmware do *bootloader*. Para a gravação deste *firmware*, siga o procedimento a seguir:

Procedimento de Gravação de um Firmware

- Após abrir o MPLAB, vá ao DVD Didático, clique no ícone “Projetos”. No menu “Projetos”, vá em “Projetos Avançados” e clique sobre o projeto “Bootloader”. Em seguida, clique no botão “Salvar Projeto” e escolha um diretório onde será salvo o projeto e clique “Salvar” (ver Figura 3.45). Retorne ao MPLAB e abra o projeto do bootloader que desejar (por exemplo, o **HID Bootloader PIC18 Non J.mcp** para o PIC18F4550) a partir da pasta que você o salvou.



Figura 3.45 - DVD Didático - Ícone Projetos - Projeto do Bootloader.

- 2) Sem alterar o código, vá ao menu **Programmer** -> **Select Programmer** -> **2 MPLAB ICD2**. (Figura 3.46)

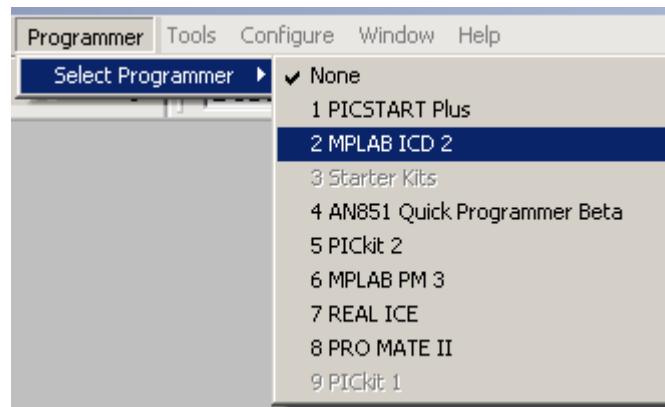


Figura 3.46 - Seleção do módulo de gravação.

- 3) Caso o MPLAB não reconheça o módulo de gravação, exibindo uma mensagem de aviso: “**Auto-connect not enabled - Not connecting (Try enabling auto-connect on the ICD2 settings pages.)**”. Siga os passos abaixo, caso contrário, pule para o passo 10.
- 4) Clique no menu **Programmer** -> **MPLAB ICD2 Setup Wizard**. (Figura 3.47)



Figura 3.47 - Seleção da opção Setup Wizard.

- 5) Na janela de boas vindas (**Welcome**), clique no botão “**Avançar**”. Na janela seguinte, **Communications**, selecione a porta de comunicação USB e clique em **Avançar**. (Figura 3.48)

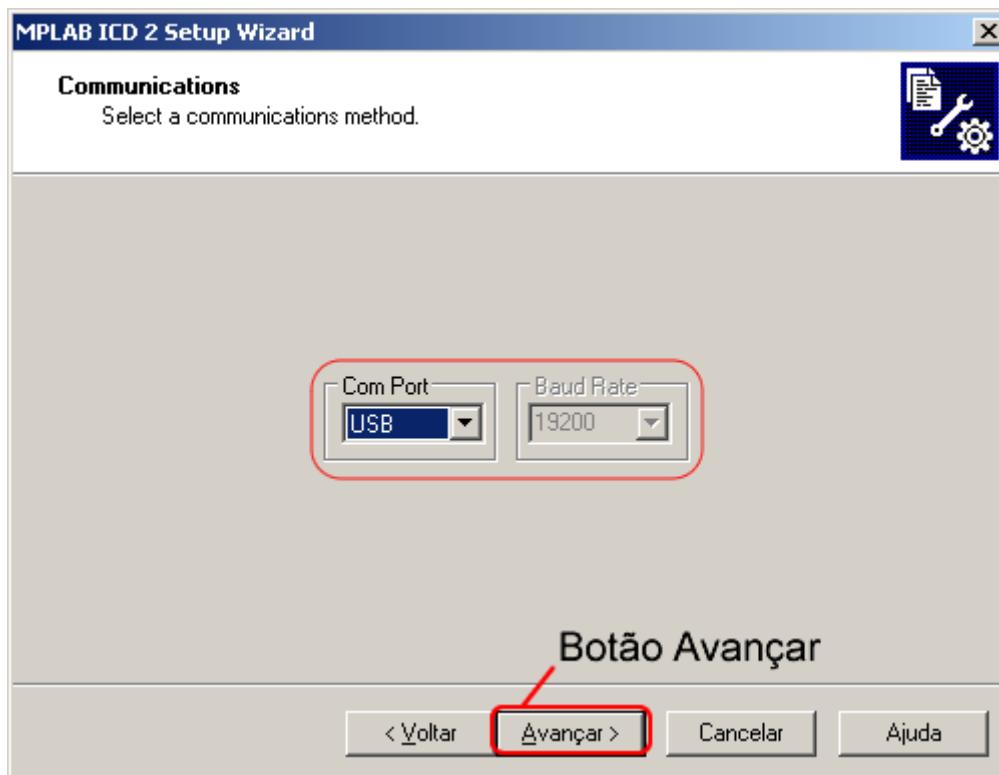


Figura 3.48 - Seleção da Porta de Comunicação.

- 6) Em seguida, aparecerá a janela “Power”, onde é necessário escolher a opção de alimentação do dispositivo. Selecione a opção “Power target from the MPLAB ICD 2”. Clique em “Avançar”. (Figura 3.49)

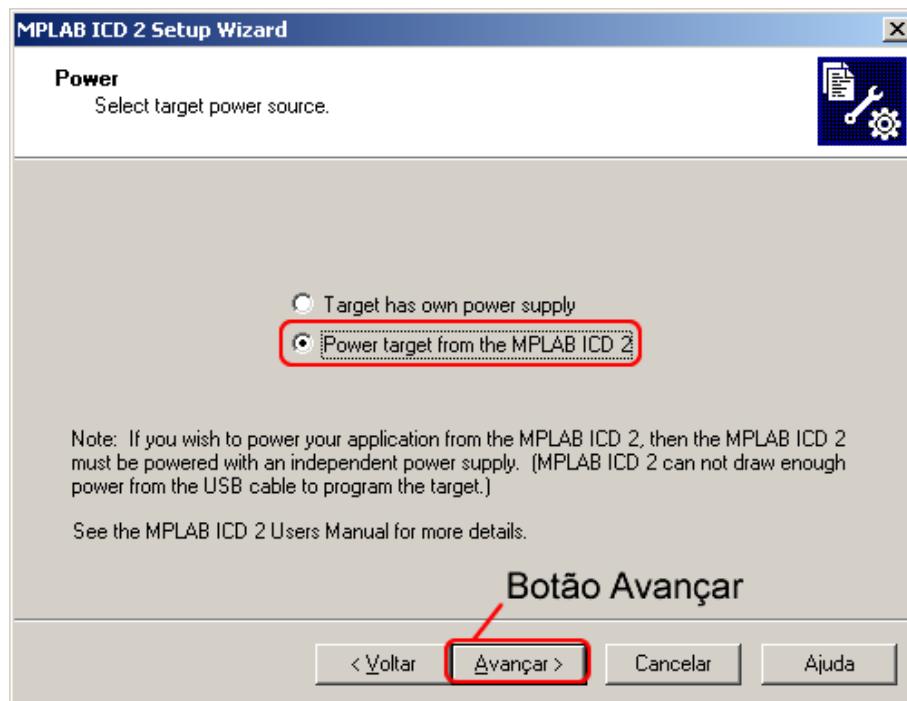


Figura 3.49 - Seleção da alimentação do dispositivo.

- 7) Na janela seguinte, **Connection**, marque a opção “**MPLAB IDE automatically connects to the MPLAB ICD2**”, e clique em “**Avançar**” (ver Figura 3.50).

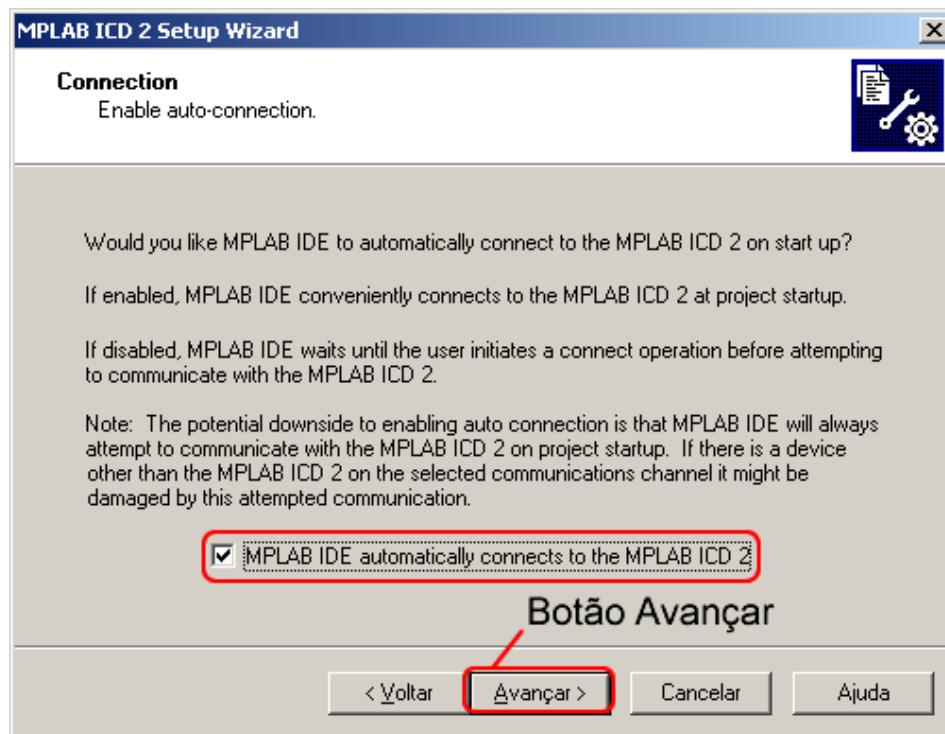


Figura 3.50 - Janela de Conexão.

- 8) No passo seguinte, aparecerá a janela “**Download**”, marque a opção “**MPLAB ICD2 automatically downloads the required operating system**” e, em seguida, clique em “**Avançar**” (ver Figura 3.51).

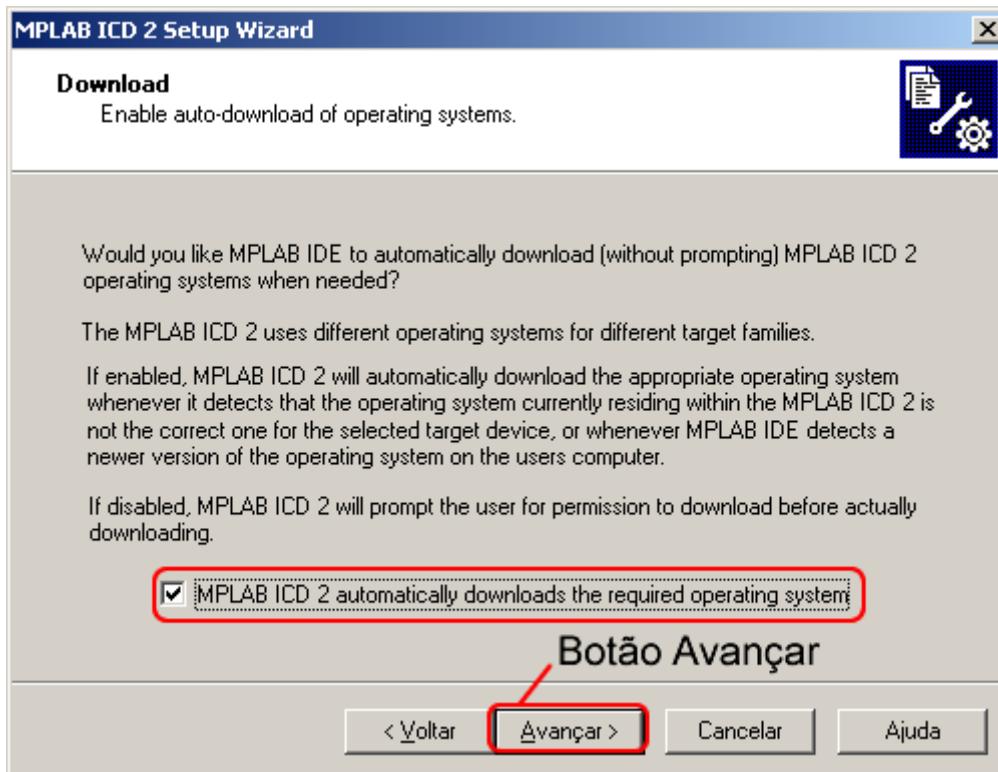


Figura 3.51 - Janela Download.

- 9) Na janela final clique em “**Concluir**”. Repita o passo 2 e verifique se o módulo de gravação foi reconhecido, caso contrário, entre em contato com o fabricante do módulo.
- 10) Quando a gravadora estiver corretamente conectada ao MPLAB, será possível visualizar uma série de ícones habilitados na barra de ferramentas. (Figura 3.52)



Figura 3.52 - Ícones de gravação habilitados.

A Tabela 6 descreva cada um desses ícones:

 Program target device	 Read target device	 Reads device EEPROM
 Verify target device memory	 Erase target device	 Verify target device is erased
 Release from Reset	 Hold in Reset	 Reset and Connect to ICD

Tabela 6 - Descrição dos ícones de gravação.

- 11) Para gravar o *firmware* desejado, clique no botão “**Erase target device**” e, em seguida, clique em “**Program target device**”. É importante apagar o conteúdo do dispositivo antes de realizar a gravação do novo *firmware*.
- 12) O MPLAB exibirá uma mensagem de Programação realizada com sucesso. (ver Figura 3.53)

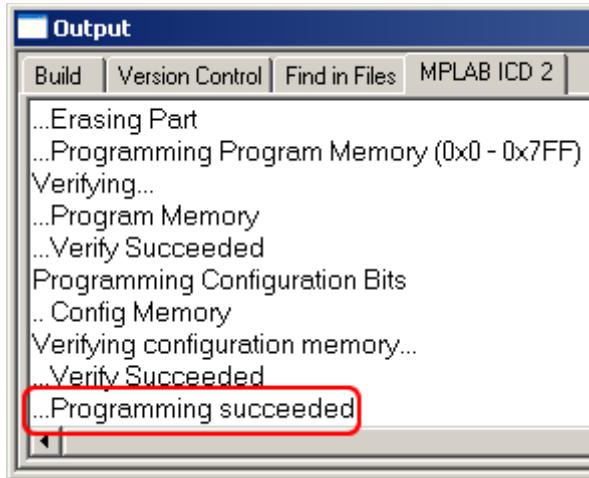


Figura 3.53 - Mensagem informando que a programação foi realizada com sucesso.

3.7 COMO UTILIZAR A FERRAMENTA DE SIMULAÇÃO DO MPLAB

- 1) Você pode acompanhar a simulação do programa criado e verificar o seu funcionando através do “MPLAB SIM”. Trata-se de uma ferramenta muito útil para a identificação e eliminação de erros (*bugs*) em seu código. Para isso clique no menu “**Debugger**” e escolha a opção “**Select Tool**”. Em seguida, selecione a opção “**MPLAB SIM**” (esse é o simulador integrado ao MPLAB). (ver Figura 3.54)

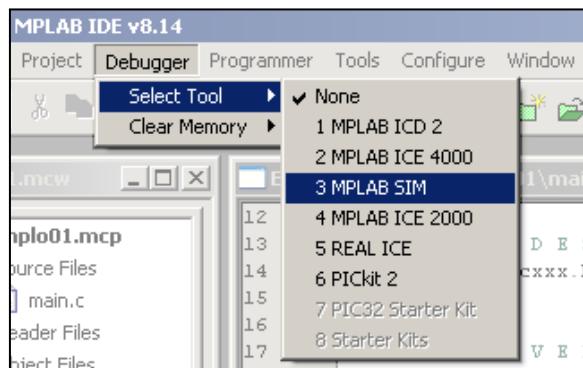


Figura 3.54 - Selecionando o simulador MPLAB.

- 2) Ao selecionar o simulador irá aparecer uma série de ícones adicionais na barra de ferramentas do MPLAB (ver Figura 3.55).



Figura 3.55 - Botões do simulador MPLAB.

A Tabela 1 mostra, suscintamente, a função de cada um dos botões da ferramenta de simulação e suas respectivas teclas de atalho:

Menu de debugação	Botões na barra de ferramentas	Tecla de Atalho
Run (Inicia a simulação)	▶	F9
Halt (Pausa a simulação)		F5
Animate (Simulação com animação dos SFR)	▶▶	
Step Into (Executar Passo-a-Passo)	↷	F7
Step Over (Pular uma sub-rotina)	↷↷	F8
Step Out (Sair uma sub-rotina)	↶↷	
Reset (Reiniciar)	⟲	F6

Tabela 7 - Função de cada botão.

- 3) Aparecerá uma aba MPLAB SIM na janela **Output**. (ver Figura 3.56)

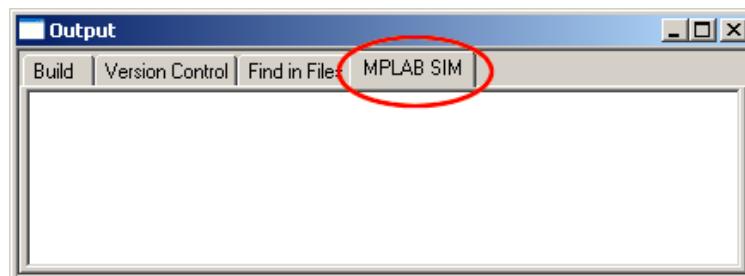


Figura 3.56 - Aba do MPLAB SIM na janela Output.

- 4) A barra de *status* indicará que o MPLAB está em modo de Simulação.



Figura 3.57 - Barra de Status indicando MPLAB SIM.

- 5) Além de ser possível acompanhar a execução de seu código passo-a-passo, através do MPLAB SIM, é possível acompanhar os valores das variáveis criadas pelo usuário e também dos SFRs (Special Function Registers – Registradores com funções especiais). Clique no menu “View” e escolha a opção “Watch” (ver Figura 3.58). Será possível visualizar o conteúdo das variáveis ou dos SFRs. (ver Figura 3.59)

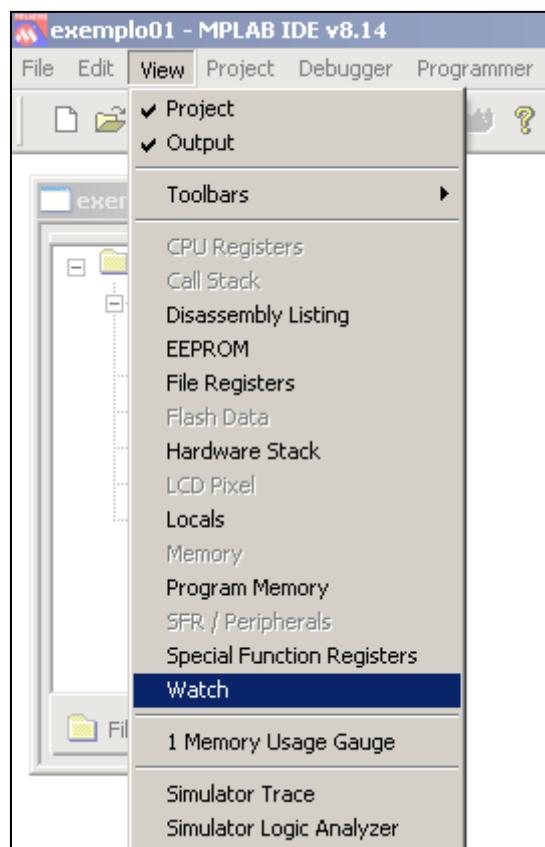


Figura 3.58 - Opção Watch.



Figura 3.59 - Janela Watch.

A variável será apresentada na base hexadecimal, pois este é o formato padrão da opção “Watch”. Para alterar a base numérica clique com o botão direito sobre a variável e selecione “Properties...”, na janela que irá se abrir selecione no campo “Format” a opção que desejar (ver Figura 3.60).

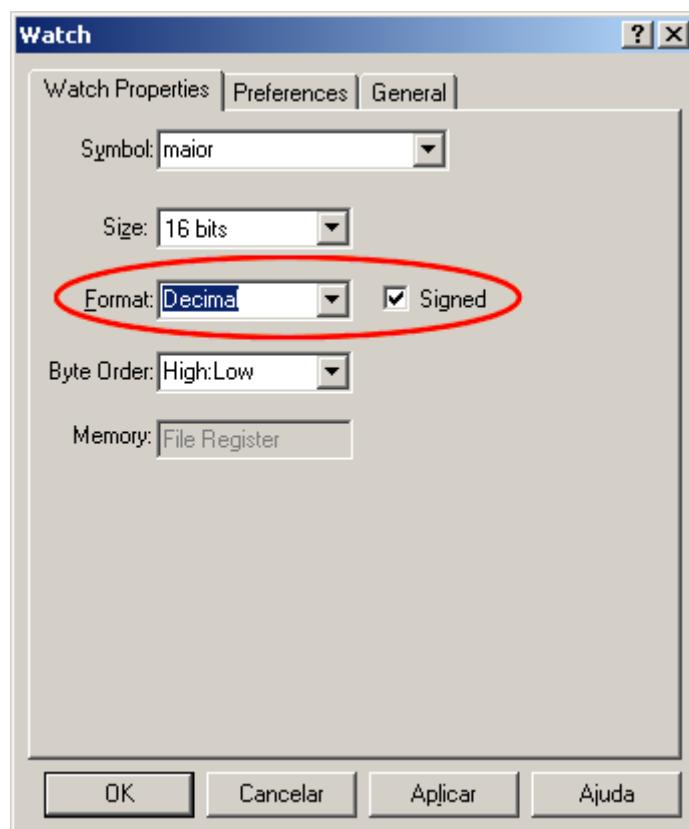
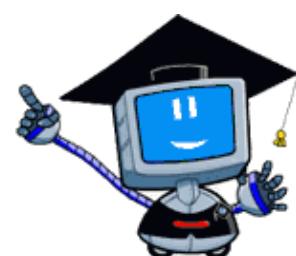


Figura 3.60 - Alterando para exibição em decimal.

- 6) Por último, é possível colocar pontos de parada (*Breakpoints*) em seu código. Para isso, clique com o botão direito no ponto que deseja adicionar uma parada e selecionar a opção **Set Breakpoint**, para remover é só clicar com o botão direito na linha que contém o ponto de parada e selecionar a opção **Remove Set Breakpoint**.

Neste capítulo, falamos sobre:

- Ferramentas para desenvolvimento de *firmware* na linguagem C;
- Procedimentos de instalação dos mesmos;
- Procedimentos de gravação;
- Criação, compilação e simulação de projetos.



Capítulo 4 – Programação de Microcontroladores

Os primeiros dispositivos programáveis utilizavam linguagem de máquina, que consiste em dígitos binários (zeros e uns) executados pelo dispositivo. Essa programação é extremamente complexa e requer muito tempo para o desenvolvimento da aplicação.

Para facilitar a programação dos sistemas microprocessados foi criada a linguagem *Assembly* (baixo nível). Esta linguagem substituiu as codificações binárias por comandos pré-definidos de acordo com o processador utilizado. Isto facilitou a programação destes dispositivos. No entanto, a linguagem de programação *Assembly* também apresenta algumas limitações, como por exemplo, possuir baixa portabilidade, pois existe um tipo de *Assembly* para cada tipo de microprocessador.

A linguagem C foi criada para aproximar a programação da linguagem humana (linguagem de alto nível). Utilizando compiladores adequados, a linguagem C é portável, ou seja, pode ser executada em máquinas com arquiteturas diferentes.

A alta eficiência de compilação dos programas para a linguagem de máquina a transforma em uma das linguagens mais utilizadas atualmente para desenvolvimento de *firmwares* para microcontroladores. Esta eficiência é devido a sua proximidade com a linguagem *Assembly*, reduzindo o tamanho dos programas e, consequentemente, seu tempo de processamento.

A escolha da linguagem de programação C é natural, pois a maioria dos microcontroladores disponíveis no mercado dispõem de compiladores para linguagem C. Assim, com a utilização de uma linguagem de alto nível como C, o programador foca-se em solucionar os problemas da aplicação abstraindo-se das tarefas de baixo nível como localização das variáveis, operações matemáticas e lógicas, verificação de bancos de memória, controle de pilha e alocação de memória, entre outras.

Com o tempo os recursos disponíveis nos microcontroladores, como memórias e capacidade de processamento, permitiram que os mesmos pudessem ser programados em linguagem de alto nível, como por exemplo, a linguagem de programação C.

4.1. VANTAGENS E DESVANTAGENS DE SE PROGRAMAR EM ASSEMBLY E EM C:

	Vantagens	Desvantagens
Linguagem C	Linguagem mais amigável	Pode ocupar um espaço de memória desnecessário, dependendo da otimização do compilador
	Rápido desenvolvimento de aplicações	
	Fácil manutenção do código	
	Fácil acesso aos recursos de Hardware	Uma determinada tarefa pode não ser realizada da maneira mais eficiente
	Linguagem bastante difundida	
	Compiladores atuais tem bom desempenho	
	Vários Exemplos implementados na Web	
Linguagem Assembly	Possibilidade de otimização do processamento	Aumenta o tempo de desenvolvimento de novos projetos;
	Controle total do hardware programado	Dificulta a manutenção de projetos antigos
	Economia de espaço e velocidade de memória	Exige projetistas experientes e bem treinados;
		Depuração complexa

Tabela 8 - Vantagens e desvantagens entre linguagem Assemplby e C.

4.2 PRINCÍPIOS DE PROGRAMAÇÃO

O processo de programação pode ser dividido em duas etapas, na primeira elaboramos um algoritmo capaz de solucionar o problema. Na segunda, implementamos este algoritmo em linguagem de programação.

Para elaborar um bom algoritmo é necessário conhecer alguns conceitos básicos de programação, como linguagem estruturada, fluxogramas, variáveis, dados e operadores.

Uma abordagem amplamente utilizada é a divisão de um programa em partes menores (modularização). Para a codificação de cada parte podemos seguir um conjunto de procedimentos genéricos, listados abaixo:

- 1º) Exposição do problema:** descrever detalhadamente o problema a ser resolvido;
- 2º) Analise e esquematização da solução:** descrição sequencial da solução que melhor resolve o problema. (Montagem do algoritmo)
- 3º) Tradução do código:** tradução do algoritmo em comandos que serão corretamente interpretados pela linguagem de programação utilizada.
- 4º) Depuração (em inglês DEBUG):** é o processo de verificação e teste do programa de forma a localizar e solucionar todas as eventuais falhas e erros de codificação que tenham acontecido em quaisquer fases anteriores.

Para facilitar a execução das tarefas listadas e achar uma solução são usados diagramas e fluxogramas, que permitem uma melhor visualização do problema.

4.2.1. ALGORITMOS ESTRUTURADOS

A Programação Estruturada é uma metodologia de construção de algoritmo (programas) para proporcionar:

- Maior facilidade de escrita (elaboração);
- Maior facilidade de leitura (compreensão);
- Permitir o aproveitamento de partes dos programas (reuso);
- Facilitar a manutenção e modificação dos programas (manutenção).

A ideia básica da Programação Estruturada é permitir, ao programador, redução da complexidade dos problemas propostos em três níveis:

- 1) **Desenvolvimento Top-Down:** antes de escrever um programa, é necessário um processo de raciocínio que leve a uma análise do problema, passando por um algoritmo em termos gerais até um algoritmo mais detalhado. As diferentes fases desse processo de concepção de programa são fixadas por escrito. Cada nova fase é obtida por refinamento da fase anterior, até chegar a um nível de detalhe que permita implementar o algoritmo, diretamente na máquina, em linguagem de programação.
- 2) **Modularização:** a solução do problema, durante o processo descrito acima, vai sendo dividida em soluções de subproblemas, permitindo a divisão do programa em forma natural, com subfunções claramente definidas, podendo ser implementadas separadamente por diversos programadores. A modularização também permite o reaproveitamento do código (ou solução) em outros problemas.
- 3) **Estrutura de Controle:** programar cada módulo com as estruturas básicas de controle: sequência simples, condicional e comando repetitivo, que correspondem à sequência natural de execução, evitando o uso de desvios.

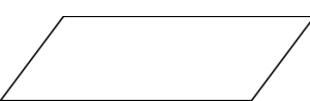
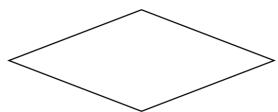
A técnica do desenvolvimento estruturado de algoritmos, por si só, não atinge automaticamente os objetivos visados. Ela apenas preconiza uma maneira sistemática que ajuda a atingir estes objetivos

A prática de programar, por mais simples que seja o programa, é muito importante, para assim ter a experiência e aperfeiçoamento na busca de um programa simples e claro.

4.2.2. FLUXOGRAMAS

Fluxogramas são elementos gráficos utilizados para estabelecer a sequência de operações necessárias para o cumprimento de determinada tarefa e consequentemente a resolução de um problema. Eles permitem esquematizar e visualizar os sistemas de forma racional, clara e concisa, facilitando seu entendimento geral por todos os envolvidos.

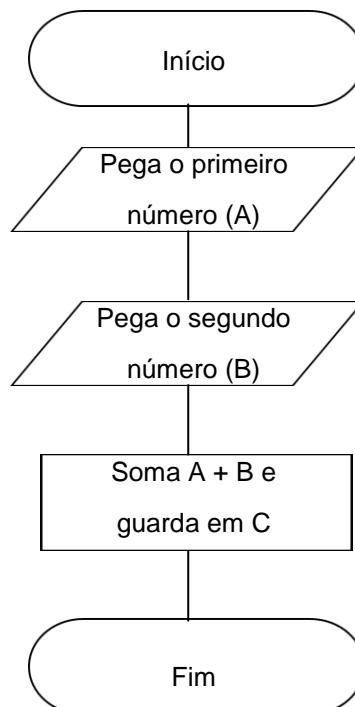
Alguns elementos básicos para a construção de um fluxograma estão listados abaixo:

Símbolo	Descrição
	Início ou Terminação: este tipo de símbolo é utilizado para representar o início ou término do programa ou algoritmo.
	Processamento: este símbolo é utilizado para descrever uma determinada tarefa.
	Dados: normalmente este símbolo é utilizado para descrever entrada e saída de dados no sistema.
	Tomada de decisão: este símbolo é utilizado para representar um ponto de tomada de decisão, ou teste condicional. A tomada de decisão pode conduzir sempre a um resultado: verdadeiro ou falso.

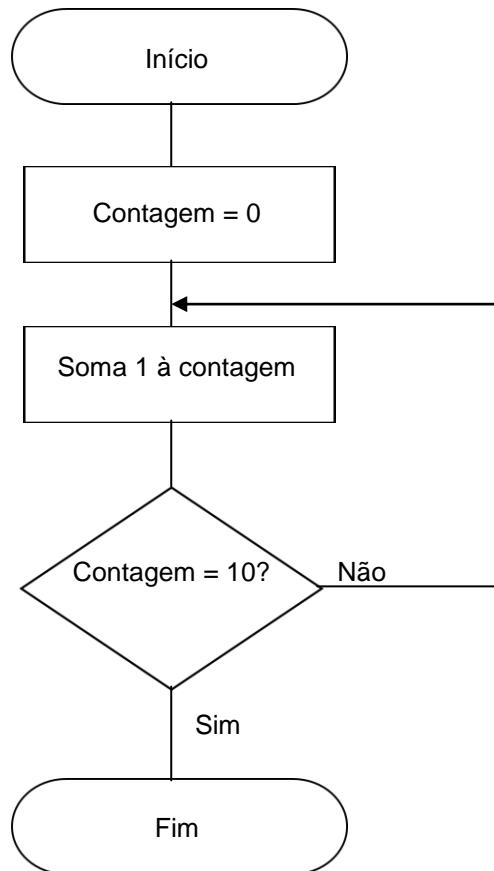
Os símbolos utilizados nos fluxogramas têm por objetivo evidenciar origem, processo e destino de uma informação de um sistema.

Um fluxograma fornece uma compreensão detalhada de um processo e procura apresentar o processo passo a passo.

O primeiro exemplo mostra um problema simples de somar dois números (A e B) e armazenar o resultado em C.



Outro exemplo, um programa para contar de 0 até 10



4.2.3. VARIÁVEIS E DADOS

Na programação, uma **variável** é um objeto (uma posição, frequentemente localizada na memória) capaz de reter e representar um valor ou expressão. Enquanto as variáveis só "existem" em tempo de execução, elas são associadas a "nomes", chamados identificadores, durante o tempo de desenvolvimento.

Sempre que necessitamos armazenar algum tipo de dado, seja ele proveniente do mundo exterior (uma tecla pressionada, uma tensão lida externamente) como do próprio programa (o resultado de uma equação) será utilizada uma variável.

As variáveis ficam localizadas em um espaço na memória de dados (volátil) do dispositivo. Durante o processo de execução de um programa, uma variável poderá assumir diversos valores diferentes, mas nunca dois ou mais valores ao mesmo tempo. A classificação de uma variável depende do conteúdo que elas armazenam: numéricas, ponto flutuante, caractere, alfanuméricos e lógicos.

A quantidade de memória utilizada para armazenar uma variável depende do seu tipo: variáveis inteiras são normalmente encontradas com tamanhos de 16 bits. Uma variável de 8 bits pode armazenar um valor inteiro entre 0 e 255, uma variável de 16 bits pode armazenar um valor inteiro entre 0 e 65535 e uma variável de 32 bits pode armazenar um valor inteiro entre 0 e 4 294 967 295. Observe que os números representados anteriormente são somente positivos, no caso de representação de números com sinal, a magnitude de representação reduz-se à metade daquela sem sinal.

Variáveis do tipo real ou ponto flutuante são normalmente encontradas em tamanho de 32 bits. É importante saber que num sistema de 8 bits, uma variável de 16 bits vai ocupar duas posições de memória.

OPERADORES

Sempre que precisamos relacionar ou modificar um ou mais elementos como variáveis ou dados de um programa, utilizamos operadores. Eles são elementos ou símbolos gráficos utilizados então para relacionar ou modificar dados ou variáveis.

São cinco categorias de operadores:

- **Matemáticos:** utilizados para efetuar determinada operação matemática em relação a um ou mais dados. (Adição, subtração e outras);
- **Relacionais:** utilizados para relacionar dois elementos ou dados. (Maior, menor, igual);
- **Lógicos:** utilizados para efetuar operações lógicas booleanas entre dois ou mais elementos ou dados. Este tipo de operação somente pode chegar a um dos resultados: verdadeiro (1) ou falso (0), e são frequentemente utilizados na criação de testes condicionais com múltiplas variáveis. (E lógico, OU lógico, NÃO lógico);
- **Lógicos bit a bit:** utilizados para realizar operações lógicas bit a bit entre um ou mais elementos ou dados.
- **Memória:** são operadores utilizados para efetuar operações em relação à memória do dispositivo. (Atribuição (=), & e *)

4.3 CARACTERÍSTICAS DO COMPILADOR C18

4.3.1. TIPOS DE DADOS E LIMITES

TIPO INTEIRO

O compilador MPLAB C18 suporta os tipos de dados inteiros definidos pelo padrão ANSI. A faixa dos padrões inteiros está documentada na Tabela 9:

Type	Size	Minimum	Maximum
char	8 bits	-128	127
signed char	8 bits	-128	127
unsigned char	8 bits	0	255
int	16 bits	-32,768	32,767
unsigned int	16 bits	0	65,535
short	16 bits	-32,768	32,767
unsigned short	16 bits	0	65,535
short long	24 bits	-8,388,608	8,388,607
unsigned short long	24 bits	0	16,777,215
long	32 bits	-2,147,483,648	2,147,483,647
unsigned long	32 bits	0	4,294,967,295

Tabela 9 - Tipos de dados inteiros e seus limites.

TIPO PONTO-FLUTUANTE

Os dados com ponto-flutuante de 32 bits também são suportado através dos tipos float e double. No caso do compilador MPLAB C18, esses dois tipos possuem a mesma magnitude. A faixa do ponto-flutuante está mostrada na tabela 6:

Type	Size	Minimum Exponent	Maximum Exponent	Minimum Normalized	Maximum Normalized
float	32 bits	-126	128	$2^{-126} \approx 1.17549435e - 38$	$2^{128} * (2 - 2^{-15}) \approx 6.80564693e + 38$
double	32 bits	-126	128	$2^{-126} \approx 1.17549435e - 38$	$2^{128} * (2 - 2^{-15}) \approx 6.80564693e + 38$

Tabela 6: Tipos de dados ponto-flutuante e seus limites.

MODIFICADORES DE TIPO

A linguagem C admite comandos especiais para modificar os tipos básicos apresentados. Através desses modificadores podemos obter outros tipos de dados. Esses modificadores são: **signed**, **unsigned**, **short** e **long**.

O modificador **signed** é utilizado para modificar um tipo de dado básico para que ele possa representar números com sinais.

O modificador **unsigned** define um tipo sem sinal permitindo representações de números de maior magnitude.

O modificador **short** é utilizado para definir uma variável com tamanho menor que o tipo modificado, ou seja, uma versão reduzida do tipo especificado. Assim, se especificarmos uma variável como sendo do tipo **short int**, ela será uma versão reduzida do tipo **int**

O modificador **long**, utilizado para ampliar a magnitude de representação do tipo especificado. Um tipo de dados **long int**, por exemplo, terá um tamanho de 32 bits, ou seja, ocupará mais espaço na memória de dados do PIC.

TIPO DE ARMAZENAMENTO DE DADOS – ENDIANCESS

Endianness refere-se ao método utilizado para ordenar os Bytes de uma palavra binária multi-Bytes. Essa ordenação pode ser no formato *Big-endian*, onde os Bytes menos significativos são armazenados nos maiores endereços de memória, ou no formato *Little-endian*, onde os Bytes menos significativos são alocados nas menores posições de memória. O MPLAB C18 armazena os dados em formato *Little-endian*, conforme mostrado na Figura 4.1.

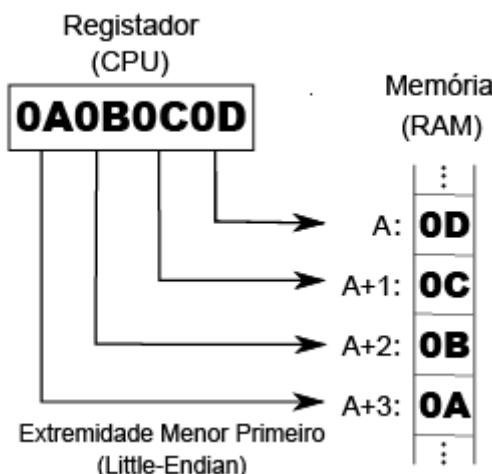


Figura 4.1 - Ordenação Little-Endian.

MODIFICADORES DE ARMAZENAMENTO

Os modificadores de armazenamento são elementos utilizados para controlar a forma como o compilador irá lidar com o armazenamento da variável.

MPLAB C18 suporta os modificadores de armazenamento ANSI (auto, extern, register, static e overlay), alguns deles descritos a seguir:

AUTO

É utilizado para definir o âmbito ou escopo da variável local ou função/classe definida. Não é necessário escrever o modificador, pois ele é usado como padrão na linguagem C.

EXTERN

É utilizado para definir variável ou funções/classes externas ao programa. Isto é muito utilizado na criação de programas grandes e complexos, utilizando diversos módulos separados e que são depois ligados por meio de um programa especial chamado *linker*.

STATIC

As variáveis ou funções/classes declaradas como **static** funcionam como globais no sentido que não serão destruídas ao término da execução. **Static** é utilizado quando necessitamos manter uma variável ou função/classe em uma posição permanente na memória. O parâmetro *static* é válido apenas quando o compilador está operando em modo Non-extended. O padrão de classe de uma função é *auto*. Isso pode ser modificado explicitamente com a palavra *static* antes da função.

4.4. CAMINHO DE PROCURA DE ARQUIVOS

ARQUIVOS DE SISTEMAS HEADER

Arquivos objetos específicos de uma determinada biblioteca são incluídos automaticamente em seu código quanto utilizado #include <filename>. Esses arquivos são procurados em caminho especificado no ambiente de variáveis MCC_INCLUDE.

ARQUIVOS DE USUÁRIO HEADER

Arquivos fontes (Source files) incluídos no código com #include “filename” são procurados no caminho especificado pelo usuário, onde está contido o arquivo.

4.5 LINGUAGEM DE PROGRAMAÇÃO C

A linguagem C foi criada na década de 70 por Dennis Ritchie. O C é derivado de outra linguagem, o B, criado por Ken Thompson. O B, por sua vez, se originou da linguagem BCPL, inventada por Martin Richards.

A linguagem de programação C é utilizada para a criação dos mais variados programas, como por exemplo, processadores de texto, planilhas eletrônicas, programas de comunicação, sistemas operacionais, programas para a automação industrial, gerenciadores de bancos de dados, programas de projeto assistido por computador, programas para a solução de problemas de Engenharia, Física, Química, entre outros.

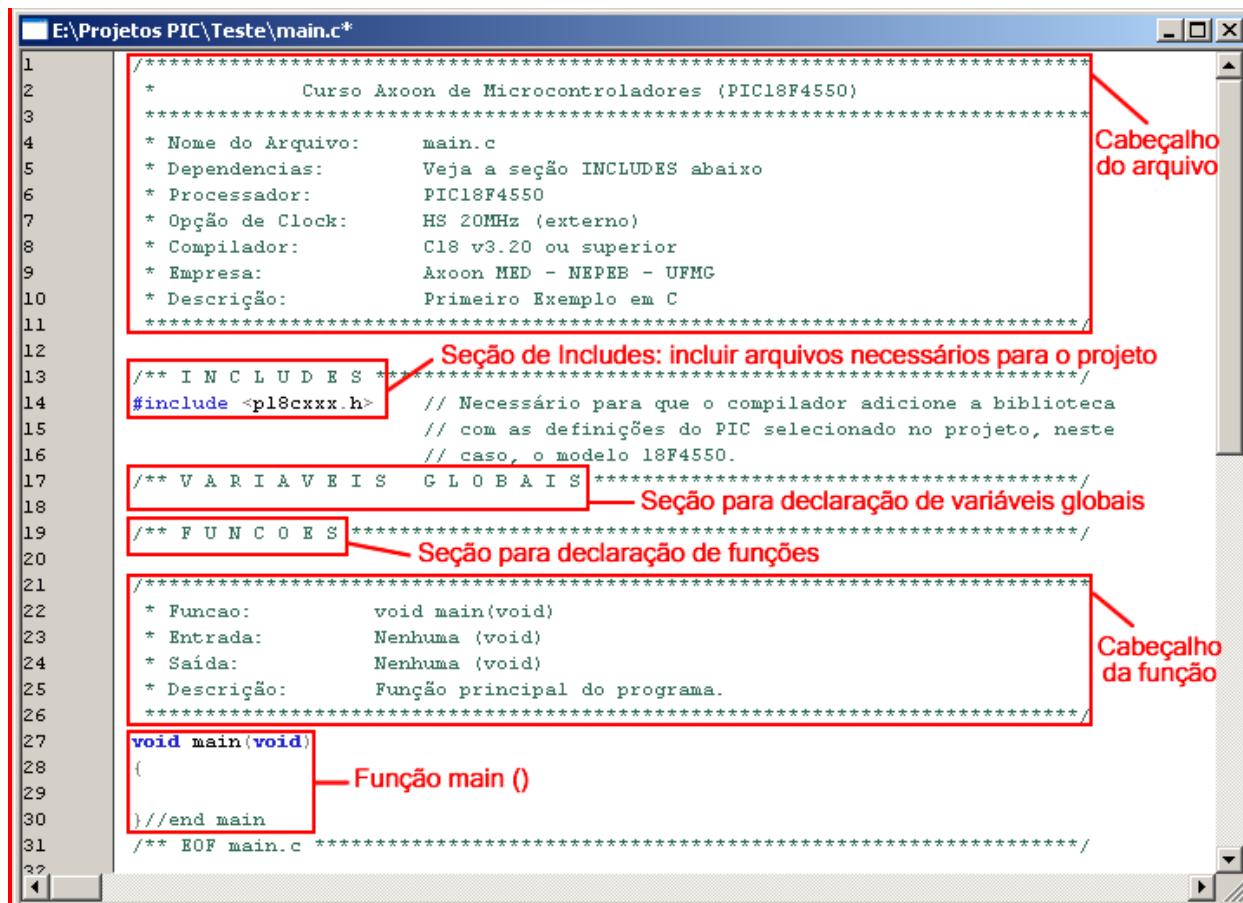
A capacidade de programar transcende o conhecimento de estruturas, funções e recursos de uma linguagem de programação. É necessário que o programador adquira familiaridade com o compilador,

experiência na identificação de "bugs" e desenvolva um bom raciocínio lógico. Portanto, é importante que o leitor escreva seus próprios códigos (evitando cópias na íntegra de outros programas), compile e execute todas as práticas apresentadas.

4.5.1 PRIMEIROS PASSOS

Para facilitar o estudo, iniciaremos com uma visão geral sobre cada elemento que compõe o código de um programa C através de um exemplo simples.

De acordo com o que foi visto na secção três, crie um projeto no MPLAB IDE. O arquivo main.c deve possuir o conteúdo mostrado na Figura 4.2. Para isso, utilize o modelo do arquivo que está no DVD do curso, na aba **Projetos**, no tópico **Arquivos modelos** no link **PIC18 – main e linker**.



```

1  **** Curso Axoon de Microcontroladores (PIC18F4550)
2  * Nome do Arquivo: main.c
3  * Dependencias: Veja a seção INCLUDES abaixo
4  * Processador: PIC18F4550
5  * Opção de Clock: HS 20MHz (externo)
6  * Compilador: C18 v3.20 ou superior
7  * Empresa: Axoon MED - NEPEB - UFMG
8  * Descrição: Primeiro Exemplo em C
9
10 ****/
11
12 /**
13  * INCLUSES ****
14 #include <p18cxx.h> // Necessário para que o compilador adicione a biblioteca
15 // com as definições do PIC selecionado no projeto, neste
16 // caso, o modelo 18F4550.
17 /**
18  * V A R I A V E I S   G L O B A I S ****
19 /**
20  * F U N C O E S ****
21 /**
22  * Funcao: void main(void)
23  * Entrada: Nenhuma (void)
24  * Saída: Nenhuma (void)
25  * Descrição: Função principal do programa.
26 /**
27 void main(void)
28 {
29     // Função main ()
30 } //end main
31 /**
32 EOF main.c ****

```

Figura 4.2 - Conteúdo do arquivo main.c.

Para ter certeza de que o código está funcionando compile-o.

Você pode compilar o código por duas maneiras:

- ✓ Clicando no menu **Project** e escolhendo a opção **Build All (Ctrl+11)**;
- ✓ Ou na barra de ferramentas clicando no botão .

O conteúdo do arquivo criado é dividido em algumas seções descritas abaixo:

- ⊕ A primeira seção contém o **CABEÇALHO** do arquivo: Composto apenas por **comentários** onde o desenvolvedor descreve o projeto com nome do arquivo, o microcontrolador para qual o projeto está sendo desenvolvido, o compilador utilizado para transformar o código escrito em C em linguagem binária (linguagem de máquina) e um breve resumo das funcionalidades do mesmo.
- ⊕ **Comentários** podem ser iniciados em qualquer ponto de uma linha e são usados para descrever o funcionamento ao final de cada código, ou realizar qualquer observação que o

programador julgar pertinente. Os comentários são ignorados pelo compilador, possuindo apenas a função de facilitar a compreensão do código de um programa.

Os comentários podem ser:

- Uma linha apenas, quando usamos duas barras //;
 - Ou de múltiplas linhas, quando usamos /* (para iniciar o comentário) e */ (para finalizar o comentário).
- ⊕ A segunda seção contém os **INCLUDES**: serão incluídos todos os arquivos de **header** (.h) necessários para o funcionamento do projeto. No nosso exemplo incluímos o arquivo p18cxx.h (disponível no endereço C:\MCC18\h). Esse arquivo contém os includes de todos os cabeçalhos dos microcontroladores PIC da família 18, inclusive o PIC18F4550.h.
- O código **#include <p18cxx.h>** é uma diretiva do compilador. Neste caso ele está determinando ao compilador que anexe ao programa o arquivo especificado: **p18cxx.h**.
- ⊕ A terceira seção contém as **VARIÁVEIS GLOBAIS**: são declaradas as variáveis globais utilizadas pela(s) função(ões) do sistema.
- ⊕ A quarta seção contém as **FUNÇÕES**: é onde são escritos a função **main()** e outras funções que serão chamadas pela **main()**.
Uma função, em C, é um conjunto de instruções que, uma vez definidas, podem ser executadas a partir de qualquer ponto do programa, quantas vezes forem necessárias.

Repare que antes de qualquer função declarada existe um cabeçalho com nome, entrada, saída e descrição da função.

Caso não haja nenhum erro no código a janela de Output aparecerá a mensagem **BUILD SUCCEEDED** (ver Figura 4.3). Caso contrário, verifique o erro e corrigia-o. Um programador não apenas deve saber como programar, mas também identificar e corrigir os erros de um código.

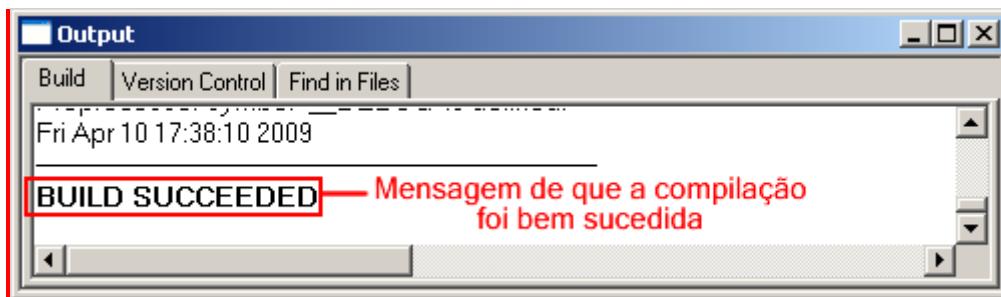


Figura 4.3 - Mensagem de compilação bem sucedida.

. A função **main()** tem um significado especial nos programas em C, pois é a função que é inicialmente executada (em inglês **entry point**) e é utilizada para definir a função principal, ou o corpo principal do programa (ver Figura 4.4).

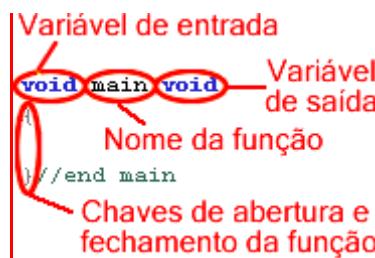


Figura 4.4 - Detalhamento da função main () .

4.5.2 VISÃO GERAL DE UM PROGRAMA

```
#include <pl8cxxx.h>

void main(void)
{
    int maior, menor;      // declara duas variáveis do tipo inteiro
    menor=100;             // atribui o valor 100 à variável menor
    maior=0;               // atribui o valor 0 à variável maior
    while (maior < menor)
    {
        maior=maior+1;    // soma um ao valor de maior
        menor=menor-1;    // subtrai um ao valor de menor
    }

}//end main
```

Figura 4.5 - Exemplo de um programa C.

- a. Na linha **void main(void)** a declaração **main ()** especifica a função principal do programa.
 - b. O caractere abertura de chave “{“ é utilizado para delimitar o início de uma função, e o caractere de fechamento de chave ”}” indica o final da função. As chaves delimitam o que chamamos de bloco de programa, ou bloco de código.
 - c. A primeira linha dentro do bloco de comandos é uma declaração de duas variáveis do tipo inteiro.
 - d. As duas linhas seguintes são atribuições de valores as essas duas variáveis criadas.
 - e. Note que ao final de cada linha de comando o último caractere é um ponto-e-vírgula (;), elemento usado para delimitar o final de um comando.
 - f. Na linha seguinte, encontramos um **while (maior < menor)**. Esse é um comando de controle, utilizado para repetição de um determinado bloco de instruções. Um **while** fica em loop enquanto a condição for satisfeita (no exemplo enquanto a variável **maior** tiver um valor menor que a variável **menor**).
 - g. Em seguida, têm-se dois comandos: o primeiro é para somar uma unidade à variável maior e o outro é para diminuir uma unidade a variável menor.
-
- 7) Você pode simular o funcionamento do programa como foi visto na seção três, e verificar se está funcionando corretamente.
 - 8) Além de observar a simulação passo-a-passo, através do MPLA SIM, é possível acompanhar os valores das variáveis

4.5.3 PALAVRAS RESERVADAS

Toda linguagem de programação possui um conjunto de palavras que não podem ser usadas para declarar funções ou variáveis. São chamadas de palavras reservadas e só podem ser usadas em seus propósitos originais.

Apresentamos a seguir, na Tabela 10, as palavras reservadas do ANSI C:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Tabela 10 - Palavras reservadas em C.

4.5.4 IDENTIFICADORES

O C é “Case Sensitive”, isto é, diferencia letras maiúsculas e minúsculas. Por exemplo: uma variável com nome “**soma**” é diferente de uma variável com nome “**Soma**”, ou “**sOma**”. O mesmo ocorre com os comandos em C, o comando “**if**”, por exemplo, deve ser escrito com letras minúsculas, caso seja escrito com letras maiúsculas o compilador poderá interpretá-lo como variável.

Identificadores são utilizados pelos programadores para nomear variáveis funções e outros elementos da linguagem C. Identificadores podem conter letras, números e o caractere sublinhado “_”, nenhum outro caractere ou acento são permitidos. Além disso, um identificador deve ser iniciado por letras ou sublinhado. Identificadores de até 32 caracteres são permitidos.

Dicas quanto aos nomes de identificadores (variáveis):

- ✓ É uma prática tradicional do C, usar letras minúsculas para nomes de variáveis e maiúsculas para nomes de constantes. Isto facilita na hora da leitura do código;
- ✓ Quando se escreve código usando nomes de variáveis em português, evitam-se possíveis conflitos com nomes de rotinas encontrados nas diversas bibliotecas, que são em sua maioria absoluta, palavras em inglês.

A Tabela 11 mostra alguns exemplos de identificadores válidos e inválidos:

Válido	Variavel	variavel1	_teste_2
Inválido	Variável	1abc	Return

Tabela 11 - Exemplos de variáveis válidas e inválidas.

4.5.5 VARIÁVEIS E TIPOS DE DADOS

A linguagem C disponibiliza ao programador diversos tipos de dados. Os tipos de dados são identificados pelas palavras reservadas da linguagem: **char**, **int**, **double**, **float** e **void** (ver seção 4.3.1).

Antes de usar uma variável é necessário declará-la. Ou seja, informar ao compilador o nome e o tipo de dado que ela armazenará na memória de dados. Como boa prática de programação, é recomendável utilizar nomes que façam referência ao conteúdo ou função da variável. Por exemplo, utilizar o nome “temperatura” ou “tempo” para uma variável que armazena um número.

A sintaxe para declarar uma variável é:

Tipo da variável lista_de_variáveis;

Por exemplo:

```
Int    temperatura; // declara uma variável (temperatura) do tipo inteiro
char   senha1, senha2; // declara duas variáveis (senha1, senha2) do tipo caractere
float  tensao;        // declara uma variável (tensao) do tipo float
```

Podemos também declarar e inicializá-la com um conteúdo, por exemplo:

```
int    temperatura=25; // declara uma variável (temperatura) do tipo inteiro com o valor 25
```

```
char senha1= 'b', senha2 = 0x1F; // declara duas variáveis (senha1, senha2) do tipo caractere
```

Outro aspecto importante é local de declaração de uma variável, pois determina a acessibilidade dessa variável às diversas partes de um programa.

Quanto à acessibilidade, existem três tipos básicos de variáveis:

- ✓ **Variáveis Locais:** são aquelas declaradas em um bloco (delimitado por {}), por exemplo: em uma função. São visíveis e acessadas apenas no bloco em que estão declaradas;
- ✓ **Variáveis Globais:** são aquelas declaradas no início do código. Podem ser visualizadas e acessadas em qualquer parte do programa.
- ✓ **Variáveis Parâmetro Formal:** são aquelas passadas como parâmetros de funções.

4.5.6 CONSTANTES

Constantes são valores, numéricos ou não, que não podem ser alterados pelo programa durante sua execução.

A sintaxe para declarar uma constante é:

```
const nome_da_constante = valor;
```

Por exemplo:

```
const COTADOR = 10           // declara uma constante CONTADOR com valor 10
```

```
const PI = 3.1415            // declara uma constante PI com valor 3.1415
```

```
const LCD_LINHA1 = "PIC Minas" // declara uma constante LCD_LINHA1 com valor PIC Minas
```

De maneira geral, qualquer tipo de dado pode ser utilizado para definir uma constante.

CÓDIGOS DE BARRA INVERTIDA

A linguagem C utiliza vários códigos chamados de códigos de barra invertida. Esses códigos permitem ao programador inserir alguns caracteres do ASCII que não podem ser inseridos diretamente em uma constante *string*.

São diversos os motivos que impedem a inserção de determinados caracteres, mas, como exemplo, podemos citar o retorno de carro (que é aquele inserido pela tecla ENTER) e o caractere de aspas ‘ “ ’: O primeiro não pode ser inserido diretamente pelo teclado, pois o retorno de carro é inserido no código-fonte, provocando um efeito indesejado. O segundo, mesmo estando facilmente disponível no teclado, não pode ser inserido diretamente na string porque o compilador irá interpretá-lo como sinal do término da *string*.

Na Tabela 12 encontram-se os códigos mais comumente usados:

Código	Significado
\b	Retrocesso ("back")
\f	Avanço de formulário ("form feed")
\n	Nova linha ("new line")
\t	Tabulação horizontal ("tab")
\"	Aspas
\'	Apóstrofo
\0	Nulo (0 em decimal)
\\\	Barra invertida "\\"
\v	Tabulação vertical
\r	Retorno de carro (Enter)

Tabela 12 - Código de barra invertida.

CONSTANTES BINÁRIAS, HEXADECIMAIS E OCTAIS

É importante o conhecimento de bases numéricas diferentes da decimal para a programação de microcontroladores. A linguagem C possui a capacidade de operar com dados numéricos nas bases binária, octal, decimal e hexadecimal. Para utilizar valores com bases numéricas diferentes da decimal, utilizamos os códigos na Tabela 13

Valor	Base numérica
27	Decimal
011	Octal
0xA9	Hexadecimal
0b10001001	Binário

Tabela 13 - Bases numéricas.

Exemplos:

```
int valor1;  
valor1 = 50;           // decimal  
valor1 = 0x32;         // hexadecimal  
valor1 = 062;          // octal  
valor1 = 0b110010;     // binário
```

4.5.7 OPERADORES

A linguagem C possui uma série de operadores, que podem ser classificados em: atribuição, aritméticos, relacionais, lógicos, lógicos bit a bit, de memória e outros.

ATRIBUIÇÃO

Em C o operador de atribuição é o “=”, usado para atribuir um determinado valor a uma variável. Por exemplo:

```
x = 10;    // a variável x recebe o valor 10  
y = x;    // a variável y recebe o valor da variável x
```

Observe que a variável que receberá o valor se posiciona a esquerda do sinal de atribuição, enquanto o valor, ou a variável que contém o valor, se posiciona à direita do sinal de atribuição.

ARITMÉTICOS

São utilizados para indicar ao compilador a operação matemática que deverá ser efetuada entre um ou mais dados. São eles (Tabela 14):

Operador	Ação
+	Adição
-	Subtração ou menos unário
*	Multiplicação
/	Divisão
%	Resto da divisão inteira
++	Incremento
--	Decremento

Tabela 14 - Operadores aritméticos.

Os operadores aritméticos incremento e decrecimento servem para somar 1 e subtrair 1 de uma variável, respectivamente.

A forma geral para utilização destes dois operadores é:

variável `++;` ou variável `--;`

RELACIONAIS

São utilizados em testes condicionais para determinar a relação existente entre variáveis. Esses operadores servem para construir expressões condicionais. São eles (Tabela 15):

Operador	Ação
<code>></code>	Maior que
<code>>=</code>	Maior ou igual a
<code><</code>	Menor que
<code><=</code>	Menor ou igual a
<code>==</code>	Igual a
<code>!=</code>	Diferente

Tabela 15 - Operadores relacionais.

LÓGICOS BOOLEANOS

São utilizados para realizar associações lógicas entre elementos e podem somente resultar em um dos valores: verdadeiro ou falso. Eles são de suma importância na construção de testes condicionais. São eles (Tabela 16):

Operador	Ação
<code>&&</code>	AND (E)
<code> </code>	OR (OU)
<code>!</code>	NOT (NÃO)

Tabela 16 - Operadores lógicos booleanos.

LÓGICOS BIT A BIT

São utilizados para realizar operações lógicas entre elementos ou variáveis. No entanto, ao contrário dos operadores lógicos booleanos, os operadores lógicos bit a bit podem resultar em um valor com uma mesma quantidade de bits dos elementos operados. São eles (Tabela 17):

Operador	Ação
<code>&</code>	AND (E)
<code> </code>	OR (OU)
<code>^</code>	XOR (OU exclusivo)
<code>~</code>	NOT (complemento de um)
<code>>></code>	Deslocamento à direita
<code><<</code>	Deslocamento à esquerda

Tabela 17 - Operadores lógicos bit a bit.

Vale à pena destacar os dois últimos operadores, deslocamento à direita e deslocamento à esquerda. O primeiro faz os bits deslocarem uma posição para a direita e o segundo faz os bits deslocarem uma posição para a esquerda.

O exemplo abaixo ilustra o uso dos operadores de deslocamento:

```
unsigned char c;  
  
c = 7;           /* código binário 0000 0111 */  
c = c<<1;       /* código binário 0000 1110 = 14 */  
c = c<<2;       /* código binário 0011 1000 = 56 */  
c = c<<3;       /* código binário 1100 0000 = 192 */  
c = c>>1`;      /* código binário 0110 0000 = 96 */
```

PONTEIROS

Ponteiros permitem ao programador acessar diretamente a memória do sistema. Por isso, são considerados elementos de grande importância na linguagem C.

Os operadores de ponteiros estão na Tabela 18:

Operador	Ação
&	Endereço do operando
*	Conteúdo do endereço apontado pelo operando

Tabela 18 - Operadores de memória.

O operador unário **&** é utilizado para retornar o endereço de memória de uma variável. Ou seja, se escrevermos:

```
endereco_a = &a;
```

Teremos que a variável “endereco_a” conterá o endereço em que está armazenada a variável “a”.

O operador unário ***** é utilizado para retornar o conteúdo da posição de memória endereçada pelo operando que o segue. Vejamos o exemplo:

```
a = *endereco_a;
```

Este exemplo fará com que o valor armazenado no local apontado pela variável “endereco_a” seja atribuído à variável “a”.

OUTROS OPERADORES

Além de todos esses operadores citados anteriormente, existem outros não tão conhecidos em C. São eles (Tabela 19):

Operador	Ação
?	Operador ternário condicional
,	Separador de expressões
.	Separador de estruturas
->	Ponteiro de elemento de estrutura
(tipo)	Operador de modelagem de dado

Tabela 19 - Outros operadores.

O operador ternário “?” é utilizado para substituir uma expressão condicional baseada no comando IF.

Sua sintaxe é:

```
Variável = Expressão1? Expressão2; Expressão3;
```

O que significa: avalie a expressão 1 se ela for verdadeira atribua à variável a expressão 2; caso contrário, atribua a expressão 3. Por exemplo:

```
int x, y;           // declaração de duas variáveis do tipo inteiro
x = 5;             // atribui o valor 5 a variável x
y = x==7? 10; x+3; // se x é igual a 7, y recebe o valor 10, senão y recebe o valor x+3
```

ASSOCIAÇÃO DE OPERADORES

A linguagem C admite associar operadores e montar expressões abreviadas equivalentes a expressão original. Na Tabela 20, podemos encontrar os tipos de abreviações admitidas em C.

Expressão Original	Expressão Equivalente
x=x+k;	x+=k;
x=x-k;	x-=k;
x=x*k;	x*=k;
x=x/k;	x/=k;
x=x&k;	x&=k;
x=x k;	x =k;
x=x^k;	x^=k;
x=x>>k;	x>>=k;
x=x<<k;	x<<=k;
x=x&k;	x&=k;

Tabela 20 - Associação de operadores.

PRECEDÊNCIA DOS OPERADORES

A Tabela 21 mostra a precedência dos operadores:

Operador	Associatividade
() [] -> .	esq-dir
! ~ ++ -- - (type) * & sizeof	dir-esq
* / %	esq-dir
+ -	esq-dir
<< >>	esq-dir
< <= > >=	esq-dir
== !=	esq-dir
&	esq-dir
^	esq-dir
	esq-dir
&&	esq-dir
	esq-dir
? :	dir-esq
= += -= etc.	dir-esq
,	esq-dir

Tabela 21 - Precedência de operadores.

4.5.8 COMANDOS E FUNÇÕES IMPORTANTES

COMANDO IF

O comando **if** (em português “se”) é utilizado em teste condicionais, sua sintaxe é:

```
if (condições)
{
    comandos;
}
```

As condições, serão avaliadas, se forem verdadeiras os comandos especificados serão executados. Caso contrário não executará os comandos. Estas condições podem ser quaisquer expressões que possam ser avaliadas como verdadeiro ou falso.

O if pode ser complementado com comando **else** e sua sintaxe fica:

```
if (condições)
{
    comandos_1;
}
else
{
    comandos_2;
}
```

Caso as condições sejam verdadeiras, executarão os `comandos_1`, caso contrário executarão os `comandos_2`.

EXEMPLO 01

```
void main(void)
{
    int val1, val2, aux;      // declara três variáveis do tipo inteiro
    val1=100;                // atribui o valor 100 à variável menor
    val2=0;                  // atribui o valor 0 à variável maior
    if (val1 >= val2)
    {
        aux = val1;
        val1 = val2;
        val2 = aux;
    }
    else
    {
        aux = val2;
        val2 = val1;
        val1 = aux;
    }

} //end main
```

Figura 4.6 - Uso das estruturas if else.

COMANDO SWITCH

O comando switch é usado para tomada de decisão através de comparação de uma variável com diversos valores.

Sintaxe do **switch**:

```
switch (variável)
{
    case constante_1:
        comando_1;
        break;
    case constante_2:
        comando_2;
        break;
    .
    .
    .
    case constante_n:
        comando_n;
        break;
    default
        comando_default;
}
```

O comando **switch** aceita apenas valores constantes na condição **case**. Se a variável e a constante possuem o mesmo valor, então os comandos dessa condição serão executados. Sempre após a sequência de comandos da condição **case** é necessário usar o comando **break**. Se não for utilizado, todos os comandos seguintes ao **case** especificado serão executados até que encontre uma cláusula **break** ou o final do bloco **switch**.

EXEMPLO 02

```
void main(void)
{
    int variavel, numero; // declara uma variável do tipo inteiro
    variavel = 3; // atribui o valor 3 a variável
    switch (variavel)
    {
        case 0:
            numero=0;
            break;
        case 1:
            numero=1;
            break;
        case 2:
            numero=2;
            break;
        case 3:
            numero=3;
            break;
    } //end switch
} //end main
```

Figura 4.7 - Uso da estrutura switch.

COMANDO WHILE

O comando **while** (em português “enquanto”) é usado para especificar uma estrutura ou laço de repetição em que um ou mais comandos são repetidamente executados **enquanto** a condição de validação for verdadeira. Sua sintaxe é:

```
while (condição)
{
    comando;
}
```

Após a execução dos comandos dentro do **while** a condição é testada novamente. O loop é executado até o momento em que a condição se torna falsa. Esta condição pode ser qualquer expressão que possa assumir o valor verdadeiro ou falso (expressão booleana). É possível fazer um loop infinito, para tanto basta colocar uma expressão eternamente verdadeira na condição.

EXEMPLO 03

```
void main(void)
{
    int val1, val2, contador;      // declara três variáveis do tipo inteiro
    val1=1;                      // atribui o valor 1 à variável val1
    val2=1;                      // atribui o valor 1 à variável val2
    contador=2;                  // atribui o valor 2 à variável contador
    while (contador <=20)         // faça enquanto contador for menor que 20
    {
        val1=val1+val2;          // atualiza o valor de val1
        val2=val1+val2;          // atualiza o valor de val2
        contador++;
    }
}//end main
```

Figura 4.8 - Uso da estrutura while.

COMANDO FOR

O laço **for** é utilizado para repetir um comando ou um bloco de comandos por diversas vezes. A sintaxe do for é:

```
for (inicialização ; condição ; incremento)
{
    comando1;
    comando2;
    ...
}
```

Podemos ver, que o **for** executa a inicialização e testa a condição. Se a condição for falsa o programa desvia para o fim do comando for. Se a condição for verdadeira ele executa o comando ou o bloco de comandos dentro dele, faz o incremento e, novamente, testa a condição. Ou seja, o comando for repete seus comandos até que a condição se torne falsa.

Cada um dos três elementos do comando **for** possui uma função distinta conforme descrita abaixo:

- **Inicialização:** Esta seção conterá uma expressão para inicialização da variável de controle do laço **for**.
- **Condição:** Esta seção pode conter a condição a ser avaliada para decidir pela continuidade ou não do laço de repetição. Enquanto a condição for avaliada como verdadeira, o laço **for** permanecerá em execução.
- **Incremento:** Esta seção pode conter uma ou mais declarações para incremento da variável de controle do laço. É importante comentar que a variável de controle do comando **for** pode ser incrementada de qualquer valor, podendo, inclusive, ser decrementada.

EXEMPLO 05

```
void main(void)
{
    int i, fatorial, numero;      // declara três variáveis do tipo inteiro.
    numero=5;                    // atribui o valor 5 à variável numero.
    fatorial=1;                  // atribui o valor 1 à variável fatorial.
    for (i=1; i<=numero; i++)   // i inicia com o valor 1 e vai até o
    {                           // valor do número incrementando de um e um.
        fatorial=fatorial*i;   // calcula o fatorial
    } //for
} //end main
```

Figura 4.9 - Uso da estrutura **for**.

EXEMPLO 06

```
void main(void)
{
    int i, fatorial, numero;      // declara três variáveis do tipo inteiro.
    numero=5;                    // atribui o valor 5 à variável numero.
    fatorial=1;                  // atribui o valor 1 à variável fatorial.
    for (i=numero; i>=1; i--)   // i inicia com o valor 1 e vai até o
    {                           // valor do número incrementando de um e um.
        fatorial=fatorial*i;   // calcula o fatorial
    } //for
} //end main
```

Figura 4.10 - Uso da estrutura **for** com decremento de índice.

COMANDO DO-WHILE

. O comando **do while** (em português “faça enquanto”) é usado para criar uma estrutura de repetição com funcionamento um pouco diferente do comando **while**.

O **do-while**, avalia a condição de teste ao final de cada ciclo, garantindo que o bloco de comandos será executado pelo menos uma vez.

. Sua sintaxe é:

```
do
{
    comandos;
} while (condição);
```

EXEMPLO 08

```
void main(void)
{
    int val1, val2, contador;      // declara três variáveis do tipo inteiro
    val1=1;                      // atribui o valor 1 à variável val1
    val2=1;                      // atribui o valor 1 à variável val2
    contador=2;                  // atribui o valor 2 à variável contador
    do                            // faça
    {
        val1=val1+val2;          // atualiza o valor de val1
        val2=val1+val2;          // atualiza o valor de val2
        contador++;              // incrementa o valor de contador
    }while (contador <=20);       // enquanto contador for menor que 20
}//end main
```

Figura 4.11 - Uso da estrutura do-while.

4.6 BOAS PRÁTICAS DE PROGRAMAÇÃO

Esta seção contem um apanhado de preceitos, não obrigatórias, que entraram em consenso ao longo do tempo entre programadores mais experientes. Tem como objetivo tornar os códigos mais claros, reduzindo os riscos de *bugs* (erros).

É importante adotar um padrão e sempre segui-lo em seus projetos. Apesar de a maior parte dessas dicas serem aplicáveis para a maioria das linguagens de programação, esta seção faz uso exemplos apenas em linguagem C.

Preceitos:

1- Espaços em branco:

O compilador ignora espaços em branco no seu código, por isto você pode usá-los com generosidade. Utilize o recuo para refletir a estrutura de blocos do programa e separe com quebra de linhas partes diferentes do seu código. Caso uma linha de código seja muito grande quebre-a em duas ou mais linhas. Se estiver chamando uma função que recebe muitos parâmetros ou que os parâmetros são máscaras de bits, definidos anteriormente, quebre-os em linhas e comente seu código. A Figura 4.12 trás um exemplo que faz uso das dicas mencionadas anteriormente.

```
// Configura o CAD por meio dos registradores ADCON0, ADCON1 e ADCON2.
OpenADC(
    //Parâmetro Config
    ADC_FOSC_64 &           // ADC_FOSC_64:          Clock de conversão do A/D igual a
                            //                                FAD = FOSC/64 = 48MHz/64 = 750kHz
                            //                                Desta Forma, TAD=1/FAD = 1,33us.
    ADC_RIGHT JUST &         // ADC_RIGHT JUST:       Resultado da conversão ocupará os
                            //                                bits menos significativos dos regis-
                            //                                tradores ADRESH e ADRESL.
    ADC_2_TAD,                // ADC_2_TAD:            Determina o tempo de aquisição do CAD,
                            //                                o tempo para carregar o Chold, onde
                            //                                Taqc_min=2,45us. Deve-se escolher um
                            //                                múltiplo de TAD que atenda esse valor,
                            //                                no nosso caso será igual a 2*TAD
                            //                                2*TAD = 2*1,33us = 2,66us.
    //Parâmetro Config2
    ADC_CH5 &               // ADC_CH4:              Atua sobre os bits CHS4:CHS0 do ADCON0
                            //                                para selecionar o canal no qual será
                            //                                realizada a conversão. Neste caso o AN3.
    ADC_INT_OFF &             // ADC_INT_OFF:          Habilita a interrupção de término de
                            //                                conversão.
    ADC_REF_VDD_VSS,           // ADC_REF_VDD_VSS:      Determina as tensões de refe-
                            //                                rência do CAD:
                            //                                VREF+ = VDD (+5V)
                            //                                VREF- = VSS (0V)
    //Parâmetro PortConfig
    ADC_6ANA );               // ADC_5ANA:              Configura os pinos AN4(RA5), AN3(RA3), AN2(RA2), AN1(RA1)
                            //                                e AN0(RA0) como Entradas Analógicas
```

Figura 4.12 - Exemplo do uso de espaços e comentários.

2- Comentários:

Assim como os espaços em branco, comentários também são ignorados pelo compilador. Use comentários de bloco, `/* */`, para criar cabeçalhos de arquivos (ver Figura 4.13), cabeçalhos de funções, explicar o funcionamento de um trecho de código muito complicado, entre outros (ver Figura 4.12).

Utilize comentários de linha `//` para descrever algum funcionamento específico. Cuidado: evite comentários desnecessários como explicar o que faz um comando de C, afinal comentar de mais complica o código. Lembre-se de atualizar sempre os comentários quando o código sofrer atualizações, comentários que contradizem o código são destrutivos.

```
*****
*
*          Curso PICMINAS de Microcontroladores (PIC18F4550)
*
*****
* Nome do Arquivo:      main.c
* Dependencias:         Veja a seção INCLUDES abaixo
* Processador:          PIC18F4550
* Opcão de Clock:       HS 20MHz (externo) - CPU:48MHz
* Compilador:           C18 v3.20 ou superior
* Empresa:              PICMINAS - Axoon - UFMG
*
* Plataforma:           Kit PICMINAS v3.0
* Prática:              Prática 4.0 - Interrupção Timer0
* Descrição:            Usar o Timer0 para ser a base de tempo do cronometro simples, que faz
*                       contagens de 0,0 a 9,9 segundos e as apresenta no display de 7 segmentos.
*                       Utilize a interrupção do Timer0 para fazer os incrementos do contador
*                       a cada 100ms. A interrupção do Timer0 deve ser tratada dentro da ISR
*                       (rotina de tratamento de interrupção) do vetor de alta prioridade.
*                       Como o valor da contagem deve ser apresentado no display 7 segmentos,
*                       pode-se utilizar a biblioteca de usuário "display_7seg.h".
*                       O cronometro deve usar dois botões com a seguinte funcionalidade:
*                           - Botão 1: inicia/para o cronometro
*                           - Botão 2: zera o cronometro
*                       Use um dos LED's para indicar se o cronometro está ligado/desligado,
*                       por exemplo, o LED verde.
*                       É importante lembrar que antes de configurar as interrupções
*                       deve-se desabilitar TODAS elas através do registrador INTCONbits.GIE
*                       (General Interrupt Enable) e elas só devem ser habilitadas novamente
*                       antes de entrar no laço infinito da função principal.
*
*****
```

Figura 4.13 - Cabeçalho de arquivo.

3- Nomes:

Use nomes significativos para funções, variáveis, defines, constantes, etc., que refletem a sua função. Exceção a regra: cont ou i, j, k para contadores.

Muitos programadores utilizam nomes em inglês para internacionalizar seus códigos. Independente da língua utilizada existem dicas de como devem ser criados os nomes das variáveis, constantes, funções, etc., de modo a possibilitar a distinção entre elas. A seguir são apresentadas algumas dessas dicas para cada um desses tipos:

- A- Constantes e #defines devem ser escritas usando caixa alta (letras maiúsculas). Caso a palavra seja composta separe-as com _

Exemplo: #define LED_VERDE PORTCbits.RC2

 const float PI = 3,1415;

Contra-exemplo: #define pino17 PORTCbits.RC2

- B- Variáveis: devem ser escritas usando caixa baixa (letras minúsculas). Caso a palavra seja composta, inicie a segunda palavra com a letra maiúscula.

Exemplo: int numeroVolts;

- C- Funções: Funções devem começar com letra maiúscula e as demais devem ser minúsculas. Caso o nome seja composto, inicie a segunda palavra com letra maiúscula;

Exemplo: void ScreenWrite(char *string)

Em alguns casos é interessante o uso de prefixos e sufixos nos nomes das funções. Estes sugerem o comportamento de entrada ou saída da função.

Exemplo1: função retorna “0” caso o display esteja apagado e “1” se aceso. Na função abaixo é utilizado o prefixo “is”.

```
isDisplayOn();
```

Exemplo2: função faz delays (atrasos) de múltiplos (x) de 100TCY. Na função abaixo é utilizado o sufixo “x” para denotar a multiplicação.

```
Delay100TCYx(10);
```

4- Uso de #define.

Use #define em elementos que poderão ser modificados posteriormente em seu programa, como, por exemplo, valores de constantes e portas de entrada e saída (Ver Figura 4.14).

```
// Botões do Kit PICMINAS: entradas digitais
//atribuição: 0 = botao pressionado
//              1 = botao não pressionado
#define BOTAO_1      PORTEbites.RE1
#define BOTAO_2      PORTEbites.RE2

// Chaves DIP-Switch do Kit PICMINAS: entradas digitais
//atribuição: 0 = chave no lado '1' (chave ligada)
//              1 = chave no lado '0' (chave desligada)
#define CH_1          PORTAbits.RA4
#define CH_2          PORTAbits.RA3
#define CH_3          PORTAbits.RA2
#define CH_4          PORTAbits.RA1

//Temperatura:

#define TEMPMAX      50
#define TEMPMIN      25
```

Figura 4.14 - Uso de Defines.

Funções simples, escritas em uma única linha (inline), podem ser substituídas por defines. Neste caso, o nome atribuído ao trecho de código, por meio do define, pode ser escrito de forma a se parecer com uma função. Por exemplo:

```
#define Delay1TCY() Nop()
```

5- Funções:

Sempre que um trecho de código possuir uma funcionalidade específica, que possa ser aproveitada em outras partes do seu programa, esse trecho deve ser encapsulado em uma função. Caso você tenha um conjunto de funções correlacionadas, crie uma biblioteca para agrupá-las. Por exemplo, as funções seno, cosseno, tangente, exponencial, logaritmo, estão todas agrupadas em uma biblioteca chamada “math”. Deixe sempre seu código organizado utilizando protótipos de funções.

6- Reinvenção da Roda:

Não reinvente a roda. Antes de começar a escrever uma solução procure por ela na internet, em bibliotecas do compilador, em outros projetos de sua empresa ou laboratório. Gaste seu tempo desenvolvendo soluções novas que ainda não foram feitas.

7- Ponteiros:

Ponteiros são ferramentas muito poderosas e por isto devem ser utilizados com muita atenção. Para evitar acessos indesejados a memória de dados do programa, sempre que declarar um ponteiro inicialize-o com zero.

Exemplo: **int *vectPoint = 0;**

8- goto e break:

Apesar de existirem em linguagem C, o uso de **goto** e **breaks** deve ser evitado ao máximo em seu programa.

9- calloc() / malloc()

Sempre verifique se o ponteiro retornado por estas funções é diferente de zero. Se o ponteiro for igual a zero, houve uma falha na alocação da memória e nenhuma operação deve ser feita com o uso deste ponteiro.

Referências:

- 4- Práticas e layouts PICMinas.
- 5- IBM - http://www.ibm.com/developerworks/aix/library/au-hook_duttaC.html

Neste capítulo, falamos sobre:

- A linguagem de programação C;
- Tipos de dados, comandos básicos;
- Exemplos de códigos.
- Boas práticas de programação.

