```
0D82:0100 B402          MOV     AH,02
0D82:0102 B241          MOV     DL,41
0D82:0104 CD21          INT     21
0D82:0106 CD20          INT     20
0D82:0108 69            DB      69
0D82:0109 64            DB      64
0D82:010A 61            DB      61
0D82:010B 64            DB      64
0D82:010C 65            DB      65
0D82:010D 206573        AND     [DI+73],AH
0D82:0110 7065          JO      0177
0D82:0112 63            DB      63
0D82:0113 69            DB      69
0D82:0114 66            DB      66
0D82:0115 69            DB      69
0D82:0116 63            DB      63
0D82:0117 61            DB      61
```

Noções preliminares

```
0D82:0100 B402          MOV     AH,02
0D82:0102 B241          MOV     DL,41
0D82:0104 CD21          INT     21
0D82:0106 CD20          INT     20
0D82:0108 69            DB      69
```

Capítulo 1

INTRODUÇÃO

O objetivo deste capítulo é apresentar conceitos introdutórios e básicos para a utilização da linguagem de programação para computadores Assembly baseada nos microprocessadores Intel e AMD usados nos computadores pertencentes a linha IBM-PC baseados na família x86, mais precisamente o modelo 8086. São apresentadas as diferenças entre os termos assembly e assembler, o que são mnemônicos e o que motiva o aprendizado da programação com a linguagem assembly.

1.1 Histórico da linguagem

É oportuno esclarecer um ponto de grande confusão por parte de pessoas leigas no assunto. Não é possível programar computadores com linguagem ***Assembler***. Programa-se computadores, quando em baixo nível ou em linguagem de máquina, com linguagem ***Assembly*** (lê-se assembli).

O termo *assembly* significa *montagem*, ou seja, *linguagem de montagem* (*assembly language*), utilizada para programar um computador próximo ao nível operacional do seu microprocessador, denominado baixo nível, sendo necessário montar o programa na memória do computador para que o programa seja então executado. *Assembly* não é uma linguagem de máquina (como muitos afirmam), mas é a linguagem de programação que está mais próxima disso. Um programa escrito em *Assembly* é montado para ser executado dentro do microprocessador.

A linguagem de máquina é uma ferramenta utilizada por um microprocessador para controlar as funções internas de um computador digital por meio de comandos chamados *opcodes*. Segundo (VISCONTI, 1991, p. 13), o microprocessador utilizado em um computador "*é um circuito que possui a capacidade de executar diversos tipos de funções distintas*". A linguagem de máquina (em seu sentido mais puro) só aceita e manipula informações numéricas expressas em notação de códigos binários (os quais matematicamente representam os estados de tensão alta "1" ou tensão baixa "0" para os circuitos eletrônicos de um computador conectados a energia elétrica) na forma de grupos numéricos como: *nibble*, *byte*, *word* entre outros agrupamentos que são adiante comentados. Por questões de facilidade operacional, os agrupamentos de números binários usados em microprocessadores são representados no formato hexadecimal.

A programação de computadores digitais em linguagem de máquina foi muito utilizada, principalmente durante a década de 1940 (século XX), após o surgimento do primeiro computador eletrônico, o ENIAC (*Electronic Numerical Integrator and Computer*) de 1939 a 1946, Figura 1.1. A primeira linguagem *Assembly* foi desenvolvida em 1947 pela cientista da computação Kathleen Booth que projetou também seu *assembler* para os computadores do Birkbeck College (PRIDDY, 2019) com o objetivo de facilitar o trabalho de codificação de um programa de computador em linguagem de máquina. Nessa ocasião os códigos numéricos (binários ou hexadecimais) da linguagem de máquina foram substituídos por um código alfabético que era muito mais fácil de ser utilizado, pois montar um programa em linguagem de máquina era uma atividade muito dispendiosa (como poderá ser constatado nos próximos capítulos). Um programa que "rodasse" em meia hora para conseguir um tipo de processamento poderia levar para ser escrito, ou seja, montado talvez um, dois ou mais dias de trabalho. No final do capítulo 11 é dada uma ideia deste tipo de trabalho com um simples programa para apresenta a mensagem "alô mundo". A primeira linguagem de programação de alto nível surgiu em 1954 com o lançamento da linguagem FORTRAN (*FORmula TRANslator*).

A linguagem para programação de computadores *Assembly* possui uma estrutura sintática particular. Ela é formada por um conjunto de instruções que, por meio de códigos mais legíveis, representam as instruções do código de máquina.

As instruções da linguagem *Assembly* são conhecidas pelo nome de ***mnemônicos*** (lê-se menemônicos). É muito mais fácil olhar para um *mnemônico* e lembrar o que ele faz do que seu equivalente em código binário ou hexadecimal, legível apenas pelo microprocessador (CPU[1]) do computador.

A linguagem de programação de computadores *Assembly* tem ainda uma grande vantagem sobre a linguagem de máquina, que é o fato de requer menos atenção a detalhes que são exigidos para programar em linguagem de máquina. Desta forma, é uma linguagem de fácil alteração se comparada com a linguagem de máquina.

O programa utilizado para compilar um programa escrito em linguagem de montagem (*assembly*), tornando-o executável em um computador chamasse *assembler*. Um programa ***Assembler*** é basicamente o ambiente de programação. É a ferramenta responsável por traduzir o programa-fonte (escrito em linguagem *Assembly*) para o programa objeto (programa em código de máquina) a ser interpretado por um processador. O programa *Assembler* é uma ferramenta que tem características semelhantes em alguns aspectos aos compiladores para uma determinada linguagem de alto nível.

O erro em confundir *Assembler* (programa montador - compilador) com *Assembly* (linguagem de montagem) é muito comum. Por exemplo, é comum dizer que se programa nas linguagens Delphi, Turbo Pascal, Turbo C, Visual Basic, Quick Basic, Visual C etc. Observe a Tabela 1.1 com os nomes das principais linguagens de programação de computadores e seus respectivos ambientes de programação.

Tabela 1.1 - Ambientes de Programação versus Linguagens de Programação

Linguagem	Ambiente de Programação
Assembly	MASM (Macro Assembler - Microsoft)
	MASM (Macro Assembler - IBM)
	TASM (Turbo Assembler - Borland)
	emu8086
C	Visual C / Visual Studio (Microsoft)
	Turbo C (Borland)
C++	GPP (GNU Project)
	Borland C++ / C++ Builder (Borland)
	Visual C++ (Microsoft)
Object Pascal	Delphi (Borland)
	Turbo Pascal (Borland)
	Kylix (Borland)
	Free Pascal Compiler (Software Livre)

Confundir nomes de compiladores com nomes de linguagens de programação de computadores é inadmissível para um profissional que diz pertencer à área de Tecnologia da Informação, mais precisamente para aquele que se diz um desenvolvedor de *software*, o que infelizmente é muito comum no Brasil. Jamais confunda o nome do ambiente de programação (seja em DOS, Windows, Linux) com a linguagem de programação que se utiliza.

1.2 - Por que aprender Assembly?

A aprendizagem de uso da linguagem de programação *Assembly* por parte de estudantes foi sempre cercada de grande misticismo em relação a ser considerada uma linguagem de difícil aprendizagem. Não se trata de ser uma linguagem de difícil aprendizagem, mas uma linguagem de aprendizagem complexa uma vez que diversos detalhes operacionais no uso de certo microprocessador precisa ser considerado para a execução de diversas operações que são consideradas simples e de fácil assimilação em linguagens de alto nível.

Existe certo preconceito na utilização da linguagem *Assembly* por parte de algumas pessoas. Muitos afirmam que programar nessa linguagem é coisa para "louco". Felizmente estão errados, pois quem programa nessa linguagem não é

[1] Central Processing Unit - *Unidade Central de Processamento.*

louco. É um profissional que conhece e sabe usar melhor os requisitos mais íntimos de um microprocessador, e, por conseguinte, sabe controlar melhor as funções de um computador digital.

Grosso modo pode-se comparar programar em linguagem *Assembly* com o fato de dirigir um automóvel com câmbio mecânico. Programar em uma linguagem de alto nível é como dirigir um automóvel que tem câmbio automático. Para quem gosta de dirigir, a verdadeira emoção está em controlar um automóvel com sistema mecânico de câmbio, pois o automático tira a sensação de controlar verdadeiramente a máquina, mas não tira sua dirigibilidade. Assim sendo, a primeira motivação para aprender a linguagem *Assembly* é ter vontade de controlar melhor as funções internas de um microprocessador. Desta forma, é possível desenvolver programas mais eficientes.

Muitos afirmam que a linguagem *Assembly* não é mais usada, que aprendê-la é perda de tempo. Como diria o menino-prodígio "Santa ignorância, Batman!". *Assembly* é a única linguagem que dá a um programador a capacidade de controlar totalmente as funções internas de um computador. Tanto que é comum muitas linguagens de programação de alto nível possuírem algum tipo de interação com rotinas de programas escritos em linguagem *Assembly*.

A linguagem de programação *Assembly* é muito utilizada na criação e desenvolvimento de rotinas escritas nas formas de DLLs, *drivers* (para controle de periféricos), programas embutidos para computadores de bordo, entre outras aplicações que podem ser encontradas no mercado. *Assembly* é uma linguagem de programação usada para se “conversar” diretamente com o *hardware* de um computador. Na década de 1980, século XX, um programa aplicativo de planilha eletrônica muito conhecido denominado Lotus 1-2-3 fez muito sucesso. Esse programa foi totalmente escrito em *Assembly*.

Não pense você que os melhores jogos do mercado são apenas escritos em linguagem de alto nível. Sempre existe em algum canto do programa algum trecho escrito em *Assembly*. Conhecer a linguagem *Assembly* é conhecer a verdadeira liberdade de programar um computador.

Os programas escritos em linguagem *Assembly* são rápidos e de pequeno tamanho (após a compilação) se comparados com códigos similares escritos em linguagem de alto nível. Em contrapartida, seu código-fonte é sempre maior do que o escrito em uma linguagem de alto nível. Por esta razão muitos acham que programar nessa linguagem é mais difícil, quando na verdade é apenas um pouco mais trabalhoso.

Norton & Socha (1988) afirmam que "em relação a todas as outras linguagens de programação, a linguagem *Assembly* é o denominador comum entre elas". Acrescentam ainda que aprender a linguagem *Assembly* é entender melhor o funcionamento de um microprocessador dentro de um computador, e torna-se possível entender melhor muitos dos elementos encontrados em outras linguagens de alto nível.

Como desvantagem no uso da linguagem de programação *Assembly* pode-se apontar o fato desta linguagem estar vinculada ao microprocessador em uso. A linguagem *Assembly* usada num microprocessador Intel ou AMD (para computadores da linha IBM-PC) é diferente da usada em um microprocessador da Motorola, por exemplo, o que obrigaria um programa a ser reescrito de um microprocessador Intel para um Motorola. Outra desvantagem é o fato da linguagem *Assembly* exigir do programador bom nível de conhecimento da arquitetura do computador em uso, principalmente na parte que tange ao uso do processador a ser programado e sua relação com os periféricos conectados.

Há ainda a questão do tamanho dos códigos de programa, que tendem a ser grandes, pois são necessárias muitas instruções para a realização de tarefas muitas vezes pequenas. Mas isto ocorre devido ao fato de se estar programando um computador praticamente *byte* a *byte* em sua memória e não por meio de instruções em alto nível. No entanto, a estrutura de escrita de um programa *Assembly* é muito mais simples que a estrutura utilizada nas linguagens de alto nível.

1.3 - Restrições deste trabalho

O objetivo desta obra é fazer um trabalho de introdução e apresentação dos fundamentos iniciais e básicos da linguagem de programação *Assembly 8086/8088*. Não se trata de um trabalho de grande profundidade técnica. Para tanto, está sendo levado em consideração o uso de um microcomputador da família IBM-PC dotado de um microprocessador Intel ou AMD que operem o modo 8086/8088.

O principal fator que levou à escolha da linguagem *Assembly* em modo *8086/8088* é o fato de ser o primeiro padrão dessa linguagem ensinado em vários cursos nas escolas do Brasil e no restante do mundo até os dias atuais para microcomputadores da família IBM-PC.

Outro fator considerado é a disponibilidade dos programas **DEBUG** da Microsoft (encontrado em todas as versões dos sistemas operacionais MS-DOS e nas versões 32 *bits* do sistema operacional Microsoft Windows XP, Vista, 7, 8, 8.1 e 10. Infelizmente o modo 64 *bits* dos sistemas operacionais Microsoft Windows não possui o programa *DEBUG*), **DOS Debug** (desenvolvido para o sistema operacional FreeDOS) e da ferramenta de emulação assembler **emu8086** (que pode ser executada em todas as versões do sistema operacional Microsoft Windows, tanto em modo 32 *bits* como em modo 64 *bits*). Essas ferramentas operam com base na estrutura de registradores de 16 *bits* por estarem voltadas ao padrão 8086/8088, exceto o programa **DOS Debug** que possibilita trabalhar com registradores de 16 e 32 bits.

Em relação aos programas de depuração (**DEBUG** e **DOS Debug**) será dada atenção ao programa **DOS Debug** por ser uma ferramenta distribuída de forma livre, não se tendo nenhum problema de direito autoral com seu uso, além deste ser distribuído com seu código fonte que poderá ser usado como fonte de estudo após a apresentação desta obra.

O programa **emu8086** que também será usado é um ambiente de programação e simulação do modo de operação interna de um microprocessador (micro controlador) padrão 8086/8088. A vantagem dessa ferramenta é ser executada em modo gráfico, além de apresentar todos os detalhes da operação de um microprocessador de um modo mais concreto, pode ser executado no Microsoft Windows XP, 7, 8, 8.1 e 10 de 32 ou de 64 *bits*. No entanto, há de se considerar que o programa **emu8086** é um *software* pago, devendo-se adquirir sua licença para uso além do tempo de experiência. A Figura 1.2 exibe algumas telas do programa **emu8086**.

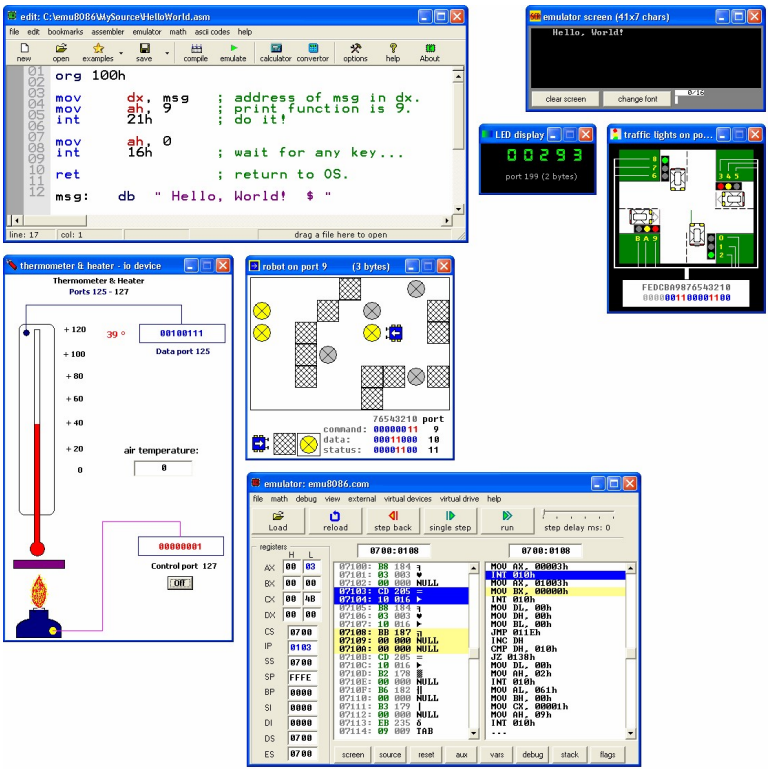

Figura 1.2 - Telas do Programa emu8086.

Como ambiente inicial de aprendizagem será utilizada a ferramenta **DOS Debug** e posteriormente será utilizado o programa **emu8086**.

Pelo fato da programação de computadores em linguagem *Assembly* para processadores de 32 ou 64 *bits* ser um pouco mais complexa que os objetivos desta obra este modo de operação não será aqui abordado. No entanto, a visão dada em modo 16 *bits* permitirá fácil entendimento de outros modos de uso.

Para estudar o conteúdo aqui apresentado, é necessário que o leitor possua alguns pré-requisitos básicos:

1. Sólidos conhecimentos das técnicas de programação (programação sequencial, utilização de decisões, laços de repetição, estruturas de dados homogêneas e heterogêneas, sub-rotinas e passagens de parâmetros) com o uso

de algoritmos computacionais[2], e preferencialmente tenha conhecimentos básicos de estruturas de dados[3], principalmente no que tange às técnicas de controle de listas (pilhas e filas), além de conhecimentos sobre o tema organização de computadores.

2. Apesar de não ser obrigatório, é interessante possuir conhecimentos de uso e aplicação de alguma linguagem de programação de alto nível, preferencialmente Pascal, C, ou C++[4].
3. Grande predisposição pessoal para aprender a utilizar a linguagem *Assembly*, pois talvez seja necessário estudar um determinado capítulo mais de uma vez e recorrer a materiais adicionais para suprir alguma deficiência que possa existir neste livro por ser o seu conteúdo básico e introdutório.
4. Conhecimento da manipulação de bases numéricas (binário, decimal, hexadecimal, octal). Conhecer e dominar as etapas do processo de cálculo nas várias bases numéricas[5].

Cabe lembrar que os pré-requisitos de conhecimento mínimos necessários para este estudo não serão tratados no livro por estarem fora do seu escopo básico.

1.4 - Por que usar o padrão 8086 e não outro mais recente?

Muitos podem se perguntar por qual motivo esta obra, contemporânea, faz uso do padrão 8086 de 1978 ao invés de fazer uso, por exemplo, de um modelo mais recente como os modelos i7 da Intel ou os modelos FX-Seies, Sempron, Athlon ou Opteron da AMD? Não seria melhor usar o padrão 64 *bits* ao invés de usar o padrão 16 *bits*?

A resposta a essa questão, pode parecer, muito simplista aos olhos da maioria, mas de fato não o é. O que na prática pedagógica de sala de aula adiantaria pensar em usar os modelos mais modernos de processadores de 64 *bits* ou até processadores de maior capacidade que venham a ser lançados com um conjunto de instruções muito extenso? Seria isso adequado ao educando? A resposta mais simples a tais questões seria que uma coisa não tem a ver com a outra, o fato de se fazer uso de um modelo mais sofisticado de processador não garante efetivamente boa aprendizagem por parte do educando. Seria apenas uma atitude em vão para ficar “bonito na fita”.

A ideia deste trabalho é passar as bases essenciais de entendimento do uso da linguagem *Assembly* operada a partir do padrão Intel/AMD. Se o educando entender adequadamente o funcionamento do padrão 8086 estará apto a entender o funcionamento de outros padrões mais sofisticados. É o mesmo raciocínio de se aprender a dirigir, nenhuma autoescola brasileira possui para aprendizagem carros muito sofisticados como, por exemplo, Ferrari ou Lamborghini; possuem modelos simples, na sua maioria com motorização 1.0. E nem por isso o motorista deixa de aprender a dirigir e poderá futuramente dirigir carros de alto desempenho.

Assim sendo, considere o conteúdo deste livro como a aprendizagem da linguagem *Assembly* em modelo 1.0 e posteriormente poderá ampliar seu foco de aprendizagem em um modelo mais sofisticado. Lembre-se de que ninguém atinge o último degrau de uma longa escada sem passar pelo primeiro degrau.

1.5 - Arquitetura: Princípios básicos

Antes de iniciar o estudo e aplicação da linguagem Assembly é pertinente abordar algumas questões relacionadas a arquitetura dos computadores padrão IBM-PC.

---

[2] *Sugere-se a leitura da obra "Algoritmos - Lógica para o Desenvolvimento de Programação de Computadores", dos autores José Augusto N. G. Manzano e Jayr Figueiredo, Editora Érica.*

[3] *É aconselhável consultar a obra "Estrutura de Dados Fundamentais", do autor Silvio do Lago Pereira, Editora Érica.*

[4] *Seria conveniente, neste caso, consultar um dos livros: "Estudo Dirigido de Linguagem C", do autor José Augusto N. G. Manzano; "Free Pascal - Programação de Computadores - Guia Básico de Orientação e Desenvolvimento para Programação em Linux, MS-Windows e MS-DOS", dos autores José Augusto N. G. Manzano e Wilson Y. Yamatumi; "Programação de Computadores com C++ ANSI (ISO/IEC 14882:2003) - Guia Prático de Orientação e Desenvolvimento", do autor José Augusto N. G. Manzano, Editora Érica.*

[5] *Para aprofundamento de alguns dos detalhes aqui apresentados, o leitor pode consultar as obras: "Estudo Dirigido de Informática Básica", publicada pela Editora Érica, dos autores André Luiz N. G. Manzano e Maria Izabel N. G. Manzano, e "Introdução à Informática", publicada pela Editora Pearson Education, do autor Peter Norton.*

O termo computador advém da palavra *computare* que no idioma Latim refere-se a calcular. Um computador é um equipamento eletrônico que efetua o processamento de dados. O processamento realizado por um computador pode ser feito nas esferas matemática e lógica. Como equipamento, esta máquina é composta por uma série de circuitos interligados que permitem a realização de diversas tarefas controladas por programas.

Os computadores usados no dia-a-dia da humanidade obedecem a uma estrutura computacional (arquitetura) chamada Eckert-Mauchly, proposta por John von Neumann e criada por John Presper Eckert e John William Mauchly. A arquitetura proposta por von Newmann propõe um equipamento computacional a partir de quatro seções operacionais, sendo:

- Unidade Lógica e Aritmética (Arithmetic Logic Unit – ULA) – responsável por executar as operações matemáticas e lógicas;
- Unidade de Controle – responsável pela execução de instruções junto ao processador, que é por sua natureza o "cérebro" de um computador por transformar dados em;
- Memória Principal – dispositivo responsável em armazenar dados e códigos de programas;
- Dispositivos de Entrada e Saída – responsáveis pelas operações de comunicação efetuadas em um ser humano e um computador. Estes dispositivos são conhecidos como periféricos.

Todas estas partes encontram-se interligadas por meio de vias de acesso denominadas barramentos (*bus*).

Todas as ações computacionais como já informado, são executadas por meio de programas, que são por sua natureza, um conjunto de instruções organizadas de forma lógica com o objetivo de realizar certa ação e operação por meio de instruções. Uma instrução de programa é uma ordem enviada a um computador por meio da definição de um comando e um dado. No contexto da programação em baixo nível uma instrução é definida a partir de um código de operação (*opcode*) e de um ou mais operandos que são elementos que representam dados e a armazenagem de endereços de acesso à memória. O *opcode* representa a instrução a ser executada e o operando representa o dado a ser tratado. O *opcode* é representado por um código binário que identifica uma instrução e determina o tipo de ação a ser realizada pelo processador. Uma instrução *Assembly* pode conter, além do *opcode* opcionalmente um ou mais campos de *operandos*. Uma instrução somente com *opcode* caracteriza-se por ser uma instrução que não faz acesso à memória. Assim sendo uma instrução pode ser definida a partir do layout:

OPCODE	OPERANDO 1	OPERANDO 2	...	OPERANDO N

As instruções *Assembly* podem ser usadas para:

- Efetuar ações matemáticas e lógicas;
- Efetuar movimentação de dados no processador;
- Efetuar operações de entrada e saída de dados/informações;
- Efetuar o controle de desvios condicionais e a execução de laços.

A execução de instruções em um processador é efetuada pelos registradores, que são dispositivos que efetuam o armazenamento de dados e de informações utilizadas para a execução de instruções de um programa. Existem dois tipos de registradores:

- Registradores gerais;
- Registradores de propósito especial (segmento, deslocamento e estado).

Os registradores de propósito geral são usados no armazenamento de dados e sua movimentação entre a memória e o processador. Já os registradores de propósito especial são usados na execução de instruções, normalmente, gerenciados pela unidade de controle.

A execução de instruções de um programa *Assembly* pelo processador segue as seguintes etapas:

- Busca da instrução na memória do computador;
- Decodificação da instrução pelo processador;
- Busca do operando na memória do computador;
- Execução da instrução pelo processador;
- Escrita do resultado, normalmente, na memória do computador.

As etapas na execução das instruções pelo processador são internamente independentes. Enquanto certa instrução está na segunda etapa, a unidade de controle coloca a próxima instrução na primeira etapa. Esse tipo de operação é

conhecida como *pipeline*. As instruções de um programa são executadas pelo *pipeline* de forma simultânea, desde que, cada instrução seja processada em uma etapa diferente da outra.

Para que um programa seja executado é importante que este tenha acesso ao endereçamento de memória do computador. O endereço de memória determina o local onde o operando se encontra, sendo possível fazer uso de cinco tipos de endereçamento de memória, sendo:

- Endereçamento imediato (modo imediato);
- Endereçamento direto (modo direto);
- Endereçamento indireto (modo indireto);
- Endereçamento por registrador;
- Endereçamento indexado (modo indexado).

O endereçamento imediato é usado quando o valor de um operando é representado pelo próprio dado. Neste caso, sendo este um valor constante. Neste endereçamento o dado não precisa ser procurado na memória, o que faz com que sua execução seja rápida. No entanto, o tamanho do dado fica limitado ao tamanho do campo do operando.

O endereçamento direto é usado quando o valor do operando é o endereço de memória onde certo dado se encontra armazenado. Este modo de endereçamento é usado quando da necessidade de operar com valores que se alteram ao longo da execução do programa. Esta técnica de ação é mais lenta que o endereçamento imediato e mais rápida que o endereçamento indireto. Este tipo de endereçamento limita-se ao tamanho do campo do operando que é reduzido, não sendo possível endereçar toda a memória.

O endereçamento indireto é usado quando o valor do operando é uma referência a uma posição de memória que contenha o endereço do próprio operando. O endereço apontado é um ponteiro vinculado a certo dado na memória principal. Não há para este tipo de endereçamento limitação de células endereçáveis de memória, o endereço da memória principal não fica limitado ao tamanho do operando, como ocorre no endereçamento direto. O endereçamento indireto é mais lento que o modo direto. Este realiza mais acessos a memória: busca de endereço (ponteiro) e busca de dado.

O endereçamento por registrador é usado quando há a necessidade do valor do operando fazer referência a um dado (endereçamento por registrador direto) ou referencia a um endereço (endereçamento por registrador indireto) quando se realiza o apontamento para um local de memória que contenha um dado. O acesso por registradores ocorre de forma mais rápida do que o acesso realizado a memória e as outras maneiras de endereçamento existentes. O endereçamento por registrador não é adequado para a transferência de dados entre memória e processador, possui como limitação a quantidade de registradores gerais disponíveis que no caso da arquitetura Intel são quatro identificados como **A**, **B**, **C** e **D**.

Por fim, não menos importante o endereçamento indexado é usado a partir da soma do campo operando com o valor que esteja armazenado em um registrador, sendo este chamado de índice. Este tipo de endereçamento é adequado para a manipulação de estruturas de dados como matrizes de uma dimensão na forma de vetores e de mais dimensões na forma de tabelas.

Note que foi muito mencionado a palavra “memória”. No contexto computacional a memória de um computador é um dispositivo que permite estabelecer o armazenamento de dados e programas de forma temporária (memória principal) ou “permanente” (memória secundária). O termo “permanente” entre aspas se refere deste tipo de armazenamento ser controlado pelo usuário.

A memória principal, conhecida como memória de acesso randômico (Ramdom Access Memory – RAM) tem por característica ser volátil e manter dados e programas enquanto os componentes internos do computador se encontram energizados.

A memória secundária caracteriza-se por ser usada para o armazenamento permanente de dados e programas na forma de mídias externas (discos rígidos, discos ópticos, pen-drives, entre outras possibilidades).

1.6 Código ASCII

O código ASCII (*American Standard Code for Information Interchange*) foi desenvolvido a partir de 1963 e concluído em 1968, com a participação de várias companhias de comunicação norte-americanas, com o objetivo de substituir o até então utilizado código de Baudot.

O código de Baudot usava apenas 5 bits, o que possibilitava obter apenas 32 combinações diferentes de caracteres (números e letras maiúsculas), muito utilizado entre teleimpressores.

Este apêndice apresenta a tabela de códigos ASCII (lê-se *asquí* e não *asqui 2)* com a definição dos seus códigos numéricos em notação decimal, notação binária e notação hexadecimal.

O código ASCII estabelece a utilização de 128 símbolos (de 0 a 127) diferentes, utilizando efetivamente um conjunto de 7 *bits* dos 8 *bits* permitidos. O *bit* mais significativo permanece zerado. Do conjunto de 128 símbolos, 96 (de 32 a 127) são caracteres imprimíveis (números, letras minúsculas, letras maiúsculas e sinais de pontuação) e 32 (de 0 a 31) são caracteres não imprimíveis (retorno de carro, retrocesso, salto de linha, entre outros). Estes 128 caracteres são definidos como padrão para qualquer tipo de computador e seguem a estrutura da tabela seguinte.

Dec	Bin	Hex	Caractere
000	00000000	00	NUL
001	00000001	01	SOH
002	00000010	02	STX
003	00000011	03	ETX
004	00000100	04	EOT
005	00000101	05	ENQ
006	00000110	06	ACK
007	00000111	07	BEL
008	00001000	08	BS
009	00001001	09	HT
010	00001010	0A	LF
011	00001011	0B	VT
012	00001100	0C	FF
013	00001101	0D	CR
014	00001110	0E	SO
015	00001111	0F	SI
016	00010000	10	DEL
017	00010001	11	DC1
018	00010010	12	DC2
019	00010011	13	DC3
020	00010100	14	DC4
021	00010101	15	NAK
044	00101100	2C	,
045	00101101	2D	-
046	00101110	2E	.
047	00101111	2F	/
048	00110000	30	0
049	00110001	31	1
050	00110010	32	2
051	00110011	33	3
052	00110100	34	4
053	00110101	35	5
054	00110110	36	6
055	00110111	37	7

Dec	Bin	Hex	Caractere
022	00010110	16	SYN
023	00010111	17	ETB
024	00011000	18	CAN
025	00011001	19	EM
026	00011010	1A	SUB
027	00011011	1B	ESC
028	00011100	1C	FS
029	00011101	1D	GS
030	00011110	1E	RS
031	00011111	1F	US
032	00100000	20	SPACE
033	00100001	21	!
034	00100010	22	"
035	00100011	23	#
036	00100100	24	$
037	00100101	25	%
038	00100110	26	&
039	00100111	27	'
040	00101000	28	(
041	00101001	29	)
042	00101010	2A	*
043	00101011	2B	+
086	01010110	56	V
087	01010111	57	W
088	01011000	58	X
089	01011001	59	Y
090	01011010	5A	Z
091	01011011	5B	[
092	01011100	5C	\
093	01011101	5D	]
094	01011110	5E	^
095	01011111	5F	_
096	01100000	60	`
097	01100001	61	a

Dec	Bin	Hex	Caractere
056	00111000	38	8
057	00111001	39	9
058	00111010	3A	:
059	00111011	3B	;
060	00111100	3C	<
061	00111101	3D	=
062	00111110	3E	>
063	00111111	3F	?
064	01000000	40	@
065	01000001	41	A
066	01000010	42	B
067	01000011	43	C
068	01000100	44	D
069	01000101	45	E
070	01000110	46	F
071	01000111	47	G
072	01001000	48	H
073	01001001	49	I
074	01001010	4A	J
075	01001011	4B	K
076	01001100	4C	L
077	01001101	4D	M
078	01001110	4E	N
079	01001111	4F	O
080	01010000	50	P
081	01010001	51	Q
082	01010010	52	R
083	01010011	53	S
084	01010100	54	T
085	01010101	55	U

Dec	Bin	Hex	Caractere
098	01100010	62	b
099	01100011	63	c
100	01100100	64	d
101	01100101	65	e
102	01100110	66	f
103	01100111	67	g
104	01101000	68	h
105	01101001	69	i
106	01101010	6A	j
107	01101011	6B	k
108	01101100	6C	l
109	01101101	6D	m
110	01101110	6E	n
111	01101111	6F	o
112	01110000	70	p
113	01110001	71	q
114	01110010	72	r
115	01110011	73	s
116	01110100	74	t
117	01110101	75	u
118	01110110	76	v
119	01110111	77	w
120	01111000	78	x
121	01111001	79	y
122	01111010	7A	z
123	01111011	7B	{
124	01111100	7C	|
125	01111101	7D	}
126	01111110	7E	~
127	01111111	7F	⌂

Os códigos ASCII existentes na faixa de 0 até 31 são de controle para envio de dados a um periférico de impressão, para controlar a comunicação entre dois computadores por rede telefônica ou definir algum tipo de comunicação entre o computador de um agente externo. Quando direcionados para a tela do monitor de vídeo, esses caracteres são substituídos por algum símbolo imprimível que não é nenhum dos caracteres imprimíveis. A título de curiosidade observe a tabela seguinte.

Dec	Bin	Hex	Controle	Imprimível	Teclas	Descrição
000	00000000	00	NUL		Ctrl + @	Caractere nulo
001	00000001	01	SOH	☺	Ctrl + A	Início de cabeçalho
002	00000010	02	STX	☻	Ctrl + B	Início de texto
003	00000011	03	ETX	♥	Ctrl + C	Fim de texto
004	00000100	04	EOT	♦	Ctrl + D	Fim da transmissão
005	00000101	05	ENQ	♣	Ctrl + E	Caractere de consulta

Dec	Bin	Hex	Controle	Imprimível	Teclas	Descrição
006	00000110	06	ACK	♠	Ctrl + F	Confirmação
007	00000111	07	BEL	▪	Ctrl + G	Alarme ou chamada
008	00001000	08	BS	◘	Ctrl + H	Retrocesso
009	00001001	09	HT	○	Ctrl + I	Tabulação horizontal
010	00001010	0A	LF	◙	Ctrl + J	Avanço de linha
011	00001011	0B	VT	♂	Ctrl + K	Tabulação vertical
012	00001100	0C	FF	♀	Ctrl + L	Avanço de página
013	00001101	0D	CR	♪	Ctrl + M	Retorno de carro
014	00001110	0E	SO	♫	Ctrl + N	Shift desativado
015	00001111	0F	SI	☼	Ctrl + O	Shift ativado
016	00010000	10	DEL	►	Ctrl + P	Caractere de supressão
017	00010001	11	DC1	◄	Ctrl + Q	Contr. de dispositivo
018	00010010	12	DC2	↕	Ctrl + R	Contr. de dispositivo
019	00010011	13	DC3	‼	Ctrl + S	Contr. de dispositivo
020	00010100	14	DC4	¶	Ctrl + T	Contr. de dispositivo
021	00010101	15	NAK	§	Ctrl + U	Confirmação negada
022	00010110	16	SYN	■	Ctrl + V	Sincronismo
023	00010111	17	ETB	↨	Ctrl + W	Fim de bloco de texto
024	00011000	18	CAN	↑	Ctrl + X	Cancelamento
025	00011001	19	EM	↓	Ctrl + Y	Fim de meio de dados
026	00011010	1A	SUB	→	Ctrl + Z	Substituição
027	00011011	1B	ESC	←	Ctrl + 9	Diferenciação
028	00011100	1C	FS	∟	Ctrl + /	Separador de arquivo
029	00011101	1D	GS	↔	Ctrl + :	Separador de grupo
030	00011110	1E	RS	▲	Ctrl + ^	Separador de registro
031	00011111	1F	US	▼	Ctrl + _	Separador de unidade

Além dos 128 códigos padronizados, os computadores padrão IBM-PC possuem uma extensão a mais da tabela ASCII, totalizando 256 caracteres. No entanto, essa extensão não deve ser considerada parte da tabela ASCII, mas um complemento definido, que somente é válido em microcomputadores da linha IBM-PC operando com ambiente de software Microsoft. Os códigos estendidos situam-se na faixa de 128 até 255 e não serão apresentados.