e-Tec Brasil
Escola Técnica Aberta do Brasil

Tecnologia da Informática

Saul Azzolin Bonaldo

Colégio Técnico Industrial
UFSM

Santa Maria - RS
2011

Presidência da República Federativa do Brasil

Ministério da Educação

Secretaria de Educação a Distância

Este Material Didático foi elaborado pelo Colégio Técnico Industrial de Santa Maria para o Sistema Escola Técnica Aberta do Brasil – e-Tec Brasil.

Comissão de Acompanhamento e Validação
Universidade Federal de Santa Catarina/UFSC

Coordenação Institucional
Araci Hack Catapan/UFSC

Coordenação do Projeto
Silvia Modesto Nassar/UFSC

Cordenação de Design Instrucional
Beatriz Helena Dal Molin/UNIOESTE

Designers Intrucionais
Helena Maria Maullmann/UFSC
Jorge Luiz Silva Hermenegildo/CEFET-SC

WEB Designers
Beatriz Helena Dal Molin/UNIOESTE
Mércia Freire Rocha Cordeiro Machado/ETUFPR

Supervisão de Projeto Gráfico
Ana Carine García Montero/UFSC

Diagramação
João Ricardo Zattar/UFSC
Luís Henrique Lindler/UFSC

Revisão
Lúcia Locatelli Flôres/UFSC

Comissão de Acompanhamento e Validação
Colégio Técnico Industrial de Santa Maria/CTISM

Coordenador Institucional
Paulo Roberto Colusso/CTISM

Professor-autor
Saul Azzolin Bonaldo/CTISM

Coordenação Técnica
Iza Neuza Teixeira Bohrer/CTISM

Coordenação de Design
Erika Goellner/CTISM

Revisão Pedagógica
Andressa Rosemárie de Menezes Costa/CTISM
Francine Netto Martins Tadielo/CTISM
Marcia Migliore Freo/CTISM

Revisão Textual
Lourdes Maria Grotto de Moura/CTISM
Vera da Silva Oliveira/CTISM

Revisão Técnica
Eduardo Lehnhart Vargas/CTISM

Ilustração e Diagramação
Gustavo Schwendler/CTISM
Leandro Felipe Aguilar Freitas/CTISM
Marcel Santos Jacques/CTISM
Máuren Fernandes Massia/CTISM
Rafael Cavalli Viapiana/CTISM
Ricardo Antunes Machado/CTISM

Ficha catalográfica elaborada por Maristela Eckhardt – CRB 10/737
Biblioteca Central – UFSM

B697t **Bonaldo, Saul Azzolin**
Tecnologia da informática / Saul Azzolin Bonaldo. – 3. ed. – Santa Maria:Universidade Federal de Santa Maria, Colégio Técnico Industrial de Santa Maria, Curso Técnico em Automação Industrial, 2011.
73 p. : il. ; 21 cm.

1. Informática 2. Tecnologia 3. Arquitetura de computadores 4. Software 5. Programa Escola Aberta do Brasil I. Universidade Federal de Santa Maria. Curso Técnico em Automação Industrial

CDU 004

Apresentação e-Tec Brasil

Prezado estudante,

Bem-vindo ao e-Tec Brasil!

Você faz parte de uma rede nacional pública de ensino, a Escola Técnica Aberta do Brasil, instituída pelo Decreto nº 6.301, de 12 de dezembro 2007, com o objetivo de democratizar o acesso ao ensino técnico público, na modalidade a distância. O programa é resultado de uma parceria entre o Ministério da Educação, por meio das Secretarias de Educação a Distância (SEED) e de Educação Profissional e Tecnológica (SETEC), as universidades e escolas técnicas estaduais e federais.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade, e promover o fortalecimento da formação de jovens moradores de regiões distantes dos grandes centros geograficamente ou economicamente.

O e-Tec Brasil leva os cursos técnicos a locais distantes das instituições de ensino e para a periferia das grandes cidades, incentivando os jovens a concluir o ensino médio. Os cursos são ofertados pelas instituições públicas de ensino e o atendimento ao estudante é realizado em escolas-polo integrantes das redes públicas municipais e estaduais.

O Ministério da Educação, as instituições públicas de ensino técnico, seus servidores técnicos e professores acreditam que uma educação profissional qualificada – integradora do ensino médio e educação técnica, – é capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!

Desejamos sucesso na sua formação profissional!

Ministério da Educação
Janeiro de 2010

Nosso contato
etecbrasil@mec.gov.br

Indicação de ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.

Atenção: indica pontos de maior relevância no texto.

Saiba mais: oferece novas informações que enriquecem o assunto ou "curiosidades" e notícias recentes relacionadas ao tema estudado.

Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.

Mídias integradas: sempre que se desejar que os estudantes desenvolvam atividades empregando diferentes mídias: vídeos, filmes, jornais, ambiente AVEA e outras.

Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Sumário

Palavra do professor-autor

"Em meu princípio está meu fim."

T. S. Eliot, "East Coker"

Juntamente com o progresso das comunicações, fenômenos interessantes surgiram: o advento do rádio não causou estragos na vendagem de jornais, a televisão não acabou com o rádio e a internet não decretou o fim de nenhuma das mídias tradicionais – jornal, rádio e televisão.

É provável que o grande interesse dos indivíduos por informação e conhecimento explique esse fenômeno. Sabemos também que, quanto mais livre for uma nação, maior a liberdade de informação. Quando surgem governos ditatoriais, uma das primeiras decisões é decretar o fim da liberdade de imprensa, seja fechando jornais, emissoras de rádio e televisão, seja criando conselhos de regulamentação, chegando até mesmo a tentativas de restrição ao livre acesso à internet, através do bloqueio a determinados *sites*, ou a tentativas de catalogar usuários e armazenar seus rastros.

No entanto, o livre acesso à informação traz outra consequência: hoje, nós dispomos de acesso instantâneo a uma quantidade de informação muito maior do que conseguimos tratar. E a principal ferramenta responsável pela disseminação da informação pelos mais diversos meios é o computador.

Assim, o objetivo principal desta disciplina é oportunizar aos estudantes conhecimentos sobre a arquitetura de computadores e sobre o funcionamento desse equipamento onipresente no mundo atual.

Acreditamos na sua capacidade de crescimento profissional e trabalhamos para que esta disciplina desempenhe um importante papel nesse sentido. Parabéns pela escolha do curso e mãos à obra!

Saul Azzolin Bonaldo

Apresentação da disciplina

A disciplina de Tecnologia da Informática apresenta-se na forma presencial e a distância. A carga horária total é de 30 horas, sendo 24 horas no Ambiente Virtual de Ensino Aprendizagem (AVEA), 4 horas teórico-práticas presenciais, divididas em dois encontros de 2 horas e, ainda, 2 horas destinadas à avaliação presencial.

Este material didático contém as palavras do professor-autor, projeto instrucional, referências e roteiro de estudo dividido em quatro unidades:

1. Sistemas e códigos numéricos e aritmética binária;

2. A arquitetura de um computador;

3. Tradução de programas;

4. Entradas e saídas.

Estas aulas serão estudadas de forma sequencial, permitindo ao aluno conhecer o funcionamento dos computadores, relacionando os conhecimentos adquiridos nas unidades estudadas, visando identificar suas diversas aplicações em automação industrial.

Cada aula de estudo apresenta uma introdução, objetivos e um roteiro de estudo, composto por textos, figuras, exemplos, equações, *link*s, mídias integradas, questionamentos, reflexões, lembretes, atividades de aprendizagem e síntese. As atividades de aprendizagem, dispostas quase ao final da aula, visam a reforçar o entendimento dos conceitos, bem como as aplicações do assunto estudado ao cotidiano. O resumo serve como reforço ao conteúdo apresentado na aula e auxilia o estudante na passagem de uma aula para a outra. Devido ao fato da Tecnologia da Informática estar em constante mutação e evolução, serão disponibilizadas no AVEA atividades de aprendizagem adicionais. Os questionamentos presentes no material didático, bem como as atividades de aprendizagem serão objetos de discussão em fóruns específicos no AVEA.

No AVEA serão utilizados recursos como hipertextos, animações, simulações, vídeos e imagens, vinculando o ambiente virtual ao material didático. Também disponibilizados cronogramas de estudo, atividades de aprendizagem e atividades de avaliação para cada aula, contendo instruções, períodos e prazos. As atividades de aprendizagem e de avaliação serão desenvolvidas através da participação em fóruns e *chat*s, bem como pela realização de tarefas, exercícios e pesquisas complementares. Essas atividades serão avaliadas pelo professor e pelos tutores da disciplina, cujos resultados e sugestões serão divulgados aos alunos.

Os critérios para a avaliação da disciplina serão disponibilizados no ambiente virtual, contemplando as atividades de aprendizagem e de avaliação no AVEA, atividades práticas presenciais e avaliação presencial.

O cronograma de aulas práticas presenciais, bem como pré-requisitos, tarefas, procedimentos e instruções de segurança nos laboratórios práticos, serão disponibilizados no AVEA.

Projeto instrucional

Disciplina: Tecnologia da Informática (carga horária: 30h).

Ementa: Sistemas e códigos numéricos e aritmética binária. A arquitetura de um computador. Tradução de programas. Entradas e saídas.

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
1. Sistemas, códigos numéricos e aritmética binária	Reconhecer, através do histórico da eletricidade, a importância dos sistemas numéricos na evolução da humanidade. Compreender os elementos básicos da base decimal: *bit* e *byte*. Conhecer a aritmética binária e a representação dos resultados.	Ambiente virtual: plataforma *moodle*. Apostila didática. Recursos de apoio: *links* de leitura complementar indicados na apostila.	06
2. A arquitetura de um computador	Compreender o funcionamento básico da arquitetura de Von Neumann. Relacionar as diferentes partes constitutivas de um computador que obedece ao modelo de Von Neumann.	Ambiente virtual: plataforma *moodle*. Apostila didática. Recursos de apoio: *links* de leitura complementar indicados na apostila.	06
3. Tradução de programas	Entender por que programas em linguagem de alto nível devem ser traduzidos para linguagem de máquina. Compreender como se dá a interpretação e compilação de programas. Conhecer emuladores e máquinas virtuais e compreender sua importância.	Ambiente virtual: plataforma *moodle*. Apostila didática. Recursos de apoio: *links* de leitura complementar indicados na apostila.	06
4. Entradas e saídas	Entender de que forma nossos comandos ingressam no computador. Compreender como podemos ter acesso aos resultados do processamento das informações pelo computador. Diferenciar as diversas formas de comunicação de e para o computador, bem como a maneira como se dá o armazenamento de informações em um computador.	Ambiente virtual: plataforma *moodle*. Apostila didática. Recursos de apoio: *links* de leitura complementar indicados na apostila.	06

Aula 1 – Sistemas, códigos numéricos e aritmética binária

"Há tantas auroras que não brilharam ainda."

Rig-Veda

Objetivos

Reconhecer, através do histórico da eletricidade, a importância dos sistemas numéricos na evolução da humanidade.

Compreender os elementos básicos da base decimal: *bit* e *byte*.

Conhecer a aritmética binária e a representação dos resultados.

1.1 Histórico

Além de transmitidas e recebidas, as informações podem ser armazenadas e depois reproduzidas em livros, em discos, em fotografias e na memória humana. Dessa forma, uma informação original pode ser transmitida muitas vezes e pelos mais diversos meios.

O termo multimídia vem da palavra inglesa *multimedia* e representa os diversos meios pelos quais as informações são transmitidas e compartilhadas (multimeios).

O **processamento da informação** relaciona-se com o armazenamento, a transmissão, a combinação e a comparação da informação.

De certa forma, cada ser vivo é também um computador. Nesse caso, os sentidos são meios de receber sinais do meio ambiente. Essas impressões sensoriais são transmitidas por uma rede nervosa ao cérebro através de sinais elétricos e químicos. Os sons emitidos pelos seres vivos são também meios de transmitir informações a outros seres; são mensagens que exprimem vontades, impressões, ordens, etc.

O cérebro humano possui dois hemisférios, sendo que o comportamento do hemisfério direito assemelha-se ao funcionamento de um processador paralelo, e o comportamento do hemisfério esquerdo assemelha-se ao funcionamento de um processador serial.

Saiba mais em: http://www.ted.com/index.php/talks/jill_bolte_taylor_s_powerful_stroke_of_insight.html

A evolução do ser humano permitiu que, com o aumento do "poder computacional" de seu cérebro e demais órgãos, fosse possível criar e utilizar

LINGUAGENS. Depois das palavras, vieram regras para combiná-las: as leis da GRAMÁTICA e da LÓGICA. Após isso, surgiram os números, que servem para expressar quantidades. Os números podem ser representados com os dedos (dedo = dígito). Possivelmente, vem daí o sistema numérico – o decimal (dez dedos) – universalmente aceito pelo Sistema Internacional de Unidades, ao qual estamos todos acostumados.

1.2 Sistemas de numeração

Os sistemas de numeração têm por objetivo estabelecer símbolos e convenções para representar quantidades, de forma a registrar a informação quantitativa e poder processá-la. A representação de quantidades faz-se com os **números**.

O homem utilizou diversos sistemas numéricos antes de adotar o sistema decimal. "Resquícios" de bases numéricas ancestrais persistem até hoje, como a base 60, utilizada na contagem do tempo e na trigonometria.

No estudo da Tecnologia da Informática, é de grande importância o conhecimento de alguns sistemas de numeração e códigos numéricos.

O sistema decimal é utilizado por nós no dia a dia e é, sem dúvida, o mais importante sistema numérico. Trata-se de um sistema que possui dez algarismos, com os quais podemos formar qualquer número através da lei geral de formação de números.

Veremos mais adiante os porquês da escolha do sistema binário por parte dos projetistas de computadores.

Os demais sistemas a serem estudados também são importantes, já que os sistemas digitais não trabalham com números decimais, e sim com números binários. Isto ocorre porque os dispositivos eletrônicos presentes nos circuitos digitais e nos computadores são projetados para operação em dois estados (operação binária).

Os sistemas de numeração são definidos pela "base" que eles utilizam, isto é, o número de dígitos que o sistema utiliza. Em qualquer um dos sistemas de numeração, um número é uma cadeia de "dígitos", dentro da qual cada posição tem um determinado peso dentro dessa cadeia. O valor do número é o valor da soma dos produtos dos dígitos pelo seu respectivo peso. Por exemplo, o número decimal 765 pode ser representado da seguinte forma, representado como na ilustração a seguir:

ex.:

$$(765)_{10} = 700 + 60 + 5 = 7 \cdot 10^2 + 6 \cdot 10^1 + 5 \cdot 10^0$$

Nesse exemplo, nota-se que o algarismo menos significativo (5) multiplica a unidade (1 ou 10^0), o segundo algarismo (6) multiplica a dezena (10 ou 10^1), e o mais significativo (7) multiplica a centena (100 ou 10^2). A soma dos resultados irá representar o número. Portanto, pode-se afirmar que, de maneira geral, a regra básica de formação de um número consiste no somatório de cada algarismo correspondente multiplicado pela base (no exemplo, o número 10) elevada por um índice, conforme o posicionamento do algarismo no número. Assim, um sistema de numeração genérico pode ser expresso da seguinte forma:

$$N = \ldots + a_3B^3 + a_2B^2 + a_1B^1 + a_0B^0 + a_{-1}B^{-1} + a_{-2}B^{-2} + a_{-3}B^{-3} + \ldots$$

onde:
N = valor do número obtido pela soma do dígito pelo seu peso;

a_i = dígitos do sistema;

B^i = base do sistema de numeração;

i = peso do dígito dentro da cadeia;

A BASE de um sistema de numeração é igual ao número de dígitos que o sistema utiliza. O NOME DO SISTEMA define o número de dígitos do sistema. Portanto, o SISTEMA DECIMAL, como o próprio nome diz, utiliza "10 DÍGITOS" (0, 1, 2, 3, 4, 5, 6, 7, 8 e 9) e possui "BASE 10".

1.1.1 Sistema binário

No sistema binário, os dígitos podem assumir somente dois valores possíveis que são: "0" e "1". A base desse sistema é "2". A vantagem do sistema binário reside no fato de que, possuindo apenas dois dígitos, esses são facilmente representados por uma chave aberta e uma chave fechada, ou um relé ativado e um relé desativado, ou um transistor saturado e um transistor cortado, o que torna simples a implementação de sistemas digitais mecânicos, eletromecânicos ou eletrônicos.

Em sistemas eletrônicos, o dígito binário (0 ou 1) é chamado de ***BIT***, enquanto que um conjunto de 4 *bits* é denominado ***NIBBLE***. O ***BYTE***, termo bastante utilizado principalmente na área de informática, é constituído de 8 *bits*.

a) Conversão binário → decimal

A conversão de binário para decimal é feita diretamente, somando-se os produtos dos dígitos ("0" e "1") pelo seu respectivo peso. O valor resultante fornece o número na base "10" ou no sistema decimal. Veja os exemplos a seguir:

ex.:

$(10101)_2 =$

$1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 =$

$16 + 4 + 1 =$

$(21)_{10}$

ex.:

$(1101100,1011)_2 =$

$1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 1 \cdot 2^{-4} =$

$64 + 32 + 8 + 4 + 0,5 + 0,125 + 0,0625 =$

$(108,6875)_{10}$

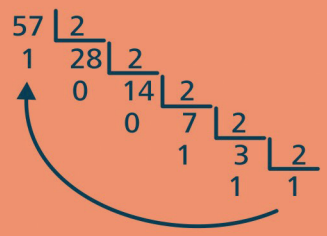
b) Conversão decimal → binário

A forma como se converte um número decimal para binário é diferente para números inteiros e números fracionários.

Para converter um número decimal inteiro em binário, aplica-se o método das divisões sucessivas. Esse método consiste em efetuar sucessivas divisões pela base a ser convertida até o último quociente possível. O número transformado será composto por este último quociente (algarismo mais significativo) e todos os restos, na ordem inversa às divisões. Dessa forma, o sentido de leitura dos restos é do último obtido para o primeiro. Veja ilustração a seguir.

ex.:

$(57)_{10} = (?)_2$



$(57)_{10} = (111001)_2$

Para obter o código binário de um número fracionário (decimal), multiplica-se por "2" o número após a vírgula. Separa-se a parte inteira do resultado obtido e multiplica-se a parte fracionária por "2" novamente. Prossegue-se o processo até que se atinja uma aproximação desejada ou até que a parte fracionária se torna nula. A sequência de "0" e "1" obtidas pelas partes

inteiras, começando pelo primeiro inteiro obtido, será a representação do número decimal fracionário no sistema binário. Veja o exemplo a seguir:

ex.:

$(0,625)_{10} = (?)_2$ →

0,625 . 2 = 1,25
0,25 . 2 = 0,5
0,5 . 2 = 1
0

$(0,625)_{10} = (0,101)_2$

Se tivermos um número decimal, como, por exemplo, $(56,625)_{10}$, deve-se converter separadamente a parte inteira e a parte fracionária nos seus números binários equivalentes e, depois, agrupar os números obtidos.

1.1.2 Sistema octal

O sistema octal possui "8" dígitos e utiliza a base "8". Os dígitos são idênticos aos dígitos do sistema decimal de 0 até 7. Esse sistema é pouco utilizado no campo da Eletrônica Digital. Trata-se apenas de um sistema numérico intermediário entre os sistemas binário e hexadecimal.

a) Conversão octal → binário

Para se converter um número octal em seu equivalente binário, cada dígito octal deve ser convertido no seu equivalente binário de três dígitos, conforme ilustrado a seguir.

ex.:

$(624)_8 = (?)_2$ →

110	010	100
6	2	4

$(624)_8 = (110010100)_2$

b) Conversão binário → octal

A conversão de binário em octal é o processo inverso ao apresentado no caso anterior, isto é, converte-se cada grupo de 3 dígitos binários pelo seu equivalente octal. Deve-se tomar cuidado para iniciar a conversão dos grupos de 3 dígitos binários sempre a partir do dígito menos significativo, conforme ilustrado a seguir.

ex.:

$(011101100110)_2 = (?)_8$ → 011|101|100|110

3 | 5 | 4 | 6

$(011101100110)_2 = (3546)_8$

c) Conversão octal → decimal

A conversão octal em decimal é feita diretamente, somando-se os produtos dos dígitos octais com o peso do dígito dentro da cadeia de numeração, conforme ilustrado a seguir.

ex.:

$(26,37)_8 = (?)_{10}$

$2 . 8^1 + 6 . 8^0 + 3 . 8^{-1} + 7 . 8^{-2} =$

$16 + 6 + 0,375 + 0,109375 =$

$(26,37)_8 = (22,484375)_{10}$

d) Conversão decimal → octal

A conversão de um número decimal (inteiro ou fracionário) em seu equivalente octal (inteiro ou fracionário) é feita de forma idêntica à conversão de decimal para binário, ou seja, através do método das divisões sucessivas, conforme ilustrado a seguir.

ex.:

$(624)_{10} = (?)_8$

624 | 8
0 78 | 8
6 9 | 8
1 1

$(624)_{10} = (1160)_8$

A-Z

software
É a parte lógica, ou seja, o conjunto de instruções e dados processado pelos circuitos eletrônicos do *hardware*. Toda interação dos usuários de computadores modernos é realizada através do *software*, que é a camada, colocada sobre o *hardware*, que trans-forma o computador em algo útil para o ser humano.

hardware
É a parte física do computador, ou seja, é o conjunto de componentes eletrônicos, circuitos integrados e placas, que se comunicam através de barramentos.

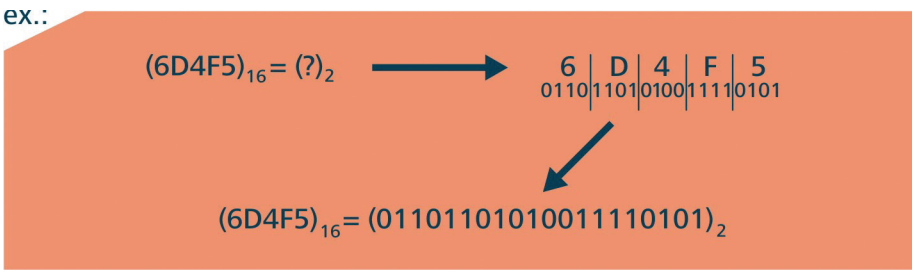
1.1.3 Sistema hexadecimal

Os números hexadecimais são amplamente utilizados para a representação de números e de dados binários na operação com micro-processadores e também no mapeamento de memórias em sistemas digitais. Trata-se de um sistema numérico muito importante, aplicado em projetos de ***software*** e ***hardware***.

Os algarismos deste sistema são enumerados da seguinte forma: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**. Nota-se que a letra A representa o algarismo **A**, que por sua vez representa a quantidade dez. O mesmo ocorre para a letra B, que representa o algarismo **B** e a quantidade onze, sucedendo assim até o algarismo **F**, que representa a quantidade quinze.

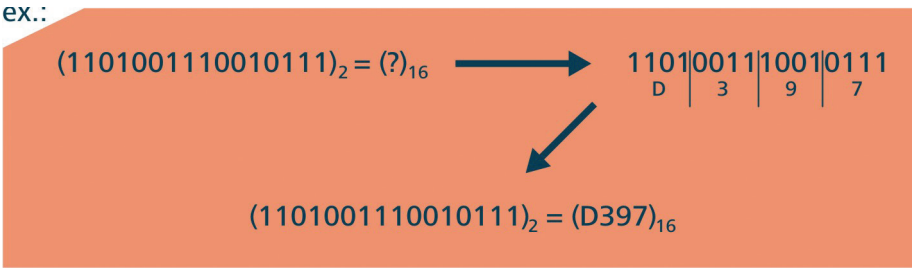
a) Conversão hexadecimal → binário

Para converter um número hexadecimal em seu equivalente binário cada dígito hexadecimal deve ser convertido no seu equivalente binário de quatro dígitos, conforme ilustrado a seguir.



b) Conversão binário → hexadecimal

A conversão de binário em hexadecimal é o processo inverso ao apresentado no caso anterior, isto é, converte-se cada grupo de 4 dígitos binários em seu equivalente hexadecimal. Da mesma forma que na conversão de binário para octal deve-se tomar cuidado para iniciar a conversão sempre a partir do dígito menos significativo. Veja o exemplo a seguir:



c) Conversão hexadecimal → decimal

A conversão hexadecimal para decimal é feita diretamente, aplicando a definição do sistema de numeração genérico na base 16, somando-se os produtos dos dígitos hexadecimais ao peso do dígito, de acordo com sua base, conforme ilustrado a seguir.

ex.:

$(A38,4D)_{16} = (?)_{10}$

$A \cdot 16^2 + 3 \cdot 16^1 + 8 \cdot 16^0 + 4 \cdot 16^{-1} + D \cdot 16^{-2} =$

$10 \cdot 16^2 + 3 \cdot 16^1 + 8 \cdot 16^0 + 4 \cdot 16^{-1} + 13 \cdot 16^{-2} =$

$2560 + 48 + 8 + 0,25 + 0,05078125 =$

$2616,30078125$

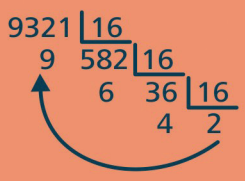
$(A38,4D)_{16} = (2616,30078125)_{10}$

d) Conversão decimal → hexadecimal

A conversão de um número decimal (inteiro ou fracionário) em seu equivalente hexadecimal (inteiro ou fracionário) é feita de forma idêntica à conversão de decimal para binário, conforme ilustrado a seguir.

ex.:

$(9321)_{10} = (?)_{16}$



$(9321)_{10} = (2469)_{16}$

1.1.4 Código "BCD" ou "8421"

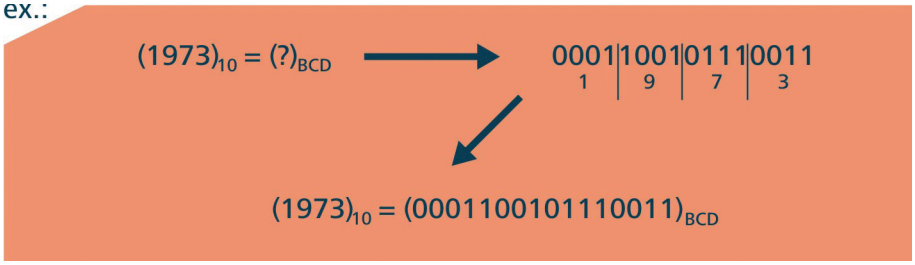
O código BCD ou 8421 significa dígitos decimais codificados em binários. Isto quer dizer que, nesse código, cada dígito decimal é substituído diretamente pelo seu equivalente binário de quatro dígitos.

Os números "BCD" são úteis sempre que a informação decimal é transferida para dentro ou para fora de um sistema digital. Os circuitos no interior de uma calculadora, por exemplo, podem processar números BCD, já que introduzimos números decimais no teclado e vemos como resposta um número decimal. Já em computadores, os números BCD têm valor e aplicação limitados, uma vez que os computadores operam com dígitos alfanuméricos.

Deve-se ressaltar que as seis últimas combinações de números binários de quatro *bits* não são utilizadas no código BCD, conforme ilustração a seguir.

Relógios binários em código BCD, veja em: http://www.glassgiant.com/geek/binaryclock/binary_clock_flash.swf

ex.:



É preciso estar atento para não confundir um número BCD com um número binário. O exemplo abaixo ilustra a diferenciação.

ex.:

$(137)_{10} = (10001001)_2$

$(137)_{10} = (000100110111)_{BCD}$

Resumo

Nessa aula, estudamos os sistemas de numeração mais utilizados em informática, bem como a conversão de um sistema em outro. Sabemos que os computadores trabalham com números de uma forma diversa da que nós trabalhamos: nós, seres humanos, utilizamos o sistema decimal, ou seja, o sistema numérico de base dez, enquanto que os computadores, por suas características constitutivas, trabalham com o sistema binário, ou seja, com base 2. No entanto, os computadores não processam somente números, mas também outros tipos de informação. E esse será o tema das próximas aulas. A seguir, é mostrada uma tabela com as equivalências entre os sistemas numéricos estudados (Tabela 1.1).

Tabela 1.1: Equivalências entre os sistemas de numeração estudados			
decimal	**hexadecimal**	**binário**	**octal**
0	0	0000	00
1	1	0001	01
2	2	0010	02
3	3	0011	03
4	4	0100	04
5	5	0101	05
6	6	0110	06
7	7	0111	07
8	8	1000	10
9	9	1001	11
10	A	1010	12
11	B	1011	13
12	C	1100	14
13	D	1101	15
14	E	1110	16
15	F	1111	17

Atividades de aprendizagem

1. Converta para o sistema decimal os seguintes números:

a) 010101_2 =

b) 1010101_2 =

c) 11011011_2 =

d) 000100010001_2 =

e) 10_8 =

f) 666_8 =

g) 1500_8 =

h) 2007_8 =

i) EAD_{16} =

j) $FACA_{16}$ =

Explore as funcionalidades de sua calculadora científica. A maioria das calculadoras científicas realiza as operações estudadas nesta aula.

2. Por que o número 2008 não pode ser octal? A quais das bases, dentre as estudadas, ele poderia pertencer?

3. Quantos *bits* existem em 4 *bytes*?

4. Quantos *bits* são necessários para representar o número EAD_{16}?

Aula 2 – A arquitetura de um computador

"Hardware é aquilo que você chuta,
Software é aquilo que você xinga."

Ditado popular

Objetivos

Compreender o funcionamento básico da arquitetura de Von Neumann.

Relacionar as diferentes partes constitutivas de um computador que obedece ao modelo de Von Neumann.

2.1 Modelo de Von Neumann

Von Neumann propôs construir computadores que:

1. codificassem instruções que pudessem ser armazenadas na memória e sugeriu que se usassem cadeias de uns e zeros (binário) para codificá-los;

2. armazenassem na memória as instruções e todas as informações necessárias para a execução da tarefa desejada;

3. ao processarem o programa, as instruções fossem buscadas diretamente na memória.

Saiba mais sobre Von Neumann em: http://pt.wikipedia.org/wiki/John_von_neumann

Dessa forma, um computador típico possui três componentes básicos, conforme Quadro 2.1:

Quadro 2.1: Componentes básicos de um computador

Unidade Central de Processamento (UCP ou CPU, como é mais conhecida);
Memória Principal;
Sistema de Entrada e Saída.

A CPU exerce o controle do computador; é responsável pela **busca** das instruções (as quais estão em sequência), pela sua **decodificação** (ou **interpretação**) e **execução**. A busca e a decodificação das instruções são realizadas pela **Unidade de Controle**, enquanto que a execução fica ao encargo da **Unidade Operativa**.

A unidade operativa, por sua vez, é composta pela Unidade **Lógica** e **Aritmética** e por um conjunto de **Registradores** de uso genérico. A **Memória Principal** armazena as **instruções** e os **dados** a serem processados pela CPU. Por fim, o **Sistema de Entrada e Saída – E/S** (*I/O – Input/Output system,* em inglês) tem como função conectar o computador ao meio externo, a fim de torná-lo verdadeiramente útil ao ser humano. Observe a Figura 2.1 a seguir.

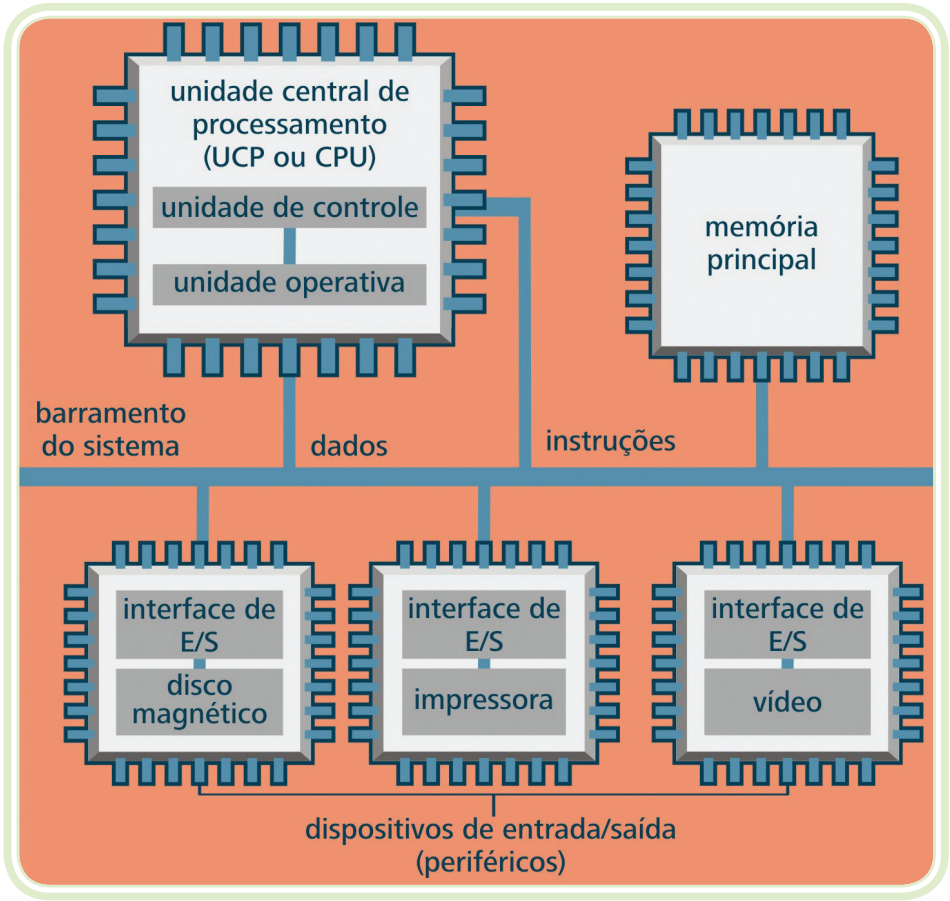


Figura 2.1: Diagrama de blocos de um computador com arquitetura convencional
Fonte: CTISM

A-Z

periféricos
São aparelhos ou placas que enviam ou recebem informações do computador. Em informática, o termo "periférico" aplica-se a qualquer equipamento acessório que seja ligado à CPU (unidade central de processamento), ou, um sentido mais amplo, o computador.

O sistema de entrada e saída é composto por um meio físico de conexão, chamado barramento (*bus*), e um conjunto de dispositivos de entrada e saída, chamados periféricos. Os **periféricos**, geralmente são sistemas mistos, do tipo eletromecânico, que permitem ao usuário entrar com os dados ou, ainda, obter ou visualizar os resultados. Podemos citar alguns exemplos de periféricos: teclado, vídeo, impressora, *mouse*, unidades de disco flexível e memórias *flash*, unidades de disco rígido (*winchester*), *modem*, placa de som, *scanner*, *pen drives* e cartões de memória. Alguns periféricos permitem somente a entrada de dados (teclado), outros só permitem a saída (vídeo e

impressora) e outros, ainda, permitem tanto a entrada como a saída (unidades de discos magnéticos e *pen drives*).

Associado a cada periférico existente num computador, há um circuito específico de controle de interface de E/S (*I/O interface processor*), cuja função é adaptar os sinais gerados pelo periférico conforme as necessidades do computador. Veja a Figura 2.2 a seguir.

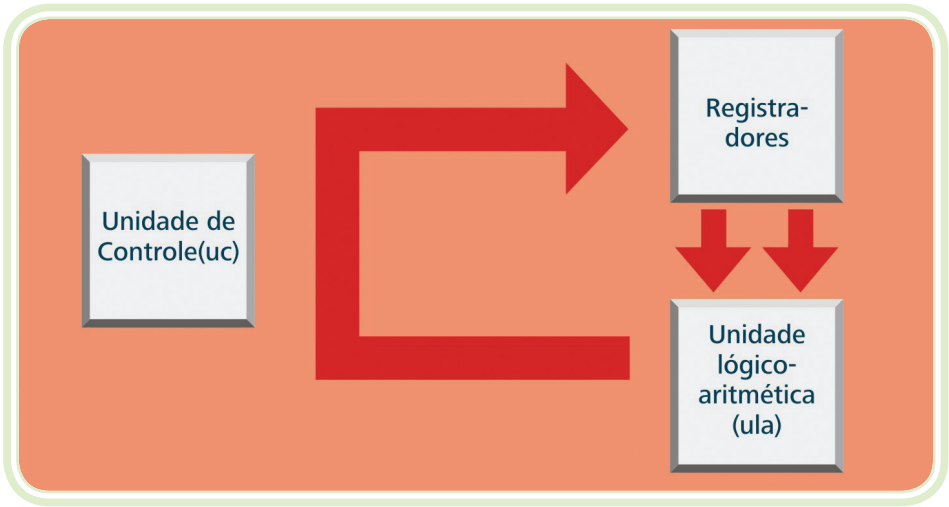


Figura 2.2: Subunidades da unidade central de processamento (CPU)
Fonte: CTISM

O diagrama de blocos da Figura 2.1 representa o modelo convencional de um computador, também conhecido como **modelo** (ou **arquitetura) de Von Neumann**, por ser descendente direto do computador desenvolvido em 1946 por Von Neumann e sua equipe. A maior parte dos computadores ainda hoje (2009) apresenta traços dessa arquitetura. Suas principais características são:

Quadro 2.2: Características dos computadores

- possuem uma unidade de processamento central, para a execução de operações lógicas e aritméticas;
- possuem uma unidade de controle de programa, a qual determina o sequenciamento das instruções a serem executadas por meio de sinais de controle;
- as instruções dos programas são armazenadas de maneira sequencial, facilitando sua busca;
- possuem registradores dedicados ao armazenamento dos operandos e dos resultados das operações;
- têm unidade de armazenamento central, na qual são guardados programas e dados de forma compartilhada;
- possuem um único barramento do sistema, o qual deve ser usado de forma compartilhada para a transferência de dados e instruções entre os diversos blocos.

2.2 Tipos de instruções

As funções possíveis de serem executadas pela CPU estão definidas no seu conjunto de instruções. Um computador típico possui entre 50 e 200 instruções distintas, que podem ser divididas em três grupos, conforme sua natureza:

- Instruções de transferência de dados;
- Instruções de processamento de dados;
- Instruções de controle.

As instruções lógicas, ou portas lógicas, são operações matemáticas baseadas na álgebra de Booble. Portas lógicas é assunto para a próxima etapa do curso de Automação Industrial. Para familiarizar-se ao assunto, acesse o *link*: http://www.users.rdc.puc-rio.br/rmano/ptlog.html

As **instruções de transferência de dados** apenas movem as informações, sem alterar seu conteúdo. As transferências podem ocorrer dentro da CPU, entre a CPU e a memória principal, entre algum periférico e a CPU, ou entre algum periférico e a memória principal. As **instruções de processamento de dados** transformam as informações, utilizando para isso os recursos de *hardware* disponíveis na unidade operativa da CPU. Neste grupo, encontram-se as instruções **aritméticas**, tais como adição, subtração e multiplicação, e as instruções **lógicas**, tais como adição lógica (ou), multiplicação lógica (e), complementação (não ou) e ou-exclusivo. As **instruções de controle** determinam a sequência segundo a qual as instruções são executadas, permitindo que o controle seja transferido de uma parte do programa para outra, ou entre diferentes subprogramas. Exemplos de instruções desse tipo são *jump* (salto), chamadas de sub-rotina e retorno de sub-rotina.

A unidade de controle é responsável por controlar o endereço da memória principal no qual estão armazenadas as instruções. Para tanto, existe um registrador especial, denominado **contador de programa** (*program counter* – PC), cuja função é armazenar o endereço no qual está armazenada a instrução que está sendo executada. Como normalmente as instruções são armazenadas e carregadas em sequência, a operação mais comum realizada sobre o conteúdo do PC é a "soma um" (incremento).

Existem ainda outros registradores que facilitam o acesso às instruções e aos dados. Por exemplo, uma região contínua da memória, denominada pilha (*stack*), é utilizada na transferência do controle do sistema (computador) entre subprogramas. O apontador de pilha (*stack pointer* – SP) é um registrador usado no controle da posição de memória para colocar/retirar dados do topo da pilha.

ferrite
É um material ferromagnético, composto de ferro, boro, bário, estrôncio ou molibdênio. O Ferrite tem alta permeabilidade magnética, porém não retém magnetismo.

2.3 Memórias

2.3.1 Tecnologias

As primeiras tecnologias utilizadas em memórias foram as memórias de núcleos magnéticos (**ferrite**). As memórias modernas são compostas por cir-

cuitos semicondutores, com novas tecnologias sendo criadas a cada ano, permitindo que grandes quantidades de células de memória sejam encapsuladas em pequenas pastilhas.

2.3.2 Hierarquia de memória

A Memória Principal (MP) não é o único dispositivo de armazenamento de um computador. Em função de características como tempo de acesso, capacidade de armazenamento, custo, etc., há uma hierarquia de dispositivos de memória em computadores. Observe a Quadro 2.3 a seguir:

Quadro 2.3: Características dos diversos tipos de memória

Tipo	Capacidade	Velocidade	Custo	Localização	Volatilidade
registrador	*bytes*	muito alta	muito alto	UCP	volátil
memória cache	*Kbytes*	alta	alto	UCP/placa	volátil
memória principal	*Mbytes*	média	médio	placa	volátil
memória auxiliar	*Gbytes*	baixa	baixo	externa	não volátil

A UCP vê nessa ordem e acessa primeiro a memória que está mais próxima. Verificamos no diagrama abaixo (Figura 2.3) que, quanto mais próxima da UCP, maior é a velocidade, maior custo, porém, menor capacidade de armazenamento.

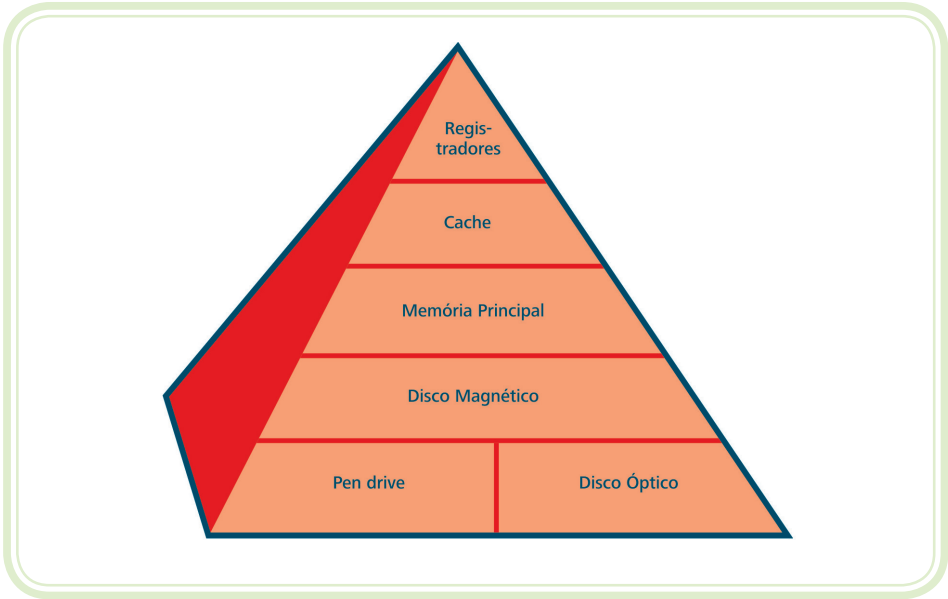


Figura 2.3: Hierarquia da memória de 5 níveis
Fonte: CTISM

a) Registradores

Registradores são dispositivos de armazenamento temporário, localizados na UCP, extremamente rápidos, com capacidade para apenas um dado (uma palavra). Devido a sua tecnologia de construção e por estar localizado na própria pastilha ("***chip***") da UCP, são muito caros.

A-Z

chip
É um dispositivo microeletrônico que consiste de muitos transistores e outros componentes interligados, capazes de desempenhar muitas funções. Suas dimensões são extremamente reduzidas; os componentes são formados em pastilhas de material semicondutor de silício. A importância da integração está no baixo custo, no alto desempenho e no tamanho reduzido dos circuitos aliado à alta confiabilidade e estabilidade de funcionamento.

O conceito de registrador surgiu da necessidade de a UCP armazenar temporariamente dados intermediários durante um processamento. Isso ocorre, por exemplo, quando um dado resultado de operação precisa ser armazenado até que o resultado de uma busca da memória esteja disponível para com ele realizar uma nova operação.

Registradores são VOLÁTEIS, isto é, dependem de estar energizados para manter armazenado seu conteúdo.

b) Memória cache

Com o desenvolvimento da tecnologia de construção da UCP, as velocidades foram ficando muito mais altas que as das memórias, as quais não tiveram a mesma evolução de velocidade. Pelas mais variadas razões, o aperfeiçoamento das memórias foi maior no fator capacidade. Dessa forma, os tempos de acesso às memórias foram ficando insatisfatórios e a UCP, ao buscar um dado na memória, precisa ficar esperando muitos ciclos de máquina até que a memória retorne o dado buscado (*"wait states"*), configurando um gargalo (*"bottleneck"*) ao desempenho do sistema.

Por esse motivo, desenvolveram-se outras arquiteturas de memória privilegiando a velocidade de acesso. A arquitetura da memória ***cache*** é muito diferente da arquitetura da memória principal, e o acesso a ela é, muitas vezes, mais rápido. No entanto, o custo de fabricação da memória *cache* é muito maior que o da memória principal. Dessa forma, não é economicamente viável construir um computador somente com tecnologia de memória *cache*. Criou-se, então, um artifício, incorporando-se ao computador uma pequena porção de memória *cache*, localizada entre a UCP e a MP (memória principal), que funciona como um espelho de parte da MP.

A-Z

cache
É um termo do inglês que deriva do francês "*cacher*" (esconder, em português). Na prática, para os demais componentes do sistema, tudo se passa como se não houvesse *cache*, já que nenhum deles, exceto o subsistema que controla o acesso à memória, toma conhecimento de sua existência. Os dados nele armazenados são fornecidos como se proviessem diretamente do local de armazenamento original, apenas mais rapidamente. É essa a razão do nome, já que as coisas se passam como se os dados estivessem "escondidos" na *cache*.

Desenvolveram-se, ainda, algoritmos que fazem com que, a cada momento, a memória cache armazene a porção de código ou dados (por exemplo, uma sub-rotina) que estão sendo usados pela UCP. Essa transferência (MP <--> Cache) é feita pelo *hardware*: ela independe do *software*, que ignora se existe ou não memória *cache*, portanto ignora essa transferência; nem

o programador nem o sistema operacional tem controle sobre essa movimentação.

Memórias cache também são VOLÁTEIS, isto é, dependem de estar energizadas para manter gravado seu conteúdo.

A memória *cache* opera em função de um princípio estatístico comprovado: em geral, os programas tendem a referenciar várias vezes pequenos trechos de programas, como *loops*, sub-rotinas, funções e só tem sentido porque programas executados linearmente são raros. Desta forma, algoritmos (chamados algoritmos de *cache*) podem controlar qual parte do código ficará copiada na *cache* a cada momento.

c) Memórias auxiliares

Memórias auxiliares resolvem problemas de armazenamento de grandes quantidades de informações. A capacidade da MP é limitada pelo seu custo relativamente alto, enquanto que as memórias auxiliares têm maior capacidade e menor custo. Dessa forma, o custo por *bit* armazenado é muito menor. Outra vantagem importante é que as memórias auxiliares não são VOLÁTEIS, isto é, não dependem de estar energizadas para manter gravado seu conteúdo.

Os principais dispositivos de memória auxiliar são: discos rígidos (ou HD), *drives* de disquete, unidades de fita, CD-ROM, DVD, unidades ótico-magnéticas, *pendrives* e cartões de memória.

Cache de disco não é a mesma tecnologia da memória *cache*. Trata-se do emprego do mesmo **conceito** da memória *cache*, para acelerar a transferência de dados entre disco, MP e UCP, usando um programa (um *software*, por ex.: *SmartDrive*) para manter um espelho do conteúdo de parte do disco (a mais provável de ser requisitada a seguir pela UCP) gravado em uma parte da Memória Principal. Recentemente, as unidades de disco passaram a incorporar em sua interface *chips* de memória para acelerar a transferência de dados, utilizando um algoritmo de *cache*.

d) Memória principal

Conforme definimos anteriormente, a memória principal é a parte do computador em que programas e dados são armazenados para processamento. A informação permanece na memória principal apenas enquanto necessário para seu emprego pela UCP e, após isso, a área de MP ocupada pela informação é liberada para ser, posteriormente, sobregravada por outra informação. Quem controla a utilização da memória principal é o sistema operacional.

A memória precisa ter uma organização que permita ao computador guardar e recuperar informações quando necessário. Portanto, não basta transferir informações para a memória. É preciso haver uma maneira de encontrar essa informação mais tarde, quando ela for necessária e, para isso, é preciso haver um mecanismo que registre exatamente onde a informação foi armazenada.

Sendo assim, a memória principal é organizada em células. Célula é a menor unidade da memória que pode ser endereçada (não é possível buscar uma "parte" da célula) e tem um tamanho fixo (para cada máquina). As memórias são compostas por um determinado número de células ou posições. Cada célula é composta por um determinado número de *bits*. Todas as célu-

las de um dado computador têm o mesmo tamanho, isto é, todas as células daquele computador terão o mesmo número de *bits*.

Uma célula não significa o mesmo que uma palavra; uma célula não contém necessariamente uma palavra. Palavra é a unidade de processamento da UCP. Uma palavra deve representar um dado ou uma instrução, que poderia ser processada, armazenada ou transferida em uma única operação. No entanto, em geral não é assim que acontece e os computadores comerciais não seguem um padrão único para a organização da UCP e MP. Computadores comerciais (tais como, por exemplo, os baseados nos processadores Intel 486) podem ter o tamanho da palavra definido como de 32 *bits*, porém sua estrutura de memória tem células de 16 *bits*.

Cada célula é identificada por um endereço único, pelo qual é referenciada pelo sistema e pelos programas. As células são numeradas sequencialmente, uma a uma, de 0 a (N-1); esse número é chamado de endereço da célula. Endereço é o localizador da célula, que permite identificar, sem sombra de dúvida, uma célula. Assim, cada célula pode ser identificada pelo seu endereço.

2.3.3 Classificação das memórias

Quanto à leitura e à escrita, as memórias podem ser classificadas como:

R/W – *Read and Write* (memória de leitura e escrita), comumente (e impropriamente) é chamada de RAM (*Random Access Memory* ou memória de acesso aleatório), embora não seja a única RAM.

Esta memória permite operações de escrita e leitura pelo usuário e pelos programas. Seu tempo de acesso independe do endereço acessado. É construída com tecnologia de semicondutores (bipolar, CCD), pode ser estática (SRAM) ou dinâmica (DRAM) e é volátil. A MP é construída com memória R/W.

ROM – *Read Only Memory* ou Memória apenas de Leitura

Esta memória permite apenas a leitura e, uma vez gravada, não pode mais ser alterada. Também é de acesso aleatório (isto é, é também uma RAM), mas não é volátil. É utilizada geralmente por fabricantes para gravar programas que não devem ser alterados ou apagados acidentalmente (tal como a BIOS – *Basic Input Output System* – e microprogramadores de memórias de controle). A ROM é mais lenta que a R/W e é barata, porém o processo produtivo depende de ser programada.

PROM – *Programmable Read Only Memory* ou Memória apenas de Leitura Programável

Esta memória é um ROM programável (em condições e com máquinas adequadas, chamadas queimadores de PROM) e geralmente é comprada "virgem" (sem nada gravado); é muito utilizada no processo de testar programas no lugar da ROM, ou sempre que se queira produzir ROM em quantidades pequenas. Uma vez programada (em fábrica ou não), não pode mais ser alterada.

EPROM – *Erasable Programmable Read Only Memory* ou Memória apenas de Leitura, Programável e Eletronicamente Alterável

Também chamada EAROM (*Electrically Alterable ROM*). Esta memória é um PROM apagável para o processo eletrônico, sob controle da UCP, com equipamento e programas adequados. É mais cara e geralmente utilizada em dispositivos aos quais se deseja permitir a alteração, possibilitando a carga de novas versões de programas à distância ou, então, para possibilitar a reprogramação dinâmica de funções específicas de um determinado programa, geralmente relativas ao *hardware*.

2.4 Unidade central de processamento

A Unidade Central de Processamento – UCP (*Central Processing Unity* – CPU) – é a responsável pelo processamento e pela execução dos programas armazenados na MP. As funções da UCP compreendem executar as instruções e controlar as operações no computador. A UCP é composta de duas partes:

ULA ou UAL – Unidade Lógica e Aritmética – tem por função a efetiva execução das instruções;

UC – Unidade de Controle Principal – tem por funções a busca, a interpretação, o controle de execução das instruções e o controle dos demais componentes do computador. Veja a Figura 2.4.

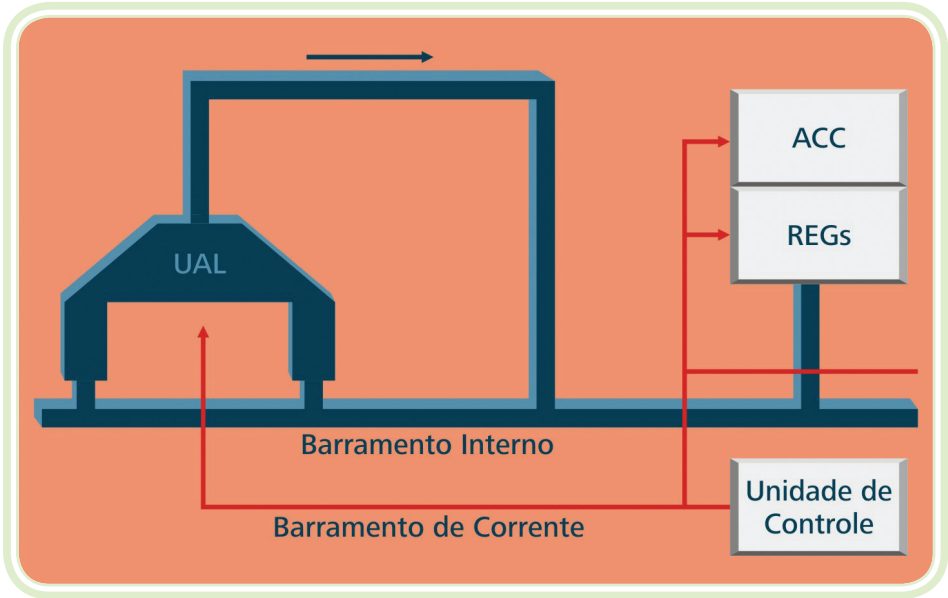


Figura 2.4: Diagrama esquemático da UAL
Fonte: CTISM

A seguir, é apresentado o diagrama esquemático de uma UCP (Figura 2.5).

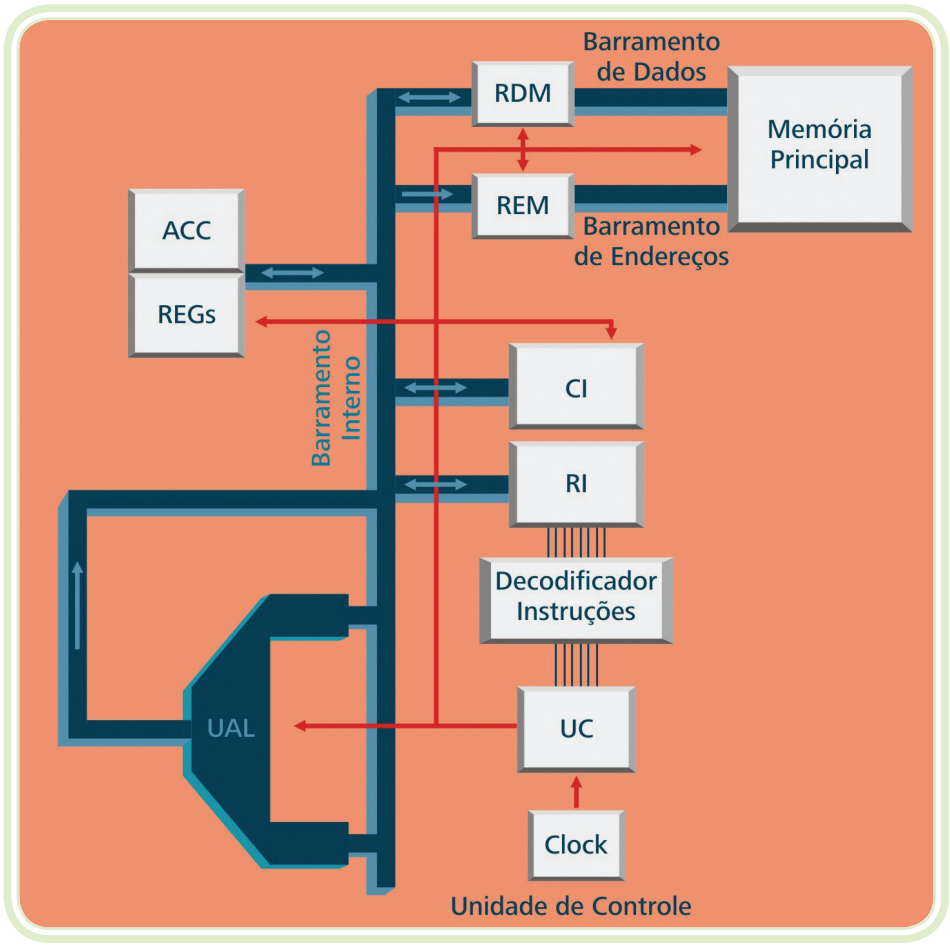


Figura 2.5: Diagrama principal da CPU
Fonte: CTISM

Registradores importantes na UCP:

- Na **UC, CI (Contador de Instruções, PC – *Program Counter)*** – é armazenado o endereço da próxima instrução a ser executada.

- Na **UC, RI (Registrador de Instrução, IR – *Instruction Register*)** – é armazenada a instrução a ser executada.

- Na **UAL, ACC (Acumulador, ACC – *Accumulator*)** – são armazenados os dados (de entrada e resultados) para as operações na ULA; o acumulador é um dos principais elementos que definem o tamanho da palavra do computador. O tamanho da palavra é igual ao tamanho do acumulador.

Os dados são representados, no barramento, na forma de sinais de tensão, sendo que um sinal de tensão de uns poucos volts ("*high*") representa o *bit* "1", e um sinal próximo de zero volts ("*low*") representa o *bit* "0".

2.4.1 Comunicação entre memória principal e UCP

a) Barramentos

Os diversos componentes dos computadores comunicam-se através de barramentos. Um barramento é um conjunto de condutores elétricos que interligam os diversos componentes de um computador entre si. Para um dado ser transportado de um componente a outro, é preciso emitir os sinais de controle necessários para o componente-origem colocar o dado no barramento e para o componente-destino ler o dado do barramento. Como um dado é composto por *bits* (geralmente um ou mais *bytes*), o barramento deverá ter tantas linhas condutoras quantos forem os *bits* a serem transportados de cada vez.

Assim, se quisermos transferir um *byte* – por exemplo, 01100100 – da UCP para a memória principal, os circuitos de controle encarregam-se de colocar o *byte* 01100100 no barramento, ou seja, colocarão sinais de tensão *"high"* nas 3ª, 6ª e 7ª linhas do barramento (por convenção, os *bits* são sempre ordenados da direita para a esquerda) e informarão a memória para ler o dado no barramento.

A comunicação entre MP e UCP usa dois registradores da UCP chamados de Registrador de Endereços de Memória – **REM** – ou, em inglês, *Memory Address Register* (MAR), bem como o Registrador de Dados da Memória – RDM – ou, em inglês, *Memory Buffer Registe*r (MBR). Veja a Figura 2.6.

Tempo de ciclo (ou ciclo de memória) é conceituado como o tempo decorrido entre dois ciclos sucessivos de acesso à memória. As memórias dinâmicas perdem seu conteúdo em alguns instantes e dependem de periódica atualização (ciclo de *"refresh"*). No caso das SRAMs (*Static RAM* ou memórias estáticas), que não dependem de *"refresh"*, o tempo de ciclo é igual ao tempo de acesso. As memórias dinâmicas, no entanto, requerem ciclos periódicos de *"refresh"*, o que faz com que a memória fique indisponível para novas transferências a intervalos regulares necessários para os ciclos de *"refresh"*. Assim, as memórias DRAM têm ciclo de memória maior que o tempo de acesso.

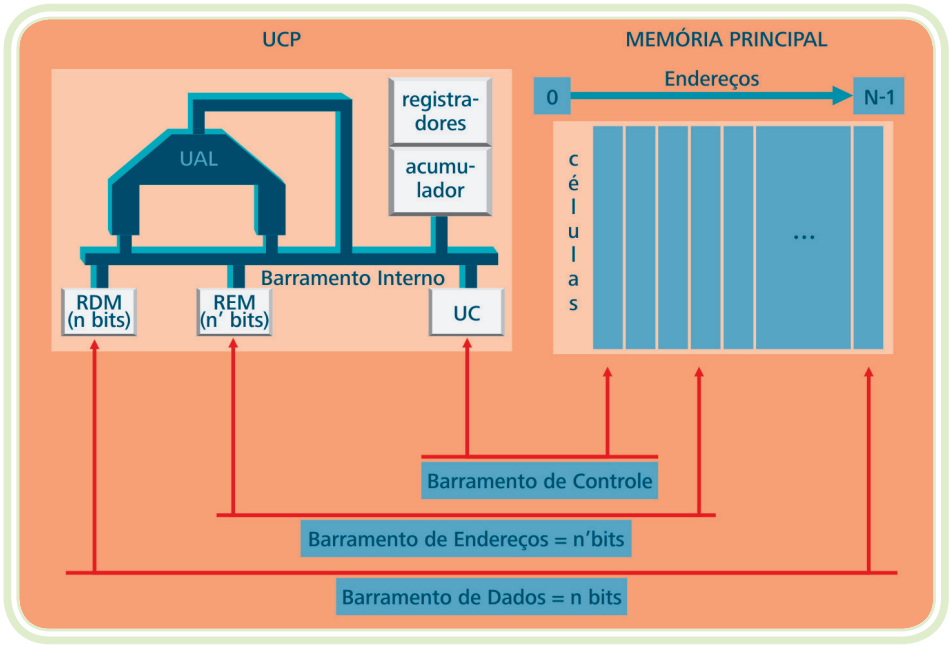


Figura 2.6: Esquema de funcionamento da comunicação MP / UCP
Fonte: CTISM

- **UCP / MP**

Barramento de endereços – unidirecional (só a UCP envia dados – *write* – ou lê dados – *read* – da MP);

Barramento de dados – bidirecional;

Barramento de controle – bidirecional.

- **Registradores**

REM – Registrador de Endereços de Memória (MAR – *Memory Address Register*);

RDM – Registrador de Dados de Memória (MBR – *Memory Buffer Register*).

b) Palavra (unidade de informação)

Palavra é a unidade de informação do sistema UCP / MP. A conceituação mais usada define palavra como capacidade de manipulação de *bits* do núcleo do computador (UCP e MP). Pressupõe-se, assim, que todos os elementos do núcleo do computador (o que inclui o tamanho da UAL e do acumulador e registradores gerais da UCP e o barramento de dados) tenham a mesma largura (processem simultaneamente o mesmo número de *bits*), o que nem sempre acontece. Muitas vezes, encontram-se computadores em que o tamanho da UAL e do acumulador (e registradores gerais) não corresponde

ao tamanho dos barramentos. Dessa forma, encontram-se especificações de "computadores de 64 *bits*", mesmo quando seu barramento de dados é de 32 *bits*, nesse caso, referindo-se exclusivamente à capacidade de manipulação da UCP de 64 *bits* (isto é, sua UAL e seu acumulador têm 64 *bits*). Essa conceituação é imprecisa, às vezes, enganosa e pode levar a erros de avaliação da capacidade de processamento de um computador. Portanto, deve-se analisar caso a caso, porque a simples menção ao tamanho da palavra não é uma terminologia que permite definir, de forma conclusiva, a arquitetura do computador.

2.4.2 Tempo de acesso

Tempo de acesso (ou tempo de acesso para leitura) é o tempo decorrido entre uma requisição de leitura de uma posição de memória e o instante em que a informação requerida está disponível para utilização pela UCP. Ou seja, é o tempo que a memória consome para colocar o conteúdo de uma célula no barramento de dados. O tempo de acesso de uma memória depende da tecnologia da memória.

O tempo de acesso de qualquer memória tipo RAM (*Random Access Memory* ou memória de acesso aleatório) é independente do endereço a ser acessado (a posição de memória a ser escrita ou lida), isto é, o tempo de acesso é o mesmo, qualquer que seja o endereço acessado.

2.4.3 Funcionamento

A MP pode ser acessada através de duas operações: Ler ou Escrever. Ler da memória significa requisitar à MP o conteúdo de uma determinada célula (recuperar uma informação). Essa operação de recuperação da informação armazenada na MP consiste na transferência de um conjunto de *bits* (cópia) da MP para a UCP e é não destrutiva, isto é, o conteúdo da célula não é alterado. Seu sentido é da MP para a UCP.

Passos executados pelo *hardware*:

- A UCP armazena no REM o endereço onde a informação requerida está armazenada;
- A UCP comanda uma leitura;
- O conteúdo da posição identificada pelo endereço contido no REM é transferido para o RDM e fica disponível para a UCP.

Escrever na memória significa escrever uma informação em uma célula da MP (armazenar uma informação). Essa operação de armazenamento da informação na MP consiste na transferência de um conjunto de *bits* da UCP para a MP e é destrutiva (isso significa que qualquer informação que estiver gravada naquela célula será sobregravada). Seu sentido é da UCP para a MP.

2.4.4 Lógica temporizada

Conforme vimos ao analisar a comunicação entre UCP e memória, as instruções, os dados e os endereços "trafegam" no computador através dos barramentos (de dados, de endereços e de controle), sob a forma de *bits* representados por sinais elétricos: uma tensão positiva alta (*"high"* – geralmente entre 2,2V e 5V, dependendo do processador) significando "1", e uma tensão baixa (*"low"* – próxima de zero) significando "0". Mas, os dados no computador não ficam estáticos; pelo contrário, a cada ciclo (cada "estado") dos circuitos, os sinais variam de forma a representar novas instruções, dados e endereços. Ou seja, os sinais ficam estáticos apenas por um curto espaço de tempo, necessário e suficiente para os circuitos poderem detectar os sinais presentes no barramento, naquele instante e reagir de forma apropriada. Assim, periodicamente, uma nova configuração de *bits* é colocada nos circuitos e tudo isso só faz sentido se pudermos, de alguma forma, organizar e sincronizar essas variações, de modo que, num dado instante, os diversos circuitos do computador possam "congelar" uma configuração de *bits* e processá-las. Para isso, é preciso que exista outro elemento que forneça uma base de tempo para que os circuitos e os sinais se sincronizem. Este circuito é chamado ***clock*** – o **relógio interno** do computador. Cada um dos diferentes estados que os circuitos assumem, limitados pelo sinal do *clock*, é chamado **ciclo de operação**.

A-Z

clock
É a frequência de trabalho do processador, indica quantas operações por segundo serão executadas pelo processador.

A Unidade de Controle da UCP envia a todos os componentes do computador um sinal elétrico regular – o pulso de *"clock"* – que, por sua vez, fornece uma referência de tempo para todas as atividades e permite o sincronismo das operações internas. O *clock* é um pulso alternado de sinais de tensão, gerado pelo circuito de relógio, composto de um cristal oscilador e circuitos auxiliares.

Cada um dos intervalos regulares de tempo definidos pelo *clock* é delimitado pelo início da descida do sinal, equivalendo um ciclo à excursão do sinal por um *"low"* e um *"high"* do pulso. O tempo do **ciclo** equivale ao período da oscilação. Sabemos que período é o inverso da frequência. Ou seja:

$$P = 1 / f$$

A frequência **f** do *clock* é medida em *Hertz*. Inversamente, a duração de cada ciclo é chamada de período. Por exemplo, se f = 10 Hz, temos que P = 1/10 = 0,1 s (Figura 2.7).

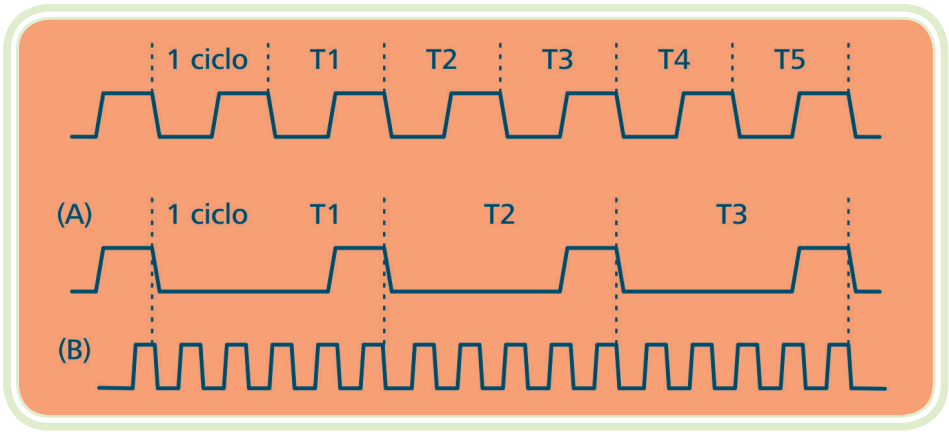


Figura 2.7: Diagrama temporal
Fonte: CTISM

Um MHz (1 *megahertz*) equivale a um milhão de ciclos por segundo. Sendo a frequência de um processador medida em *megahertz* ou em *gigahertz*, o período será, então, medido em nanossegundos, como vemos no exemplo na Quadro 2.4:

Quadro 2.4: Medidas

f = 10 MHz = 10 x 10^6 Hz
P = 1/10.10^6 = 100 ns (100 nanosegundos)
f = 1 GHz = 1 x 10^9 Hz
P = 1/10^9 = 1 ns (1 picosegundo)

Sempre que se fala sobre máquinas velozes, citam-se números em *gigahertz*. Para um melhor entendimento sobre o que ocorre na máquina, em vez de falar sobre a frequência do *clock*, seria mais ilustrativo discutir outra grandeza: o período, que pode ser definido como o tempo de duração de cada ciclo ou simplesmente tempo de ciclo.

Se tivermos dois processadores operando em um mesmo *clock*, poderemos ter desempenhos completamente diferentes, visto que o desempenho do processador deve ser analisado por um conjunto de características da arquitetura, do qual a frequência do *clock* é apenas um deles – e não o mais importante.

Quando se diz que um processador é de 2 GHz, está se definindo a frequência de operação de seu processador (seu *clock*), significando que o processador pode alternar seus estados internos 2 bilhões de vezes por segundo. Isto acarreta que cada ciclo (equivalente a um estado lógico) de operação dura 1 / 2.000.000.000 s = 0,5 x 10^{-9} s, ou seja, 0,5 picosegundo.

Os primeiros computadores tinham um único sinal de *clock* geral, válido para UCP, memória, barramentos de E/S (entrada /saída), etc. Na medida em

que a tecnologia foi se aperfeiçoando, a frequência de *clock* de operação dos processadores (e, em menor escala, também a das memórias) aumentou em uma escala muito maior que a dos demais componentes. Dessa forma, foi necessário criar diferentes pulsos de *clock* para acomodar as frequências de operação dos diferentes componentes. A placa-mãe de um PC utiliza uma frequência-mestra para seu barramento (ciclo de barramento), a qual é multiplicada ou dividida para ser utilizada pelos demais componentes. Por exemplo, o processador tem essa frequência multiplicada por 2 a 4, e as cache secundárias (ciclos entre 10 e 20 ns) usam a própria frequência da placa-mãe. As memórias *cache* primárias, por sua vez, são hoje construídas como parte do processador e usam o mesmo *clock* do processador.

O efeito prático do aumento da frequência de operação é que a precisão de fabricação dos circuitos também precisa ser maior. O tamanho de cada junção de transistor fica menor. Uma junção menor requer menos potência para sua operação, menos elétrons para operar uma transição de estados, menor tempo de propagação do sinal e, consequentemente, menor potência dissipada.

2.5 Instruções

Para que um programa possa ser executado por um computador, ele precisa ser constituído por uma série de instruções de máquina e estar armazenado em células sucessivas na memória principal. A UCP é responsável pela execução das instruções que estão na memória.

Quem executa um programa é o *hardware* e o que ele espera encontrar é um programa em linguagem de máquina, ou seja, uma sequência de instruções de máquina em código binário. A linguagem de máquina é composta por códigos binários, representando instruções, endereços e dados, e está totalmente vinculada ao conjunto (*"set"*) de instruções da máquina.

Um ser humano usa seu conhecimento e inteligência para traduzir uma tarefa complexa (tal como, por exemplo, a tarefa de buscar uma pasta num arquivo) numa série de passos elementares (identificar o móvel e a gaveta em que está a pasta, andar até o móvel, abrir a gaveta, encontrar a pasta, retirar a pasta e fechar a gaveta). Para o computador, uma instrução precisa ser detalhada, dividida em pequenas etapas de operações, que são dependentes do conjunto de instruções do computador e individualmente executáveis.

Fazendo um paralelo com linguagens de alto nível, o programa elaborado pelo programador (o código-fonte composto de instruções complexas) precisa ser "traduzido" em pequenas operações elementares (primitivas), executáveis pelo *hardware*. Cada uma das instruções tem um código binário associado, que é o código da operação.

2.5.1 Formato geral de uma instrução

O formato geral de uma instrução segue o definido a seguir:

a) **Código de operação ou OPCODE** – é o campo da instrução cujo valor binário identifica a operação a ser realizada pelo processador. Cada instrução deverá ter um código único que a identifique.

b) **Operando(s)** – é(são) o(s) campo(s) da instrução cujo valor binário sinaliza a localização do dado (ou é o próprio dado) que será manipulado (processado) pela instrução durante a operação. Em geral, um operando identifica o endereço de memória em que está contido o dado que será manipulado, ou pode conter o endereço no qual o resultado da operação será armazenado. Finalmente, um operando pode também indicar um registrador, que conterá o dado propriamente dito ou um endereço de memória em que está armazenado o dado. Os operandos fornecem os dados da instrução. Por exemplo, uma instrução típica de um programa em linguagem *assembly* de um microcontrolador PIC de 8 *bits* é:

```
movwf  0 x 05
```

Sendo "movwf" o código da operação, que consiste num mnemônico e representa mover o conteúdo do acumulador "w" para o endereço "f", e "0x05" é o operando que representa o endereço de memória "f". Dessa forma, ao final da execução dessa linha do programa, o dado que está no acumulador "w" será transferido para o endereço "0x05" na memória.

Os mnemônicos possibilitam escrever código de um modo muito mais intuitivo, e sem perda de precisão. Mnemônica, de acordo com o Dicionário Aulete Digital, é uma técnica para facilitar a memorização através de exercícios como, por exemplo, a combinação de elementos associáveis.

2.5.2 Conjunto de instruções

Quando se projeta um *hardware*, define-se o seu conjunto (*"set"*) de instruções, ou seja, o conjunto de instruções elementares que o *hardware* é capaz de executar. O projeto de um processador é centrado no seu conjunto (*"set"*) de instruções.

Funcionalmente, um processador precisa possuir instruções para realizar os comandos:

Operações matemáticas sendo:

- Aritméticas: +, -, x, ÷;
- Lógicas: *and*, *or*, *xor*;
- Relacionais: >, <, =;
- Operações de movimentação de dados (memória <--> UCP, reg <--> reg);
- Operações de entrada e saída (leitura e escrita em dispositivos de E/S);
- Operações de controle (desvio de sequência de execução, parada).

2.5.3 Ciclo de instrução

Em alguns computadores (usando uma abordagem que visa à redução de custos), os dados podem ser transportados usando mais de um ciclo de barramento.

As instruções são executadas sequencialmente (a não ser pela ocorrência de um desvio), uma a uma.

O ciclo de instrução indica a sequência de execução, isto é, ele controla o fluxo de execução das instruções. A seguir, é ilustrado o ciclo de processamento de uma instrução (fluxograma).

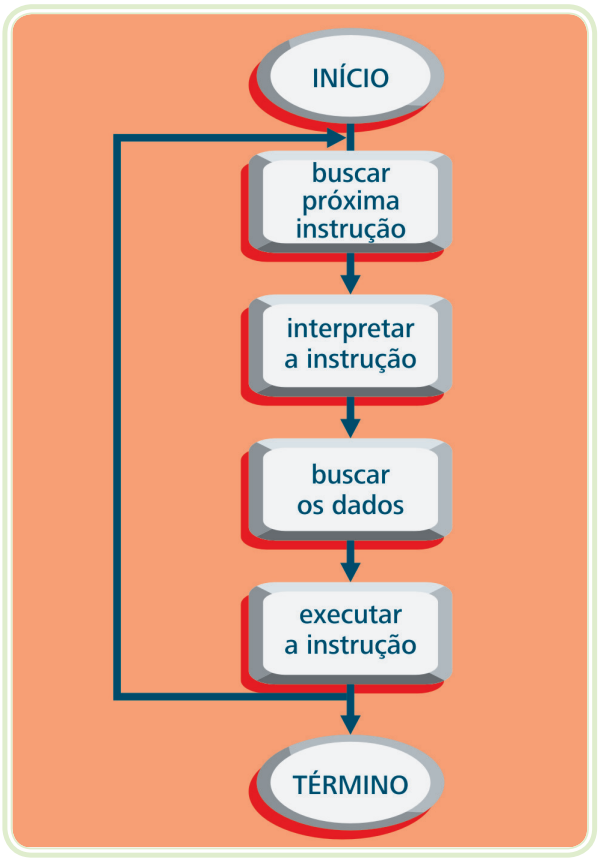


Figura 2.8: Ciclo de instrução
Fonte: CTISM

2.6 Estratégias de implementação de processadores

As principais estratégias de implementação de processadores que a indústria de semicondutores adota são o RISC e o CISC.

O CISC (*Complex Instruction Set Computer*) é a arquitetura utilizada nos processadores da família *Pentium* e dos *Mac* de primeira geração, por exemplo. Essa arquitetura caracteriza-se por possuir um conjunto de instruções maior e mais complexo e apresentar um ciclo de processamento mais lento.

O RISC (*Reduced Instruction Set Computer*), por sua vez, é utilizado nos processadores *Power* PC e *Sparc*, e nos microcontroladores PIC de 8 *bits*. Essa arquitetura consiste em um conjunto de instruções menor e mais simples e possui um ciclo de processamento mais rápido.

Lembramos que em nosso estudo adotaremos o termo **instrução**, para as instruções de máquina ou em linguagem *Assembly*, e **comando**, para linguagens de alto nível. Portanto, o projeto de um processador poderia ser resumido em:

a) Definir o conjunto de instruções (todas as instruções possíveis que o processador poderá executar), ou seja:

- Definir o formato e o tamanho das instruções;
- Definir as operações elementares.

b) Projetar os componentes do processador (UAL, UC, registradores, barramentos). Duas estratégias são possíveis na construção do decodificador de instruções da UC:

- *Wired Logic* (as instruções são todas implementadas em circuito);
- Microcódigo (apenas um grupo básico de instruções são implementadas em circuito; as demais são "montadas" através de microprogramas que usam as instruções básicas, valendo-se, para isso, até mesmo de algoritmos de lógica difusa (*Fuzzy Logic*).

Resumo

Nessa aula, estudamos o funcionamento básico da arquitetura de Von Neumann, bem como relacionamos as diferentes partes constitutivas de um computador que obedece ao modelo de Von Neumann. Já conhecemos o *hardware* básico dos computadores e como os sinais elétricos trafegam através dos diferentes componentes físicos dos computadores. Estamos aptos a avançar em nossos estudos, passando ao estudo dos *softwares* que movem estas poderosas máquinas.

Atividades de aprendizagem

1. Descreva as principais características da arquitetura de Von Neumann.
2. Faça um comparativo entre os principais tipos de memória quanto à hierarquia.

3. Com relação à leitura e à escrita, descreva os diversos tipos de memória.

4. Pesquise, na internet, vantagens e desvantagens dentre as principais estratégias de implementação de processadores.

Aula 3 – Tradução de programas

"A música não está no piano."

Alan Curtis Kay

Objetivos

Entender por que programas em linguagem de alto nível devem ser traduzidos para linguagem de máquina.

Compreender como se dá a interpretação e compilação de programas.

Conhecer emuladores e máquinas virtuais e compreender sua importância.

3.1 Programa em linguagem de máquina

Para executar uma tarefa qualquer, um computador precisa receber instruções exatas sobre o que fazer. Uma sequência adequada de instruções de computador, para a realização de uma determinada tarefa, constitui-se de um PROGRAMA de computador. Uma linguagem de programação é um conjunto de ferramentas, regras de sintaxe e símbolos ou códigos que nos permitem escrever programas de computador, destinados a instruir o computador para a realização de suas tarefas.

A primeira e mais primitiva linguagem de computador é a própria linguagem de máquina, aquela que o computador entende diretamente e pode ser diretamente executada pelos circuitos do processador (pelo *hardware*).

No início da era da computação, os programas eram escritos em linguagem de máquina, isto é, as instruções eram escritas diretamente na linguagem do computador (formada apenas com 1's e 0's). Um programa em linguagem de máquina é uma longa série de 0's e 1's, ordenados de forma que alguns representam códigos de instruções e outros representam os dados que serão processados (ou indicam onde esses dados estão armazenados). Em um programa escrito em linguagem de máquina, cada instrução escrita pelo

programador será individualmente executada, isto é, a cada instrução do programa corresponderá uma ação do computador. A relação é, portanto, 1 para 1 – uma instrução do programa corresponde a uma operação do computador.

Um programa em linguagem de máquina pode ser apresentado em binário (0's e 1's) ou em hexadecimal (usando de 0 a F, ou seja, transformando cada 4 *bits* em um dígito hexadecimal). A apresentação em hexadecimal torna mais enxuta a representação (mas não mais simples), contudo serve somente para visualização, pois um programa somente pode ser submetido ao computador em binário.

Imagine, então, um programa extenso, escrito usando apenas 1's e 0's; imagine que, para cada diferente marca ou modelo de computador, as regras para entender esses códigos serão totalmente diferentes; e finalmente imagine que você teria que escrever uma a uma as instruções e os dados adequadamente codificados e ordenados, perfurar todo o programa em cartões e submeter toda a massa de cartões ao computador, para finalmente receber, algumas horas depois, o seu programa de volta com uma mensagem de erro, do tipo "erro no cartão X"...e mais nada! Um programa escrito nessa linguagem era difícil de ser escrito sem que se cometessem muitos erros, e esse processo era longo, demorado, difícil, entediante e, principalmente, caro.

Um programa em linguagem de máquina era também extremamente difícil de ser entendido por outros programadores que futuramente viessem a trabalhar na sua manutenção. Essa complexidade levou à necessidade de desenvolver técnicas e ferramentas para tornar a escrita e a manutenção de programas mais fáceis, mais rápida e, principalmente, mais barata.

Cada família de computadores possui sua própria linguagem de máquina. Um programa em linguagem de máquina é dependente do computador, ou seja, tendo sido escrito para um determinado computador, somente poderá ser executado em computadores da mesma família, que lhe sejam 100% compatíveis.

3.2 Linguagens de montagem

A primeira tentativa bem sucedida para resolver o problema acima descrito foi a criação de uma linguagem em que os códigos numéricos foram substi-

tuídos por mnemônicos (palavras ou símbolos como, por exemplo, *LOAD* = carregar, e *ADD* = somar, que se aproximam de palavras comuns da língua inglesa). As localizações dos dados foram substituídas por referências simbólicas. Foram também definidas regras de sintaxe de fácil memorização, de forma a tornar a escrita de programas e sua posterior manutenção uma técnica de complexidade relativamente menor.

Essa linguagem simbólica recebeu o nome de *Assembly Language* (Linguagem de Montagem). Assim, o programador não mais precisava decorar os códigos numéricos que representavam as diferentes instruções e os endereços reais de armazenamento. Bastava decorar mnemônicos para as instruções e definir nomes para as referências dos endereços (por exemplo, NOME para o local onde seriam armazenados os nomes e SALARIO para o local onde seriam armazenados os salários, etc.), o que, sem dúvida, facilita o trabalho.

É importante lembrar que um computador é sempre monoglota, isto é, ele entende única e exclusivamente a sua própria linguagem de máquina. Portanto, para escrever um programa em outra linguagem, que seja entendido e processado no computador, é preciso ter outro programa que leia o que este outro escreveu, em linguagem alternativa e o traduza para uma nativa do computador, isto é, a linguagem de máquina entendida pelo computador.

O processo de tradução da linguagem de montagem para a linguagem de máquina (chamado de montagem) é realizado por um programa chamado *Assembler* (Montador). O programa *Assembler* lê cada instrução escrita em linguagem *Assembly* e a converte em uma instrução equivalente à linguagem de máquina; e também converte cada uma das referências simbólicas de memória em endereços reais (resolve as referências de memória).

A criação de programas montadores facilitou muito o trabalho dos programadores. Outra vantagem menos óbvia foi possibilitar o desenvolvimento de programas de crítica de sintaxe (os *debuggers*), facilitando o processo de depuração de erros de programação.

3.3 Linguagens de programação

Apesar da existência de programas Montadores, o processo continuava lento e complexo, exigindo do programador uma grande compreensão do processo e um profundo conhecimento da máquina que ele estava programando. Um programa de computador ainda era difícil de ser escrito, caro e dependen-

te do computador para o qual foi escrito, já que um programa escrito em linguagem de máquina para um determinado computador só poderia ser processado em computadores 100% compatíveis com ele.

Esses problemas levaram a uma busca por linguagens que fossem mais simples de programar e entender, mais rápidas e eficientes; levaram a programas mais enxutos, com menos instruções, menos dependentes do computador alvo, mas que processassem com boa eficiência, não acarretando processamento lento no computador.

Foram desenvolvidas diversas linguagens de programação, buscando afastar-se do modelo centrado no computador. Além disso, essas linguagens foram estruturadas buscando refletir melhor os processos humanos de solução de problemas. Essas linguagens orientadas para problemas são também chamadas linguagens de alto nível, por serem afastadas do nível de máquina.

Algumas das primeiras linguagens de alto nível desenvolvidas e amplamente difundidas foram a FORTRAN (1957), usada basicamente para manipulação de fórmulas; ALGOL (1958), utilizada na manipulação de algoritmos; e COBOL (1959), utilizada para processamento comercial.

Nas décadas de 60 e 70 (podemos citar Pascal), a primeira linguagem foi estruturada de alto nível; BASIC, linguagem criada para facilitar a programação por não profissionais; e ADA, linguagem para processamento em tempo real, criada sob encomenda do DoD (*Department of Defense*) norte-americano.

Na década de 1980, surgiu o C e depois o C++ (com suporte a objetos), que estão entre as linguagens mais utilizadas hoje.

Cada nova linguagem criada visa a atingir níveis de abstração mais altos, isto é, afasta cada vez mais o programador do nível de máquina. Se, por um lado, essas novas linguagens facilitam muito o trabalho dos programadores, reduzindo sua necessidade de conhecer o *hardware* da máquina, por outro lado, elas cobram um alto preço em termos de desempenho, isto é, são cada vez mais lentas, ao consumir cada vez mais ciclos de máquina e espaço em memória. Esse aumento de exigência da capacidade de processamento dos computadores é compensado pelo aumento acelerado do poder de processamento dos novos *chips* e pelos avanços na arquitetura dos computadores.

Tal como na linguagem humana, as linguagens de computadores proliferam e sempre há problemas que persistem, continuando a busca por uma linguagem ideal – a solução "definitiva". A linguagem *Java* é a mais importante tendência atual e o mais recente avanço na busca pela linguagem universal. A linguagem *Java* pretende elevar a abstração ainda mais um nível e propõe-se a ser independente da máquina na qual será executada. Na realidade, quando não está sendo processado em um processador *Java* nativo, o código *Java* é interpretado por um emulador, que é uma camada de *software* chamada máquina virtual Java (*JVM – Java Virtual Machine*).

3.4 Tradução

Sabemos que um programa escrito por um programador (chamado código-fonte) em uma linguagem de alto nível é um conjunto de instruções que é claro para programadores, mas não para computadores. Ou seja, os computadores entendem única e exclusivamente suas linguagens nativas, as linguagens de máquina. Programas em linguagem de alto nível, a exemplo dos programas escritos em linguagem de montagem, também precisam ser traduzidos para a linguagem de máquina para serem submetidos ao computador e processados.

O processo de tradução do programa escrito em uma linguagem simbólica pelo programador – código-fonte (*source code*), para a linguagem de máquina do computador – código-objeto (*object code*), é chamado compilação e é realizado por um programa chamado compilador (*compiler*).

3.5 Montagem

Citamos anteriormente uma forma de tradução rápida e simples: a executada pelo programa montador. O processo de montagem traduz um programa escrito em linguagem *Assembly* em um programa equivalente em linguagem de máquina, possível de ser executado pelo computador.

A seguir, na Figura 3.1, é apresentado o fluxo que representa o processo de montagem.

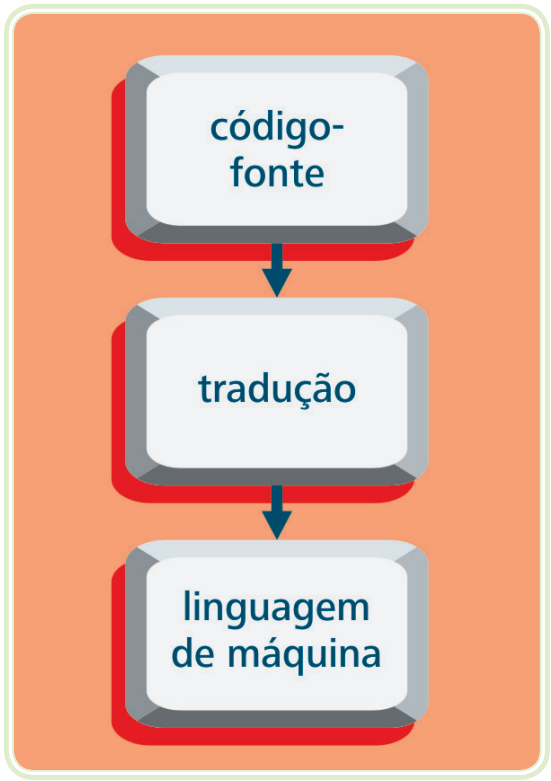


Figura 3.1: Processo de montagem
Fonte: CTISM

No processo de montagem, o código-fonte (programa em linguagem simbólica escrito pelo programador) é examinado, instrução por instrução, e é feita a tradução, gerando o código que será executado (código-objeto). Os passos executados pelo programa montador são:

a) Verificar a correção do código de instrução (se o mnemônico corresponde a uma instrução válida para o computador, se os campos definidios na estrutura da linguagem e a sintaxe estão corretos) e substituir os mnemônicos pelos códigos numéricos binários equivalentes. Qualquer erro no código acarreta a interrupção do processo e a emissão de mensagem de erro;

b) Resolver as referências de memória: os nomes simbólicos adotados pelo programador são convertidos para endereços reais de memória (valores numéricos binários de endereços);

c) Reservar espaço em memória para o armazenamento das instruções e dos dados;

d) Converter valores de constantes em binário.

3.6 Compilação

Compilação é o processo de tradução de um programa escrito em linguagem de alto nível para código em linguagem de máquina. Esse processo é análogo ao da montagem (verificação/análise do código-fonte, resolução das referências de memória, reserva de espaço em memória e conversão para código de máquina binário). O que diferencia a compilação do processo de montagem é sua maior complexidade. No processo de montagem, há uma relação de 1:1, ou seja, cada instrução do código-fonte resulta em uma instrução de máquina, enquanto que na compilação a relação é múltipla, cada instrução do código-fonte pode gerar várias instruções de máquina. Observe a Figura 3.2 a seguir.

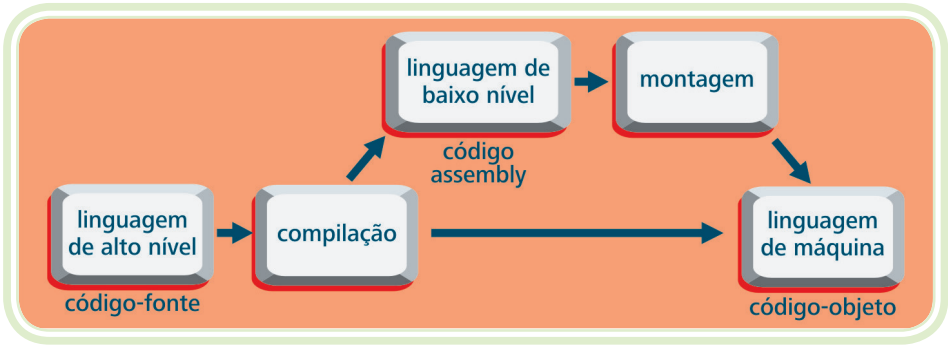


Figura 3.2: Processo de tradução
Fonte: CTISM

Durante a compilação, o código-fonte é analisado (análise léxica, sintática e semântica). É gerado um código intermediário e são construídas tabelas de símbolos; alocam-se as áreas de memória para variáveis, atribuem-se os registradores a serem utilizados, e é, finalmente, gerado o código-objeto em linguagem binária de máquina. Em alguns compiladores, é gerado um código intermediário em *Assembly* (que pode ser visualizado pelo programador) e que, em seguida, passa pelo montador para gerar finalmente o código-objeto em linguagem de máquina.

O código-objeto pode ser absoluto (os endereços constantes são endereços reais de memória), ou relocável (os endereços são relativos, tendo como referência o início do programa; já os endereços reais de memória são definidos apenas em tempo de execução).

3.7 Bibliotecas

O desenvolvimento de um programa certamente utilizará diversas operações que são comuns a muitos programas, como, por exemplo, a execução de uma instrução de entrada e saída, a classificação dos dados de um arquivo ou o cálculo de funções matemáticas. Uma linguagem de alto nível geralmente incorpora diversas rotinas prontas, que fazem parte da linguagem e compõem bibliotecas (*librarys*) de funções pré-programadas que poderão ser utilizadas pelo programador, economizando tempo, aumentando a eficiência e evitando erros. Dessa forma, um programa em alto nível possivelmente conterá diversas chamadas de biblioteca (*library calls*). Essas funções não devem ser confundidas com as instruções da linguagem. Na realidade, são pequenos programas externos que são denominados, através de instruções especiais, de chamada de biblioteca. Para serem executadas, essas rotinas precisam ser incorporadas ao código do programador, isto é, a chamada de biblioteca precisa ser substituída pelo código propriamente dito, incluindo os parâmetros necessários.

3.8 Ligação

O código-objeto preparado pelo compilador, em geral, não é imediatamente executável, pois ainda existe código (as rotinas de biblioteca) a ser incorporado ao programa. A cada chamada de biblioteca encontrada no código-fonte, o compilador precisará incluir uma chamada para a rotina e o endereço dos dados que devem ser passados para a rotina.

A tarefa de examinar o código-objeto, procurar as referências a rotinas de biblioteca (que constituem referências externas não resolvidas), buscar a rotina da biblioteca, substituir a chamada pelo código ("resolver as referências externas") e obter os parâmetros para incluí-los no código-objeto é executada por um programa chamado ligador (*LinkEditor*). O resultado da execução do ligador é o código final, pronto, para ser executado pelo computador, chamado módulo de carga ou código executável (Figura 3.3).

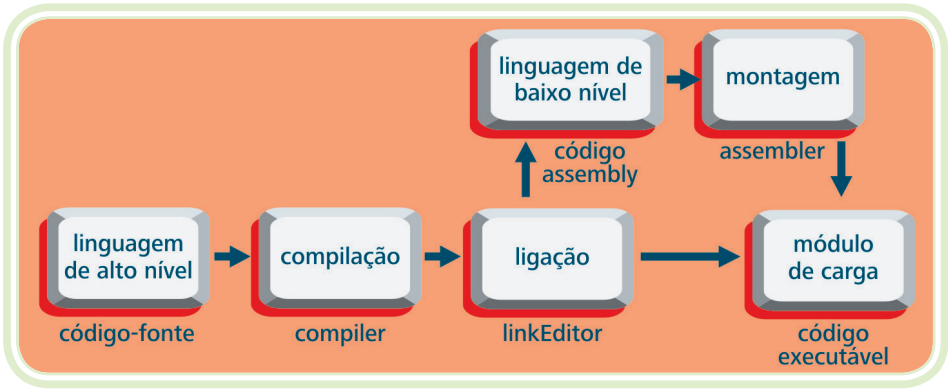


Figura 3.3: Processo de compilação e ligação
Fonte: CTISM

O módulo de carga, após testado e depurado, isto é, depois de resolvidos todos os erros, também chamados *bugs*, é armazenado em memória de massa para ser executado quando necessário.

Para saber o significado de *BUG*, e qual a sua origem, consulte o *link*: http://pt.wikipedia.org/wiki/Bug

O processo de compilação e ligação é executado apenas pelo programador na fase de desenvolvimento e não mais precisará ser executado pelo usuário quando da execução do programa.

3.9 Interpretação

Com a apresentação do processo de execução de um programa em fases distintas (compilação/ligação/execução), um programa, para ser executado, precisa, ter sido convertido para código-objeto pelo compilador e ter passado pelo ligador (ver Figura 3.4 a seguir). Esse processo é o mais utilizado, porém não é o único.

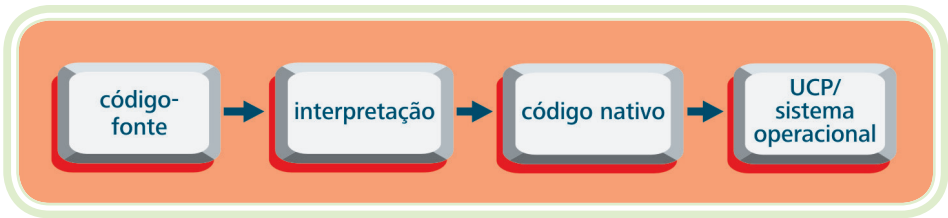


Figura 3.4: Processo de interpretação
Fonte: CTISM

O método alternativo chama-se interpretação e, a partir do programa-fonte, realiza as três fases: compilação, ligação e execução, comando por comando, em tempo de execução. Não existem fases distintas nem se produzem códigos intermediários. Todo o processo de conversão é efetuado em tempo de execução e imediatamente executado. Ou seja, cada comando é lido, verificado,

convertido em código executável e imediatamente executado, antes que o comando seguinte seja lido.

Linguagens como C e Pascal são linguagens tipicamente compiladas, enquanto que a BASIC foi desenvolvida como linguagem interpretada (hoje também existem linguagens BASIC compiladas). As linguagens de programação tipicamente de usuário, tais como das planilhas *Excel*, o *Word Basic* (linguagem de construção de Macros do *Word*) e o *Access*, são exemplos de linguagens interpretadas.

3.10 Compilação e interpretação: comparação

a) Tempo de execução

No método de interpretação, cada vez que o programa for executado, haverá compilação, ligação e execução de cada um dos comandos. No método de compilação, o tempo de execução do programa é reduzido porque todos os passos preliminares (compilação e ligação) foram previamente cumpridos.

b) Consumo de memória

No método de interpretação, o interpretador é um programa geralmente grande, que precisa permanecer na memória durante todo o tempo que durar a execução do programa. Um programa necessita do interpretador para ter traduzidos cada um dos seus comandos, um a um, até o término de sua execução (o interpretador somente é descarregado depois do término da execução do programa).

No método de compilação, o compilador é carregado e fica na memória apenas durante o tempo de compilação, depois é descarregado; o ligador é carregado e fica na memória apenas durante o tempo de ligação, depois é descarregado. Essas são funções realizadas pelo programador e executadas apenas durante o desenvolvimento do programa. Quando o usuário for executar o programa, apenas o módulo de carga (código executável) é carregado e fica na memória durante a execução. Dessa forma, vemos que o método de interpretação acarreta um consumo de memória muito mais elevado durante a execução do programa.

c) Repetição de interpretação

No método de compilação, um programa é compilado e ligado apenas uma vez, na hora da execução; é carregado apenas o módulo de carga, que é diretamente executável. No método de interpretação, cada programa terá que ser interpretado toda vez que for executado.

Outro aspecto a ser destacado é que em programas contendo *loops*, no método de interpretação as partes de código pertencentes ao *loop* serão várias vezes repetidas e terão que ser interpretadas tantas vezes quantas o *loop* tiver que ser percorrido. No método de compilação, a tradução do código do *loop* faz-se uma única vez, em tempo de compilação e ligação. Essas características levam a um maior consumo de tempo no método de interpretação, que é, portanto, mais lento.

d) Desenvolvimento de programas e depuração de erros

No método de compilação, a identificação de erros durante a fase de execução fica sempre difícil, pois não há mais relação entre comandos do código-fonte e instruções do executável.

No método de interpretação, cada comando é interpretado e executado individualmente, a relação entre código fonte e executável é mais direta e o efeito da execução (certa ou errada) é direta e imediatamente sentida. Quando a execução de um comando acarreta erro, quase sempre o erro pode ser encontrado no comando que acabou de ser executado. Assim, o interpretador pode informar o erro, indicando o comando ou a variável causadora do problema. Essa característica faz com que esse método seja escolhido sempre que se pretende adotar uma linguagem mais fácil, a não profissional, ou para a programação ser mais fácil e rápida. Por exemplo, o método de interpretação é usado pela maioria dos *Basic* (uma linguagem projetada para ser usada por iniciantes) e por todas as linguagens típicas de usuário como *Access, Excel,* etc.

3.11 Emuladores e máquinas virtuais

Na informática, um emulador é um *software* que reproduz as funções de um determinado ambiente a fim de permitir a execução de outros *softwares* sobre ele. Pode ser pela transcrição de instruções de um processador alvo para o processador no qual ele está rodando, ou pela interpretação de chamadas para simular o comportamento de um *hardware* específico. O emulador também é responsável pela simulação dos circuitos integrados ou *chips* do sistema de *hardware* em um *software*. Basicamente, um emulador expõe as funções de um sistema para reproduzir seu comportamento, permitindo que um *software* criado para uma plataforma funcione em outra.

Uma aplicação interessante dos interpretadores é a geração de código universal e de máquinas virtuais.

Sabemos que um computador somente é capaz de executar programas que tenham sido desenvolvidos para ele. Assim, um programa desenvolvido para rodar em PC's, rodando *Windows*, não funciona em PC's com ***Linux*** ou em *Macintosh*. Se você concorda com essa afirmação, imagine então uma página na internet, com textos, imagens e programas que podem ser visualizados e processados por quase todos os computadores. Como isso pode ser feito? Como computadores de marcas e modelos diferentes estarão lendo, interpretando e executando corretamente comandos que podem ter sido desenvolvidos usando outro computador?

A-Z

Linux
É um sistema operacional - programa responsável pelo funcionamento do computador - que faz a comunicação entre *hardware* (impressora, monitor, *mouse*, teclado) e *software* (aplicativos em geral). A diferença mais marcante entre *Linux* e *Windows* é o fato de o *Linux* ser um sistema de código aberto, desenvolvido por programadores voluntários espalhados por toda internet e distribuído sob licença pública. Já o *Windows* é *software* proprietário, não possui código-fonte disponível e você ainda precisa comprar uma licença para ter o direito de usá-lo.

O segredo é a utilização de linguagens padronizadas, tais como: HTML, para a escrita das páginas; CGI, *Java, Java Script,* etc., para programas, que são suportadas por diversas plataformas. Assim, cada uma das plataformas (através dos programas visualizadores de páginas da internet, conhecidos como *browsers* ou mesmo através de seus respectivos sistemas operacionais) pode interpretar corretamente qualquer página feita e hospedada em qualquer computador.

Uma situação semelhante ocorre quando alguém desenvolve um programa que "interpreta" o código executável produzido para um determinado computador e o converte para ser executado em outro computador incompatível com o primeiro. Imagine pegar um jogo desenvolvido para os consoles originais Atari (por exemplo, o *PacMan*) e poder executá-lo num moderno PC. Ou pegar um programa que você tenha escrito (por exemplo, uma planilha) para um *Apple Macintosh* e rodá-lo num PC. Esse processo, em que um computador opera como se fosse o outro, é conhecido como emulação.

Levando esse conceito um pouco mais adiante, imagine desenvolver um programa conversor que selecionasse qualquer programa escrito para uma determinada máquina e interpretasse seu código executável, traduzindo-o em tempo de execução para instruções de outro computador. Esse programa criaria uma camada de emulação em que uma máquina se comportaria como outra. Poderíamos ter um PC "virtual" emulado em um *Macintosh*, que estaria, assim, apto a rodar qualquer programa escrito para PC. Esse programa emulador criaria um ambiente a que chamamos de máquina virtual, isto é, uma máquina que se comporta como outra, diferente, não compatível.

Algo parecido foi desenvolvido pela *Sun Microsystems* na linguagem *Java*. *Java* é uma linguagem que, em princípio, permite que programas escritos nela rodem em qualquer máquina. Na realidade, a *Sun* desenvolveu uma pla-

taforma *Java* (*JVM - Java Virtual Machine*) com características de ambiente necessárias para que os programas escritos em *Java* rodem adequadamente. A *JVM* suporta uma representação em *software* de uma UCP completa, com sua arquitetura perfeitamente definida, incluindo seu próprio conjunto de instruções. Os programadores *Java* escrevem o código usando o conjunto de instruções definido pela linguagem *Java*. Essa fonte será, então, compilada gerando o código de máquina virtual *Java*. Como o código *Java* é universal, os códigos-fonte *Java* e os códigos-objeto gerados são independentes da máquina em que o *software* será depois processado. Assim, os programadores *Java* não precisam preocupar-se em qual computador ou sistema operacional o programa vai ser executado: desenvolver para *Java* é independente da máquina.

Mas como é possível que isso funcione, sem desmentir tudo o que antes dissemos sobre o código de máquina ser dependente do ambiente? Tudo bem que o código *Java* rode bem em um processador *Java*, mas, e em outras plataformas? Bom, em outras plataformas, programas escritos em *Java* rodariam em máquinas virtuais que emulariam o ambiente *Java*. Isso significa que as empresas que desenvolvem *software* construíram uma camada de *software* que, em tempo de execução, lê o código *Java* e o interpreta, traduzindo-o para o código nativo daquele ambiente. Esses programas **interpretadores** permitem que um programa *Java* seja traduzido, em tempo de execução, para o código nativo (ver Figura 3.5) e executado no computador do usuário, com maior ou menor eficiência, porém sempre mais lentamente que na máquina *Java* ideal.

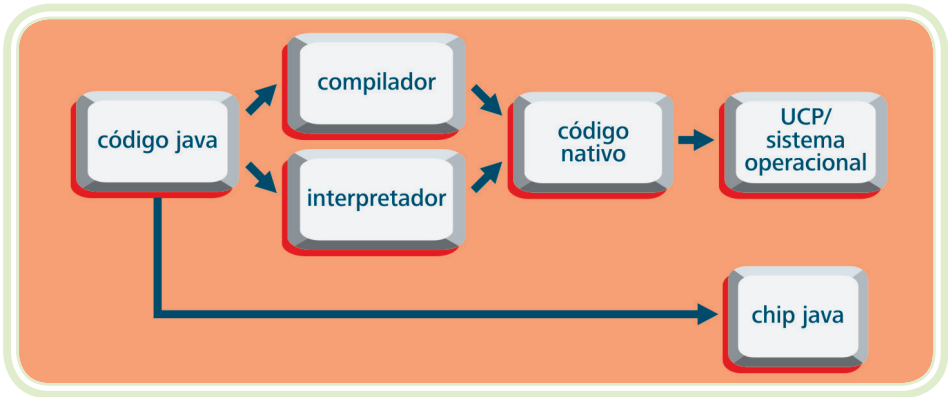


Figura 3.5: Processo código *Java*
Fonte: CTISM

Resumo

Vimos, nessa aula, que os programas de computadores feitos em algum tipo de linguagem de programação – Pascal, C, C++, Basic – devem ser traduzidos para linguagem de máquina.

Aprendemos que os emuladores permitem que um programa feito para rodar em um determinado computador possa ser rodado em outro sem perder suas funcionalidades. Portanto, após estudar *hardware* e *software*, resta-nos conhecer como interagir com o computador, ou seja, como os nossos comandos entram no computador e como podemos ter acesso aos resultados do processamento das informações pelo computador.

Atividades de aprendizagem

1. Pesquise na internet sobre a Lei de Moore e suas implicações no desenvolvimento dos computadores.

2. Por quais motivos desenvolveram-se os emuladores e as máquinas virtuais?

3. Quais as razões que obrigam os programas escritos em linguagem de alto nível, a serem traduzidos para linguagem de máquina?

4. Faça uma tabela comparativa entre compilação e interpretação.

Aula 4 – Entradas e saídas

"O bom andarilho não deixa rastros."

Lao Tse, Tao-Te King

Objetivos

Entender de que forma nossos comandos ingressam no computador.

Compreender como podemos ter acesso aos resultados do processamento das informações pelo computador.

Diferenciar as diversas formas de comunicação de e para o computador, bem como a maneira como se dá o armazenamento de informações em um computador.

4.1 Considerações iniciais

Os dispositivos de entrada e saída têm como funções básicas:

a) A comunicação do usuário com o computador;

b) A comunicação do computador com o meio ambiente (dispositivos externos a serem monitorados ou controlados);

c) Armazenamento (gravação) de dados.

As características que regem a comunicação de cada um dos dispositivos de E/S (entrada e saída) com o núcleo do computador (composto de UCP e memória principal) são muito diferentes entre si. Cada dispositivo de E/S comunica-se com o núcleo de forma diversa do outro. Entre outras diferenças, os dispositivos de entrada e saída são mais lentos que o computador, característica essa que impõe restrições à comunicação, uma vez que o computador precisaria esperar muito tempo pela resposta do dispositivo. Outra diferença fundamental diz respeito às características das ligações dos sinais dos dispositivos.

Os primeiros computadores, especialmente os de pequeno porte, eram muito lentos, e os problemas de diferença de velocidade eram resolvidos sem dificuldade e não representavam um problema importante. Dessa forma, a ligação dos dispositivos de E/S era feita através de circuitos simples (as interfaces) que apenas resolviam os aspectos de compatibilização de sinais elétricos entre os dispositivos de E/S e a UCP. Os aspectos relativos a diferenças de velocidade (especialmente tempo de acesso e taxa de transferência de dados) eram resolvidos por programa, isto é, por *software*. Entre esses componentes, trafegam informações relativas a dados, endereços e controle.

4.2 Tipos de dispositivos

As funções dos dispositivos de entrada são: coletar informações e introduzir as informações na máquina; converter informações do homem para a máquina e vice-versa; e recuperar informações dos dispositivos de armazenamento.

As funções dos dispositivos de saída são: exibir ou imprimir os resultados do processamento; controlar dispositivos externos.

A UCP não se comunica diretamente com cada dispositivo de E/S, mas com "interfaces", de forma a compatibilizar as diferentes características. O processo de comunicação (**protocolo**) é feito através de transferência de informações de controle, endereços e dados propriamente ditos. Inicialmente, a UCP interroga o dispositivo, enviando o endereço deste e um sinal que indica se quer mandar ou receber dados através da interface. O periférico, reconhecendo seu endereço, responde quando está pronto para receber (ou enviar) os dados. A UCP, então, transfere (ou recebe) os dados através da interface, e o dispositivo responde, confirmando o recebimento (ou a transferência) dos dados (*acknowledge* ou *ACK*) ou o não recebimento e, nesse caso, solicitando retransmissão (*not-acknowledge* ou *NACK*).

A-Z

protocolo
Um conjunto de informações ou dados devem ser preparados de maneira a obedecerem a determinadas regras-padrão que os computadores entendem para poderem ser transmitidos entre um computador e outro. Estas regras-padrão são chamadas de protocolo.

A compatibilização de velocidades é feita geralmente por programa, usando memórias temporárias na interface chamadas *"buffers"* - que armazenam as informações conforme chegam da UCP, e as liberam para o dispositivo à medida que este as pode receber.

As interfaces de entrada e saída são conhecidas por diversos nomes, dependendo do fabricante: Interface de E/S = Adaptador de Periférico, Controladora de E/S, Processador de Periférico, Canal de E/S.

4.3 Formas de comunicação

De uma forma geral, a comunicação entre o núcleo do computador e os dispositivos de E/S poderia ser classificada em dois grupos: comunicação paralela ou serial. Vamos, a seguir, analisar as características desses grupos.

4.3.1 Comunicação em paralelo

Na comunicação em paralelo, grupos de *bits* são transferidos simultaneamente (em geral, *byte* a *byte*), através de diversas linhas condutoras dos sinais. Dessa forma, como vários *bits* são transmitidos simultaneamente a cada ciclo, a taxa de transferência de dados (*throughput*) é alta. Veja a Figura 4.1 a seguir.

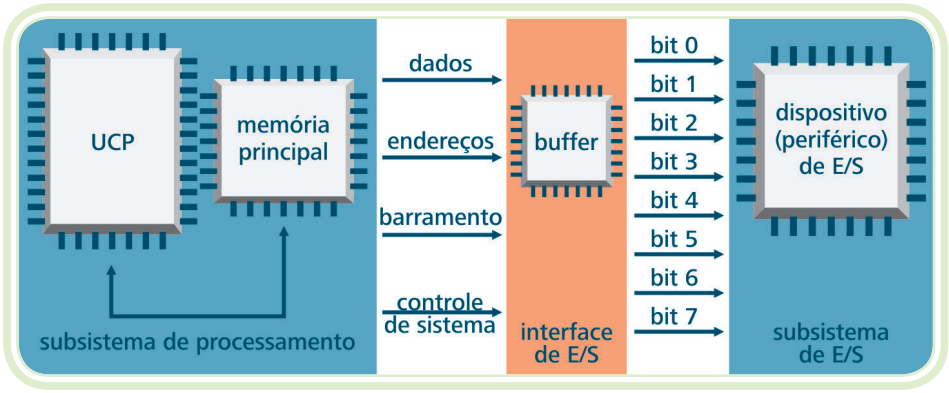


Figura 4.1: Tráfego de dados em uma transmissão paralela
Fonte: CTISM

No entanto, o processo de transferência em paralelo envolve um controle sofisticado e é razoavelmente complexo, o que o torna mais caro. Um importante problema diz respeito à propagação dos sinais no meio físico, isto é, no cabo de conexão entre o dispositivo e a interface. Essa propagação deve ser feita de modo que os sinais (os *bits*) correspondentes a cada *byte* cheguem simultaneamente à extremidade oposta do cabo, na qual serão reagrupados em *bytes*. Como os condutores que compõem o cabo usualmente terão pequenas diferenças físicas, a velocidade de propagação dos sinais digitais nos condutores poderá ser ligeiramente diferente nos diversos fios.

Dependendo do comprimento do cabo, pode acontecer que um determinado fio conduza sinais de forma mais rápida (ou mais lenta) que os demais e, dessa forma, um determinado *bit* x em cada *byte* irá propagar-se mais rápido e chegar à extremidade do cabo antes que os outros n-1 *bits* do *byte*. Esse fenômeno é chamado *skew*, e as consequências são catastróficas: os *bits* n chegariam fora de ordem (os *bytes* chegariam embaralhados) e a informação ficaria irrecuperável. Em decorrência desse problema, há limites para o

comprimento do cabo que interliga um dispositivo ao computador quando o modo paralelo é usado.

As restrições citadas contribuem para que a utilização da comunicação em paralelo se limite a aplicações que demandem altas taxas de transferência, normalmente associadas a dispositivos mais velozes tais como unidades de disco, ou demandem altas taxas de transferência, como CD-ROM, DVD, ou mesmo impressoras, e que se situem muito próximo do núcleo do computador. Em geral, o comprimento dos cabos paralelos é limitado até um máximo de aproximadamente 1,8 metros.

4.3.2 Comunicação serial

Na comunicação serial, os *bits* são transferidos um a um, através de um único par condutor. Os *bytes* a serem transmitidos são serializados, isto é, são "desmontados" *bit* a *bit* e são individualmente transmitidos, um a um. Na outra extremidade do condutor, os *bits* são contados e, quando formam 8 *bits*, são remontados, reconstituindo os *bytes* originais. Nesse modo, o controle é comparativamente muito mais simples que no modo paralelo e é de implementação mais barata. Como todos os *bits* são transferidos pelo mesmo meio físico (mesmo par de fios), as eventuais irregularidades afetam todos os *bits* igualmente. Portanto, a transmissão serial não é afetada por irregularidades do meio de transmissão e não há *skew*. Em princípio, a transmissão serial é intrinsecamente mais lenta que a paralela (pois apenas um *bit* é transmitido de cada vez). Analise a Figura 4.2.

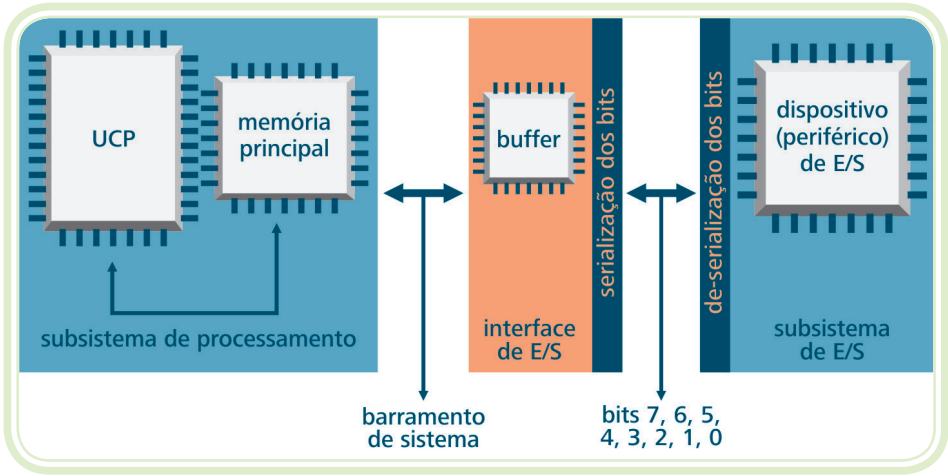


Figura 4.2: Tráfego de dados em uma transmissão serial
Fonte: CTISM

Como os *bits* são transmitidos sequencialmente um a um, sua utilização é normalmente indicada apenas para periféricos mais lentos como, por exemplo, teclado, *mouse,* etc., ou quando o problema da distância for mandatório, como nas comunicações a distâncias médias (tal como em redes locais) ou longas (comunicações via linha telefônica usando *modems*).

No entanto, ultimamente a transmissão serial tem recebido aperfeiçoamentos importantes (seja de protocolo, de interface, seja de meio de transmissão) que permitem o aumento da velocidade de transmissão por um único par de fios, cabo coaxial ou de fibra ótica. Como o aumento da velocidade de transmissão em interfaces paralelas ocasiona mais *skew*, a tendência tem sido no sentido do aperfeiçoamento das interfaces seriais que, hoje, permitem taxas de transferência muito altas com, relativamente, poucas restrições de distância. Em microcomputadores, a interface USB –- *Universal Serial Bus*– permite, atualmente, ligar até 128 dispositivos a taxas muito altas (centenas de kbps). Veja a Quadro 4.1.

Quadro 4.1: Comparativo de comunicação

Característica	Paralelo	Serial
custo	maior	menor
distância	curta	sem limite
throughput	alto	baixo

Saiba mais sobre interfaces modernas, como a interface Serial ATA: http://www.infowester.com/serialata.php

4.4 Formas de transmissão

A transmissão de caracteres através de uma linha de comunicação pode ser feita por dois diferentes métodos: transmissão síncrona e assíncrona.

4.4.1 Transmissão síncrona

Na transmissão síncrona, o intervalo de tempo entre dois caracteres subsequentes é fixo. Nesse método, os dois dispositivos – transmissor e receptor – são sincronizados, pois existe uma relação direta entre o tempo e os caracteres transferidos. Quando não há caracteres a serem transferidos, o transmissor continua enviando caracteres especiais de forma que o intervalo de tempo entre caracteres mantém-se constante e o receptor sincronizado. No início de uma transmissão síncrona, o relógio dos dispositivos transmissor e receptor são sincronizados através de um *string* (cadeia de caracteres) de sincronização e são mantidos sincronizados por longos períodos de tempo (dependendo da estabilidade dos relógios), podendo transmitir dezenas de milhares de *bits* antes de terem necessidade de ressincronizar (Figura 4.3).

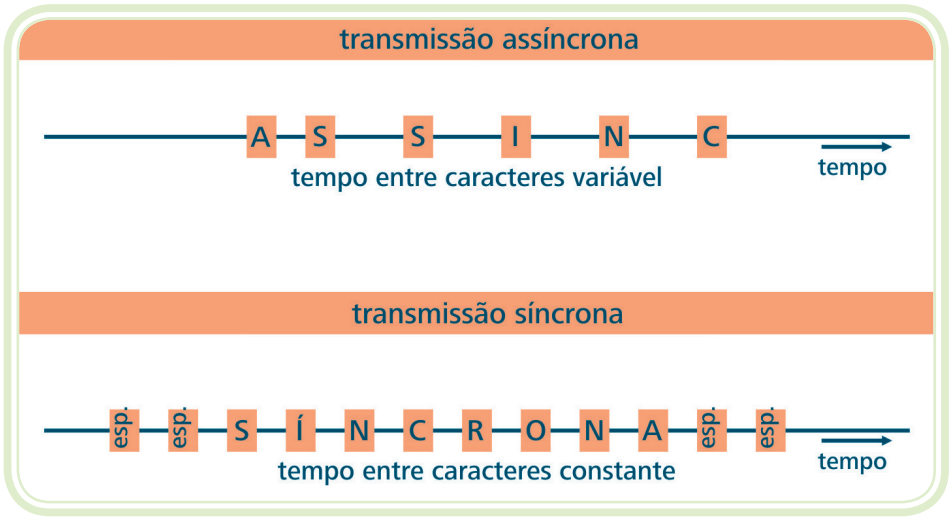


Figura 4.3: Representação temporal dos métodos de transmissão
Fonte: CTISM

4.4.2 Transmissão assíncrona

Na transmissão assíncrona, o intervalo de tempo entre os caracteres não é fixo. Podemos exemplificar com um digitador operando um terminal, no qual podemos verificar que não há um fluxo homogêneo de caracteres a serem transmitidos. Como esse fluxo não é homogêneo, não há como distinguir a ausência de *bits* sendo transmitidos de um eventual fluxo de *bits* zero; assim o receptor nunca saberia quando viria o próximo caractere, portanto, não teria como identificar o que seria o primeiro *bit*. Para resolver esses problemas de transmissão assíncrona, foi padronizado que, na ausência de caracteres a serem transmitidos, o transmissor mantém a linha sempre no estado 1, isto é, transmite ininterruptamente *bits* 1, o que a distingue também de linha interrompida. Quando for transmitir um caractere, para permitir que o receptor reconheça o início do caractere, o transmissor insere um *bit* de partida (*start bit*) antes de cada caractere. Convenciona-se que esse *start bit* será um *bit* zero, interrompendo assim a sequência de *bits* 1 que caracteriza a linha livre (*idle*). Para maior segurança, ao final de cada caractere, o transmissor insere um (ou dois, dependendo do padrão adotado) *bit* de parada (*stop bits*) e convenciona-se ser *bits* 1 para distinguí-los dos *bits* de partida. Os *bits* de informação são transmitidos em intervalos de tempo uniformes entre o *start bit* e o(s) *stop bit*(s). Portanto, o transmissor e o receptor somente estarão sincronizados durante o intervalo de tempo entre os *bits* de *start* e *stop*. A transmissão assíncrona também é conhecida como *"start-stop"*. A Figura 4.4 exemplifica essa transmissão.

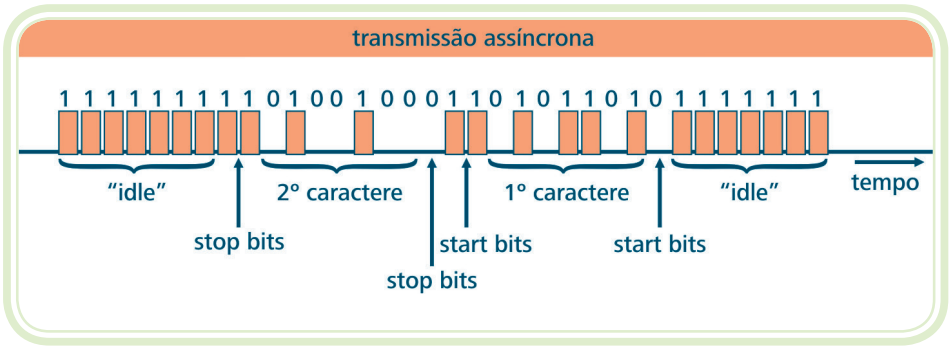


Figura 4.4: Transmissão assíncrona
Fonte: CTISM

A taxa de eficiência de uma transmissão de dados é medida como a relação de número de *bits* úteis dividido pelo total de *bits* transmitidos. No método assíncrono, a eficiência é menor do que no método síncrono, uma vez que há necessidade de inserir os *bits* de partida e parada, de forma que a cada caractere são inseridos de 2 a 3 *bits* que não contêm informação.

4.5 Transmissão *simplex*, *half-duplex* e *full-duplex*

Uma comunicação é dita *simplex* quando permite comunicação apenas em um único sentido, tendo em uma extremidade um dispositivo transmissor (*transmitter*) e do outro um dispositivo receptor (*receiver*). Não há possibilidade de o dispositivo receptor enviar dados ou mesmo sinalizar se os dados foram recebidos corretamente. Transmissões de rádio e televisão são exemplos de transmissão *simplex*.

Uma comunicação é dita *half-duplex* (também chamada *semi-duplex*) quando existem em ambas as extremidades dispositivos que podem transmitir e receber dados, mas não simultaneamente. Durante uma transmissão *half-duplex*, em determinado instante, um dispositivo A será transmissor e o outro B será receptor; em outro instante os papéis podem inverter-se, por exemplo, o dispositivo A poderia transmitir dados que B receberia; em seguida, o sentido da transmissão seria invertido e B transmitiria para A a informação se os dados foram corretamente recebidos ou se foram detectados erros de transmissão. A operação de troca de sentido de transmissão entre os dispositivos é chamada de *turn-around* e o tempo necessário para os dispositivos chavearem entre as funções de transmissor e receptor é chamado de *turn-around time*.

Uma transmissão é dita *full-duplex* (também chamada apenas *duplex*) quando dados podem ser transmitidos e recebidos simultaneamente em ambos os sentidos. Poderíamos entender uma linha *full-duplex* como funcionalmente equivalente a duas linhas *simplex*, uma em cada direção. Como as transmissões podem ser simultâneas em ambos os sentidos e não existe perda de tempo com *turn-around*, uma linha *full-duplex* pode transmitir mais informações por unidade de tempo (maior *throughput*) do que uma linha *half-duplex*, considerando-se a mesma taxa de transmissão de dados. Veja a Figura 4.5.

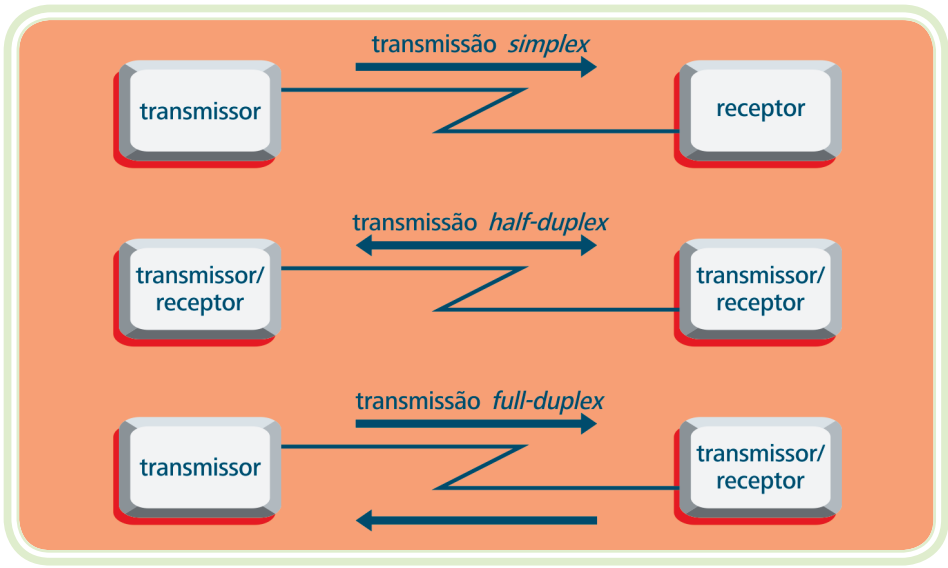


Figura 4.5: Transmissão *simplex*, *half-duplex* e *full-duplex*
Fonte: CTISM

Resumo

Sabemos que o usuário comunica-se com o núcleo do computador (composto por UCP e memória principal) através de dispositivos de entrada e saída (dispositivos de E/S ou I/O *devices*).

Nessa aula, analisamos como funcionam os dispositivos de entrada e saída e como é feita a comunicação entre eles e o núcleo do computador.

Finalizamos também o estudo dos princípios da Tecnologia da Informática. Vimos que os dispositivos de entrada e saída têm como funções básicas a comunicação do usuário com o computador, a comunicação do computador com o meio e o armazenamento de dados. Portanto, após termos estudado *hardware* e *software*, vimos quais as variáveis envolvidas, quando interagimos com o computador, ou seja, de que maneira os nossos comandos entram no computador, e como podemos ter acesso aos resultados do processamento das informações pelo computador.

Atividades de aprendizagem

1. Quais as vantagens e desvantagens das transmissões paralela e serial? Na sua opinião, a transmissão síncrona é mais confiável que a assíncrona? Justifique.

2. Diferencie entre si as transmissões *simplex*, *half-duplex* e *full-duplex*.

3. Baseado nos conhecimentos adquiridos e em pesquisas realizadas na internet, explique por que a interface serial ATA (SATA) pode ser mais rápida que a interface paralela IDE.

Referências

DAVID A.; HENNESSY, J. L. **Computer Organization & Design**: The hardware/software Interface. New York: Morgan Kaufmann Publishers, Inc, 1998.

ERCEGOVAC, M.; LANG, T.; MORENO, J. **Introdução aos Sistemas Digitais**. Porto Alegre: Bookman, 2000.

MANO, Rui. **Curso de Tecnólogos em Processamento de Dados**. Disponível em: <http://wwwusers.rdc.pucrio.br/rmano/rmano.html>. Acesso em: Jul.2010.

STALLINGS, W. **Arquitetura e Organização de Computadores**. 5. ed. São Paulo: Makron Books, 2002.

TANENBAUM, A. S. **Organização Estruturada de Computadores**. Rio de Janeiro: LTC, 2001.

TOCCI, R. J. **Sistemas Digitais**: princípios e aplicações. 10. ed. São Paulo: Pearson, 2007.

WAKERLY, J. F. **Digital Design**: principles and practices. 3rd Edition. New York: Prentice Hall, 2000.

Currículo do professor-autor

Saul Azzolin Bonaldo é professor do Colégio Técnico Industrial (CTISM) da Universidade Federal de Santa Maria (UFSM). É graduado em Engenharia Elétrica e mestre em Engenharia Elétrica pela UFSM. Trabalhou por vários anos na iniciativa privada, especialmente no projeto e execução de instalações elétricas em baixa tensão, redes lógicas e sistemas de segurança eletrônica, adquirindo boa experiência em gestão empresarial e no acompanhamento e execução de obras. Foi Inspetor do CREA-RS. No CTISM ministra as disciplinas de Eletrônica, Circuitos Digitais, Máquinas Elétricas e Projetos Elétricos. Atua também como Coordenador do Curso Técnico em Automação Industrial. É Membro do IEEE (*The Institute of Electrical and Electronics Enginners*) e filiado ao IAS (*Industry Application Society*), ao PELS (*Power Electronics Society*) e ao IES (*Industrial Electronics Society*). É revisor da revista *Potentials*, publicada pelo IEEE, e da *Industrial Electronic Magazine*, publicada pelo IES-IEEE.