



MÓDULO DE:

ARQUITETURA DE COMPUTADORES

AUTORIA:

FILIPE DE CASTRO FERREIRA

Copyright © 2009, ESAB – Escola Superior Aberta do Brasil

Módulo de: Arquitetura de Computadores

Autoria: Filipe de Castro Ferreira

Primeira edição: 2009

CITAÇÃO DE MARCAS NOTÓRIAS

Várias marcas registradas são citadas no conteúdo deste Módulo. Mais do que simplesmente listar esses nomes e informar quem possui seus direitos de exploração ou ainda imprimir logotipos, o autor declara estar utilizando tais nomes apenas para fins editoriais acadêmicos.

Declara ainda, que sua utilização tem como objetivo, exclusivamente na aplicação didática, beneficiando e divulgando a marca do detentor, sem a intenção de infringir as regras básicas de autenticidade de sua utilização e direitos autorais.

E por fim, declara estar utilizando parte de alguns circuitos eletrônicos, os quais foram analisados em pesquisas de laboratório e de literaturas já editadas, que se encontram expostas ao comércio livre editorial.

Copyright © 2008, ESAB – Escola Superior Aberta do Brasil

Apresentação

Falar sobre Arquitetura de Computadores é, no mínimo, uma tarefa desafiadora. Por um lado, existem muitos produtos que podem ser considerados como computadores. Por outro lado, a rápida evolução em todos os aspectos da tecnologia de computadores mostra-se praticamente sem fronteiras, a cada dia surgem novidades.

Apesar da grande variedade e da rapidez da evolução da área de computadores, alguns componentes básicos são comuns aos computadores, seja um micro ou um supercomputador.

Este Módulo visa promover uma discussão sobre os conceitos fundamentais de arquitetura de computadores

O**bjetivo**

Proporcionar ao aluno uma visão geral do tema Arquitetura de Computadores, entendendo os principais componentes e seus conceitos. Este entendimento é base para qualquer estudo no campo da informática.

Este Módulo de estudos não pretende abordar todos os assuntos relacionados à disciplina Arquitetura de Computadores. Esta é uma disciplina abrangente e seria impossível resumir-la em um único módulo.

Ementa

Breve histórico do computador; Arquitetura da Máquina de Von Neumann; Conceito de bit, byte, caractere e palavra; Representação numérica nos computadores, relação dos números reais com os números binários; Técnicas para efetuar operações matemáticas com os números binários; Sinal e Magnitude; Complemento a um; Complemento de dois; Hierarquia das memórias, organização e características; Classificação das Memórias quanto a leitura e escrita; Funcionamento e características da memória principal do computador; Funcionamento da memória cache e a sua relação com a memória principal; Funcionamento dos registradores especiais do processador, responsáveis por organizarem o controle da execução das instruções; Representação da Instrução e análise do Ciclo da Instrução; Tipos de modos de endereçamento; Pipelining; Organização da interface de E/S; O controle das conexões entre o processador, a memória e os outros dispositivos; Maneiras de comunicação da informação pelos dispositivos do computador; E/S com Polling e Interrupção; E/S com acesso direto à memória; Tipos de barramentos e suas diferenças; Formas do computador executar um programa; Linguagens de Programação e Tradutores; Compilação, Ligador e Interpretação; Comparação entre Compilação e Interpretação; Máquinas Virtuais e Java Bytecode.

Sobre o Autor

PMP; Pós-Graduado em Engenharia de Sistemas; MBA em Gerência de Projetos; Bacharel em Sistemas de Informação.

Gerente de Projetos e ex-Consultor Microsoft Brasil. Certificado Microsoft em SharePoint 2007, Project Server 2007 (EPM 2007) e MS Project 2007.

Experiência como Gerente de Fábrica de Software, Analista em Projetos de Desenvolvimento de Sistemas e Conteúdos para EAD.

Experiência como Projetista e Administrador de Redes Windows e Linux.

Tutor na ESAB (Escola Superior Aberta do Brasil) em Cursos de Pós-Graduação.

SUMÁRIO

| | |
|---|-----------|
| UNIDADE 1 | 9 |
| Breve Histórico I..... | 9 |
| UNIDADE 2 | 12 |
| Breve Histórico II..... | 12 |
| UNIDADE 3 | 17 |
| Componentes de um Computador – Máquina de Von Neumann I..... | 17 |
| UNIDADE 4 | 21 |
| Componentes de um Computador – Máquina de Von Neumann II..... | 21 |
| UNIDADE 5 | 25 |
| Elementos Básicos: representação dos dados | 25 |
| UNIDADE 6 | 29 |
| Representação Numérica..... | 29 |
| UNIDADE 7 | 33 |
| Operações Binárias..... | 33 |
| Overflow ou estouro do limite..... | 34 |
| UNIDADE 8 | 36 |
| Representação de Sinal Negativo I..... | 36 |
| <i>Sinal e Magnitude</i> | 36 |
| UNIDADE 9 | 39 |
| Representação de Sinal II..... | 39 |
| UNIDADE 10 | 42 |
| Sistema de Memória | 42 |
| UNIDADE 11 | 45 |
| Tipos de Memórias I..... | 45 |
| UNIDADE 12 | 47 |
| Tipos de Memórias II – Memória Principal | 47 |
| Capacidade da Memória Principal..... | 48 |
| UNIDADE 13 | 50 |
| Tipos de Memórias III - Memória Cache | 50 |

| | |
|---|-----------|
| UNIDADE 14 | 54 |
| Registradores..... | 54 |
| UNIDADE 15 | 57 |
| Representação e Ciclo de Instrução | 57 |
| Representação da Instrução | 57 |
| Formato das Instruções..... | 58 |
| Ciclo de Instrução..... | 59 |
| UNIDADE 16 | 61 |
| Modos de Endereçamento | 61 |
| Modo de endereçamento por registrador | 63 |
| Modo por registrador direto | 64 |
| Modo por registrador indireto | 64 |
| UNIDADE 17 | 65 |
| Pipelining..... | 65 |
| Método Assíncrono | 67 |
| Método Síncrono..... | 67 |
| UNIDADE 18 | 69 |
| Organização de uma Interface de E/S (Comunicação entre a Memória e UCP – Barramentos) | 69 |
| UNIDADE 19 | 72 |
| Entrada/Saída | 72 |
| UNIDADE 20 | 74 |
| Formas de Comunicação | 74 |
| UNIDADE 21 | 78 |
| Técnicas de Transferência de Dados I..... | 78 |
| UNIDADE 22 | 81 |
| Técnicas de Transferência de Dados II..... | 81 |
| UNIDADE 23 | 83 |
| Técnicas de Transferência de Dados III..... | 83 |
| UNIDADE 24 | 86 |
| Padrões de Barramento I | 86 |
| UNIDADE 25 | 88 |
| Padrões de Barramento II | 88 |
| UNIDADE 26 | 92 |

| | |
|---------------------------------|------------|
| Execução de Programas I | 92 |
| UNIDADE 27 | 96 |
| Execução de Programas II | 96 |
| UNIDADE 28 | 100 |
| Execução de Programas III | 100 |
| UNIDADE 29 | 105 |
| Execução de Programas IV | 105 |
| UNIDADE 30 | 108 |
| Execução de Programas V | 108 |
| GLOSSÁRIO | 112 |
| BIBLIOGRAFIA..... | 113 |

UNIDADE 1

Breve Histórico I

Objetivo: Conhecer a história do computador.

Ao longo dos séculos foram construídas algumas máquinas com a função de realizarem cálculos aritméticos e outras operações inteligentes. A primeira calculadora mecânica foi criada pelo alemão Wilhelm Schickard (1592-1635) em 1623. O funcionamento da máquina era baseado em rodas dentadas e ela era capaz de efetuar adições e subtrações. A invenção de Schickard, no entanto, não foi muito difundida e caiu rapidamente no esquecimento. Posteriormente, Blaise Pascal desenvolve a “Pascoalina”, uma máquina de calcular mecânica, também baseada em rodas dentadas, com o objetivo de livrar seu pai, coletor de impostos de Rouen (França), dos fastidiosos cálculos que sua profissão lhe impunha.

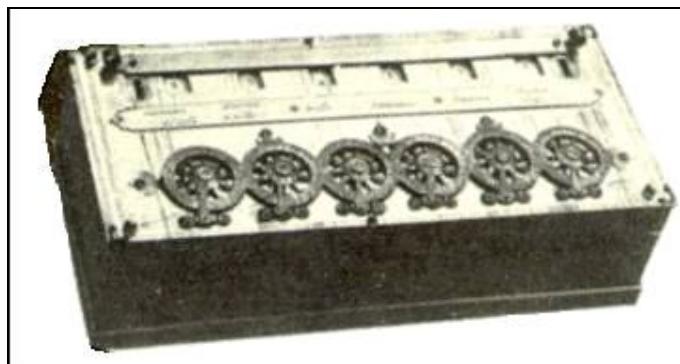


Figura 1 - Pascoalina

Entre as máquinas que antecederam os modernos computadores ou processadores eletrônicos estão, principalmente, a máquina analítica projetada na primeira metade do século XIX pelo matemático e inventor britânico Charles Babbage, que foi considerado o primeiro computador mecânico.

Charles Babbage (1792-1871) concebeu um Computador Analítico dotado de um dispositivo a que chamou de MOINHO (uma máquina de somar com precisão de até 50 casas decimais), e um dispositivo de entrada que leria cartões perfurados contendo não somente números (os dados), mas também INSTRUÇÕES (o que fazer com os dados). Imaginou ainda um dispositivo de memória que chamou de ARMAZÉM para guardar os números, um banco com 1000 "registradores" cada qual capaz de armazenar um número de 50 dígitos - os números dados pelos cartões de entrada ou então números resultados de operações do moinho. Finalmente, incluiu um dispositivo impressor para dar saída aos resultados. As instruções (gravadas em cartões) possíveis de ser implementadas pelo moinho eram:

- Entrar com um número no armazém
- Entrar com um número no moinho
- Mover um número do moinho para o armazém
- Mover um número do armazém para o moinho
- Comandar o moinho para executar uma operação
- Sair com um resultado

Para construir um dispositivo a partir destas ideias, Babbage contou com a colaboração inestimável da matemática Ada Augusta Byron, Lady Lovelace, filha do poeta Lord Byron. Ada desenvolveu séries de instruções para o calculador analítico, criando conceitos tais como sub-rotinas, loops e saltos condicionais.

Babbage é considerado o precursor do computador. Ada é considerada a precursora do software.

Também há a máquina tabuladora do americano Herman Hollerith, que trabalhava no departamento de censo dos Estados Unidos e idealizou um sistema de tratamento de informações com o qual, mediante o uso de cartões perfurados, conseguiu aumentar de dois para duzentos o número de dados processados por minuto. Esses cartões, que receberam o nome do inventor, foram utilizados pelos computadores até 1970 como sistema de entrada e saída de dados. Da companhia fundada por Hollerith, a Tabulating Machine Company, surgiu mais tarde a International Business Machines Corporation (IBM).

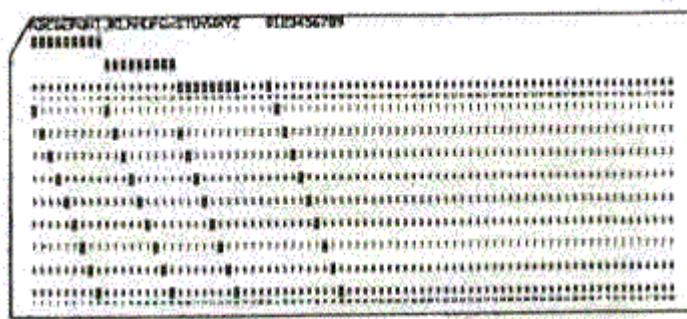


Figura 2 - Cartão Perfurado

Três décadas antes de Hollerith concretizar seu projeto, foi publicada a obra do matemático britânico George Boole, *An Investigation into the Laws of Thought* (1854; *Investigação das leis do pensamento*). Boole considerava que os processos mentais do ser humano eram resultado de uma associação sucessiva de elementos simples que se podiam expressar sobre uma base de duas únicas alternativas: sim ou não. Foi essa a origem do método matemático de análise formal conhecido como álgebra de Boole. Considerado na época uma simples curiosidade, o método viria a constituir o fundamento teórico da informática moderna.

UNIDADE 2

Breve Histórico II

Objetivo: Conhecer o computador no século XX.

Na evolução dos equipamentos de informática tornou-se habitual referir-se às etapas de desenvolvimento como "gerações", embora nem sempre haja acordo quanto a seu número ou quanto aos critérios utilizados em sua diferenciação.

Neste Módulo não serão classificados as gerações, mas o período em que foram criados.

Os primeiros computadores eram caracterizados pelo uso de válvulas a vácuo.

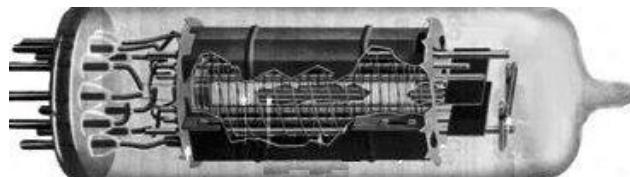


Figura 3 - Foto de válvula usada na década de 40.

1943 a 1945 - ENIAC

J. Presper Eckert, John V. Mauchly e Herman H. Goldstine, nos Estados Unidos, construíram o ENIAC (Electronic Numerical Integrator Computer), considerado o primeiro computador programável universal.

Durante a Segunda Guerra Mundial havia a necessidade, por parte do exército americano de desenvolver métodos mais rápidos para cálculo das trajetórias e alcance das novas armas. Esse trabalho era feito manualmente, utilizando-se calculadores de mesa. O Army's Ballistics

Research Laboratory (BRL) utilizava cerca de 200 pessoas para resolver as equações necessárias, sendo que a preparação das tabelas para uma única arma poderia levar várias horas ou até mesmo dias de trabalho de uma única pessoa.

Em 1943 foi aceito pelo exército americano o projeto para a construção do ENIAC. O ENIAC possuía aproximadamente 17.480 válvulas de rádio, pesava 4 toneladas, media 30 metros de comprimento por 3 de largura e ocupava uma área de 180m², chegando a consumir 150 KW. Em contrapartida conseguia realizar 5 mil operações por segundo.



Figura 4 - Parte do Galpão que abrigava o ENIAC.

O ENIAC era uma máquina decimal e não uma máquina binária, onde cada dígito era representado por um anel de 10 válvulas. A cada instante, apenas uma válvula ficava em estado “ON” (ligado), representando um dos dez dígitos. A principal desvantagem do ENIAC era que ele tinha que ser programado manualmente, ligando e desligando chaves e conectando e desconectando cabos. A programação do ENIAC era feita através de 6.000 chaves manuais. A cada novo cálculo, era preciso reprogramar várias destas chaves. Isso sem falar no resultado, que era dado de forma binária através de um conjunto de luzes.

Não foi à toa que a maior parte dos programadores da época eram mulheres, só mesmo elas para ter a paciência necessária para programar e reprogramar esse emaranhado de chaves várias vezes ao dia.

O ENIAC funcionava da seguinte maneira: Primeiro um grupo de cientistas desenvolvia equações matemáticas, na exata sequência em que elas tinham que ser digeridas pelo sistema. A seguir seis especialistas programavam o computador para executá-las, girando botões de sintonia e plugando centenas de fios nas tomadas corretas. Portanto o que hoje é o sistema operacional, em 1946 era uma operação completamente manual. O primeiro teste do ENIAC – uma demonstração feita para generais das Forças Armadas – calculou a trajetória de uma bala de canhão até um alvo determinado. Alimentado com as equações, o computador forneceu os dados para que o canhão fosse calibrado. A bala acertou o alvo, mas o que impressionou os generais foi que o tempo que o computador levou para resolver. Foi menor que o tempo da trajetória da bala. O único problema do ENIAC era que para calcular a trajetória de outra bala, até um novo alvo, tudo tinha que ser refeito: das equações até o (re)acerto dos fios.

1945 a 1952 – EDVAC

J. Presper Eckert, John V. Mauchly (engenheiros eletrônicos) e Herman H. Goldstine com a consultoria de John von Neumann, Doutor em Matemática, pela Universidade de Budapeste, húngaro, naturalizado americano, especialista em Lógica, construíram o EDVAC (Electronic Discrete Variable Computer. Em 1945 Von Neumann, que foi um dos principais consultores do projeto ENIAC, sugeriu que o sistema binário fosse adotado em todos os computadores, e que as instruções e dados fossem compilados e armazenados internamente no computador, na sequência correta de utilização. Estas sugestões tornaram-se a base filosófica para projetos de computadores.

1946 - Manchester Mark I

Max Newman e a equipe da Universidade de Manchester, na Inglaterra, que teve a participação de Alan Turing, construíram o Manchester Mark I, "primeiro computador que funcionou", que teve "a primeira visualização na tela de dados contidos na memória" e o

primeiro programa gravado executado em 21 de junho de 1948. O Harvard Mark I, parcialmente financiado pela IBM. O Mark I tinha o nome técnico de Calculador Automático Sequencial Controlado, e foi construído entre 1939 e 1944, pelo professor Howard Aiken. O Mark I talvez tenha sido a maior máquina calculadora já construída (20 metros de comprimento por 3 de altura e 750 mil componentes). Independente dessa discussão a participação da IBM nos desenvolvimentos dos computadores é inegável.

A IBM também teria outro destaque na história dos computadores por um acontecimento: foi em um Mark II em que ocorreu o primeiro bug.

A palavra já vinha sendo usada como gíria, significando qualquer complicaçāo, desde a Revolução Industrial. As máquinas eram instaladas em locais onde havia muitos insetos voando e havia grande chance de que algum isento pousar em um lugar errado e causar estragos era grande, e qualquer parada mecânica era atribuída, a princípio, por bug. Com os computadores realmente foi um bug: Em 1945, uma mariposa conseguiu entrar num Mark II do Centro Naval de Virgínia, nos EUA, e travou todo o sistema, a partir daí o nome passaria a ser sinônimo de qualquer falha ou erro.



Dica

Vídeo

Assista ao vídeo “História dos Computadores” através do link “Estudo Complementar”.

Vale à pena!



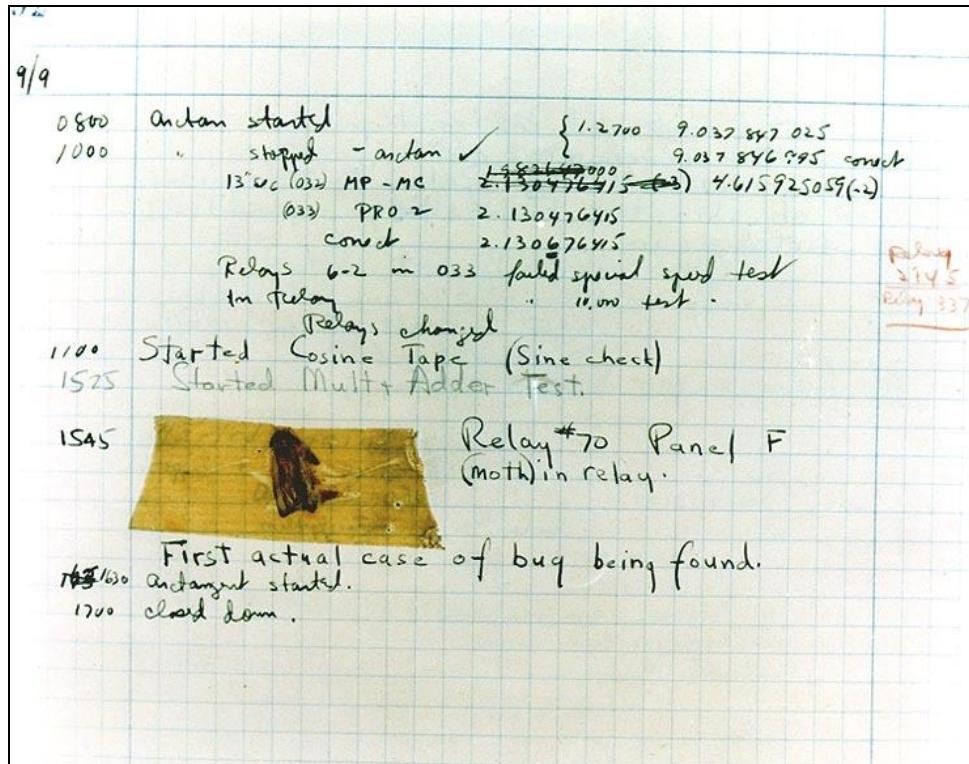


Figura 5 – Imagem do inseto que ocasionou o primeiro bug.



Dica

Caso tenha interesse em ler um histórico mais detalhado sobre os computadores, não deixe de acessar o arquivo “Histórico dos Computadores” através do link “Estudo Complementar”.



UNIDADE 3

Componentes de um Computador – Máquina de Von Neumann I

Objetivo: Conhecer a estrutura da arquitetura da Máquina de Von Neumann



MEDIATECA

VÍDEO

Atenção! Antes de dar continuidade aos seus estudos. Vá ao ambiente: CAMPUS ON-LINE e assista ao vídeo referente à UNIDADE 3.



A máquina de Von Neumann (John von Neumann) é a arquitetura base utilizada pelos computadores atuais.

É uma arquitetura de utilização genérica que permite resolver qualquer problema que se possa exprimir sob a forma de um algoritmo. Permite especificar programas que indicam as desejadas sequências de operações que devem ser realizadas, sobre a informação representada em binário.

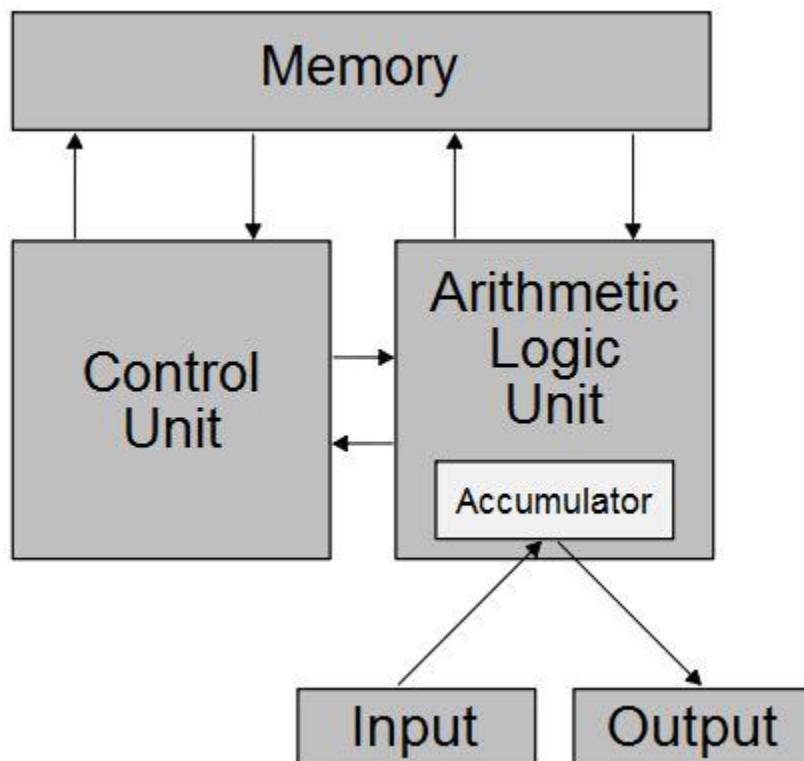


Figura 6 – Arquitetura de Von Neumann



Dica

Caso queira saber mais sobre este importante cientista acesse o link:

http://pt.wikipedia.org/wiki/John_von_Neumann

Ou a biografia completa em vídeo. http://www.youtube.com/watch?v=RF_CZpmVGzw

Apesar de estar em italiano, este material é muito bom e de fácil compreensão.



A máquina proposta por Von Neumann reúne os seguintes componentes: Memória Principal, Central de Processamento (UCP), composta pela Unidade Lógica Aritmética (ULA) e a Unidade de Controle (UC) e Dispositivos de Entrada e Saída.

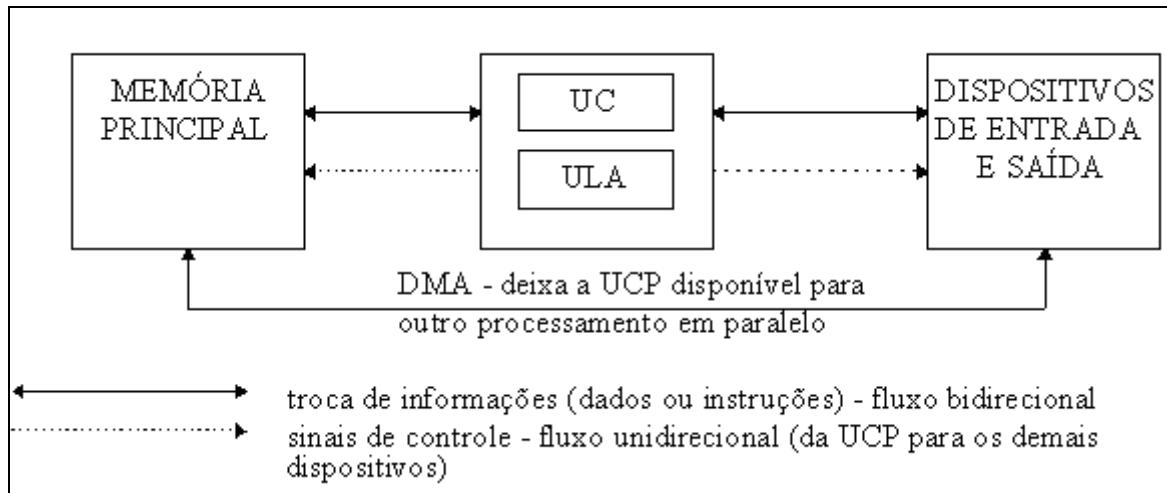


Figura 7 – Máquina de John Von Neumann

Memória Principal

A unidade de memória central serve para guardar programas e dados, sob a forma de uma representação binária. Cada instrução da máquina é codificada como uma sequência de bits. Cada valor de certo tipo é codificado por uma determinada sequência de bits.

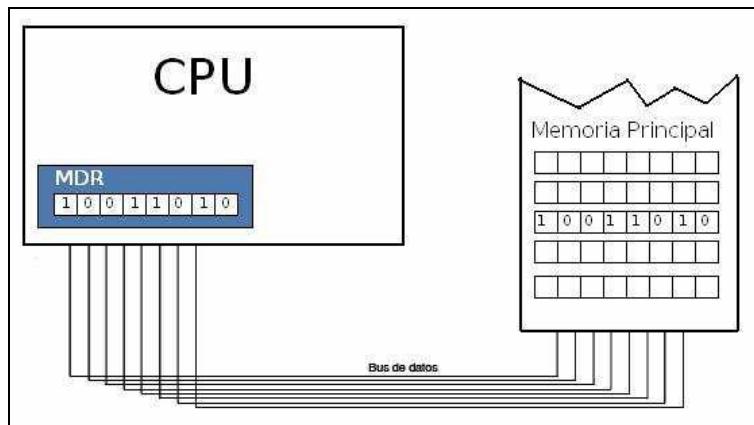


Figura 8 – Sequência de bits armazenados na memória principal



Figura 9 – Exemplos de Memória Principal

UNIDADE 4

Componentes de um Computador – Máquina de Von Neumann II

Objetivo: Conhecer mais dados sobre a Arquitetura da Máquina de Von Neumann

Unidade Central de Processamento (UCP)

A unidade de central de processamento (CPU – Central Processing Unit) trata do controle global das operações e da execução das instruções. Para esse efeito, a UCP contém as seguintes unidades internas:

- Unidade Lógica e Aritmética, ULA (ALU- arithmetic and logic unit): A ULA executa as principais operações lógicas e aritméticas do computador. Ela soma, subtrai, divide, determina se um número é positivo ou negativo ou se é zero. Além de executar funções aritméticas, uma ULA deve ser capaz de determinar se uma quantidade é menor ou maior que outra e quando quantidades são iguais. A ULA pode executar funções lógicas com letras e com números.
- Unidade de Controle – UC: Responsável por gerar todos os sinais que controlam as operações no exterior do CPU, e ainda por dar todas as instruções para o correto funcionamento interno do CPU.

A unidade de controle executa três ações básicas intrínsecas e pré-programadas pelo próprio fabricante do processador, são elas: busca (fetch), decodificação e execução. Assim sendo, todo processador, ao iniciar sua operação, realiza uma operação cíclica, tendo como base essas três ações. Dependendo do tipo de microprocessador, a unidade de controle pode ser fixa ou programável.

Cada instrução da máquina pode envolver:

- Operações aritméticas e lógicas
- Operações de transferência entre CPU e a Memória Central
- Operações de transferência entre CPU e Unidades de I/O
- Operações de controle da sequência de execução das instruções



Figura 10 – Exemplo de hardware de CPU

Entrada e Saída

As unidades periféricas destinam-se a suportar as ações de comunicação da CPU e memória com o exterior, daí, a sua designação de unidades de entrada e saída (ou I/O- input / output).

São exemplos, o teclado, o monitor, o mouse, a impressora, as interfaces de comunicação com redes de computadores. Também, há unidades periféricas destinadas ao armazenamento de dados, que, depois, são apresentados ao usuário, sob a forma de

arquivos, geridos pelos programas do Sistema Operacional. Exemplos: os discos rígidos, discos ópticos (CD/DVD), os discos flexíveis e as fitas magnéticas.

Exemplos de dispositivos de Entrada/Saída (Input/Output):

- Monitor Saída (Output)
- Teclado Entrada (Input)
- Impressora Saída (Output)
- Disco Rígido Entrada e Saída (Input/Output – I/O)
- Disco Flexível Entrada e Saída (Input/Output – I/O)
- Mouse Entrada (Input)
- Interface de Rede Entrada e Saída (Input/Output)



Figura 11 – Exemplos de dispositivos de E/S



Dica

Vídeo

Assista ao vídeo “Viagem dentro do computador” através do link “Estudo Complementar”.

Vale à pena assistir!



UNIDADE 5

Elementos Básicos: representação dos dados

Objetivo: Identificar os conceitos de bit, byte, caractere e palavra.

Toda informação introduzida em um computador, sejam dados que serão processados ou instrução de um programa, precisa ser entendida pela máquina, para que possa ser corretamente, interpretada e processada.

As informações apresentadas na forma de caracteres neste módulo de estudos, por exemplo, são entendidas porque o leitor conhece o significado dos símbolos, que representam os caracteres alfabéticos e os sinais de pontuação.

O computador, sendo um equipamento eletrônico, armazena e movimenta as informações internamente sob forma eletrônica; esta pode ser um valor de voltagem ou de corrente.

Para que a máquina pudesse representar eletricamente todos os símbolos utilizados na linguagem humana, seriam necessários diferentes valores de voltagem (ou de corrente). Tal máquina certamente seria difícil de ser construída para fins comerciais e, possivelmente, teria muito baixa confiabilidade. Esta foi uma das grandes desvantagens do primeiro computador eletrônico construído, o Eniac.

Von Neumann e equipe consideravam muito mais simples e confiável projetar um circuito capaz de gerar e manipular o menor número possível de valores distintos, capaz de entender apenas dois valores diferentes: 0 e 1.

Dessa forma, os computadores digitais são binários. Toda informação introduzida em um computador é convertida para um algarismo binário ou dígito binário, conhecido como bit. O bit pode ter então somente dois valores: 0 e 1.

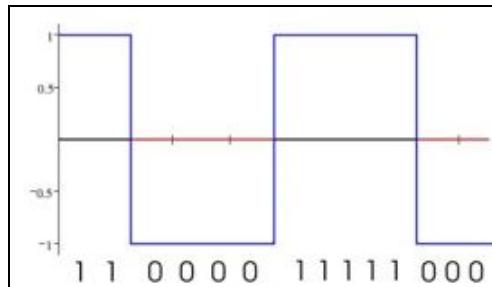


Figura 12 – Representação do bit 0 e 1

Um caractere isolado não significa nada para nosso sentido de comunicação, razão pela qual se criam palavras. Da mesma forma, as informações manipuladas por um computador são codificadas em grupos ordenados de bits, de modo a terem um significado útil.

A primeira definição formal atribuída a um grupo ordenado de bits, para efeito de manipulação interna mais eficiente, foi instituída pela IBM e é, atualmente, utilizada por praticamente todos os fabricantes de computadores. Trata-se do byte, definido como um grupo ordenado de 8 bits, tratados de forma individual, como unidade de armazenamento e transferência.

Como os computadores são máquinas binárias, todas as indicações numéricas referem-se a potências de 2 e não a potências de 10 (como no sistema métrico), por essa razão, a medida Kbyte representa 1.024 byte ($2^{10} = 1024$ bytes = 8192 bits) e não 1000 bytes, o Mbyte (abreviatura do termo mega) representa 1.048.576 bytes (valor igual a 1024×1024 ou $2^{10} \times 2^{10} = 220$) e o giga, representado pelo caractere G, indica o valor igual a 1024 mega ou 1.048.576K ou 230.

| Prefixo binário (IEC) | | | Prefixo do SI | | |
|-----------------------|---------|----------|---------------|---------|-----------|
| Nome | Símbolo | Múltiplo | Nome | Símbolo | Múltiplo |
| byte | B | 2^0 | byte | B | 10^0 |
| kibibyte(quilobyte) | KiB | 2^{10} | quilobyte | kB | 10^3 |
| mebibyte(megabyte) | MiB | 2^{20} | megabyte | MB | 10^6 |
| gibibyte(gigabyte) | GiB | 2^{30} | gigabyte | GB | 10^9 |
| tebibyte(terabyte) | TiB | 2^{40} | terabyte | TB | 10^{12} |
| pebibyte(petabyte) | PiB | 2^{50} | petabyte | PB | 10^{15} |
| exbibyte(exabyte) | EiB | 2^{60} | exabyte | EB | 10^{18} |
| zebibyte(zettabyte) | ZiB | 2^{70} | zettabyte | ZB | 10^{21} |
| yobibyte(yottabyte) | YiB | 2^{80} | yottabyte | YB | 10^{24} |

Figura 13 – Múltiplos de bytes



Dica

Caso queira saber mais sobre bit, byte, suas potências e conversões, acessem o link:

<http://pt.wikipedia.org/wiki/Byte>



Na computação também se criou o conceito de palavra. Assim, além do bit e do byte, temos o conceito relacionado com o armazenamento e a transferência de informações.

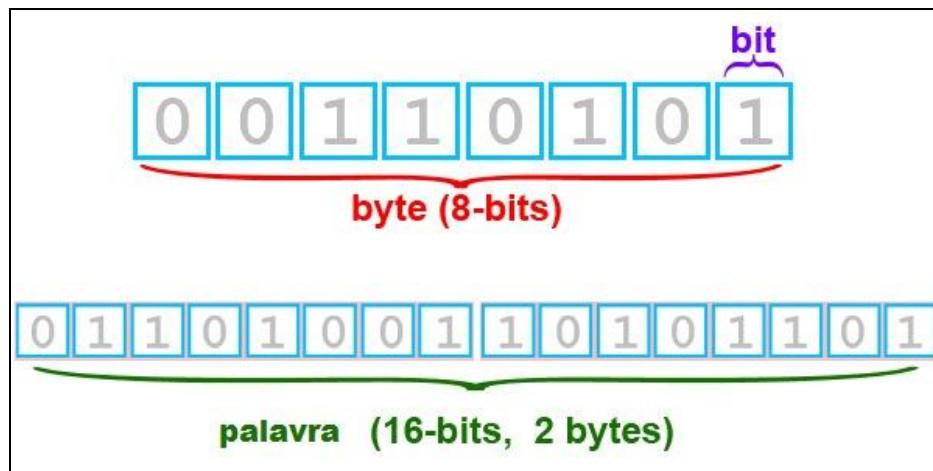


Figura 14 – Bit, Byte e Palavra.

A palavra nos computadores é um valor fixo e constante para um dado processador (16, 32 ou 64 bits).

De modo geral, usam-se dois valores diferentes: um relacionado à unidade de armazenamento – o byte; e outro para indicar a unidade de transferência e processamento – a palavra. Em geral, o computador processa valores representados por uma quantidade de bits igual à palavra, indicando assim a capacidade de processamento do sistema, por exemplo, 32 bits ou 64 bits.

Uma palavra deve representar um dado ou uma instrução, que pode ser processada, armazenada ou transferida em uma única operação.

UNIDADE 6

Representação Numérica

Objetivo: Representar numericamente, nos computadores, relação dos números reais com os números binários.

Os sistemas numéricos são sistemas de notação usados para representar quantidades abstratas denominadas números. Usa-se a base para definir o sistema numérico e esta indica a quantidade de símbolos existentes. No dia a dia usa-se 10 símbolos numéricos (de 0 até 9), que formam o sistema numérico decimal.

O Sistema Decimal

Na base decimal os números expressam potências de 10 em sua representação, como se pode observar nos exemplos abaixo:

| Número | ... | $10^3 *$ | | $10^2 *$ | | $10^1 *$ | | $10^0 *$ | | $10^{-1} *$ | | $10^{-2} *$ | ... |
|--------|-----|----------|---|----------|---|----------|---|----------|---|-------------|--|-------------|-----|
| 1758 | | 1 | + | 7 | + | 5 | + | 8 | + | | | + | |

O significado de cada número é:

$$1758 = (1 * 10^3) + (7 * 10^2) + (5 * 10^1) + (8 * 10^0) =$$

$$= 1000 + 700 + 50 + 8 = 1758$$

O Sistema Binário

Pelo fato do microcomputador ser constituído de componentes eletrônicos digitais, torna-se necessário o uso de um sistema numérico com 2 dígitos: o sistema binário.

O sistema binário, ou de base 2, funciona de maneira análoga ao decimal, porém com apenas dois dígitos, 0 e 1. Neste caso, o dígito 0 é representado pela não presença de tensão elétrica enquanto o 1 pela presença de tensão elétrica. Os dígitos de um sistema binário são chamados de bit, de Binary digit (dígito binário).

A representação de números em bases diferentes pode induzir à confusão, portanto coloca-se um número subscrito para indicar a base de numeração adotada. Por exemplo, 74_{10} e 323_{10} são números representados na base decimal; 10011_2 e 10_{12} são números representados na base binária.

Conversão de base binária para decimal

Para se converter um número, na base binária para a base decimal, deve-se escrever cada número que compõe (o bit), multiplicado pela base2 (base do sistema), elevado à posição que ocupa.

Uma posição à esquerda da vírgula representa uma potência e à direita, uma potência negativa. A soma de cada multiplicação de cada dígito binário pelo valor das potências resulta no número real representado.

Por exemplo:

$$1\ 0\ 1\ 1_2 = (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (1 * 2^0) = 11_{10}$$

$$1\ 0\ 0_2 = (1 * 2^2) + (0 * 2^1) + (0 * 2^0) = 4_{10}$$

Conversão de base decimal para binária

Para converter números decimais em binários, deve-se tratar o quociente e a parte fracionaria separadamente. Por exemplo, para converter 53₁₀ em binário faz-se:

| Operação | Quociente | Resto |
|----------|-----------|-------|
| 53/2 | 26 | 1 |
| 26/2 | 13 | 0 |
| 13/2 | 6 | 1 |
| 6/2 | 3 | 0 |
| 3/2 | 1 | 1 |
| ½ | 0 | 1 |

O resultado é a montagem do número a partir do último resto da divisão para o primeiro, ou seja, $53_{10} = 1\ 1\ 0\ 1\ 0\ 1_2$.

O Sistema Hexadecimal

Embora o sistema binário seja conveniente para os computadores, é excessivamente inconveniente para os seres humanos. Por isto a notação hexadecimal é preferida pelos profissionais de computação.

A base decimal, embora mais compacta que a binária, é desconfortável devido à tediosa conversão entre as bases. Neste caso, adotou-se uma notação conhecida como hexadecimal, onde os dígitos binários são agrupados em conjuntos de quatro. A cada combinação possível de quatro dígitos binários é atribuído um símbolo.

| | |
|----------|----------|
| 0000 = 0 | 1000 = 8 |
| 0001 = 1 | 1001 = 9 |
| 0010 = 2 | 1010 = A |
| 0011 = 3 | 1011 = B |
| 0100 = 4 | 1100 = C |
| 0101 = 5 | 1101 = D |
| 0110 = 6 | 1110 = E |
| 0111 = 7 | 1111 = F |

Conversão de base binária para hexadecimal

Para converter números binários em hexadecimal deve-se agrupar os dígitos binários quatro a quatro, a partir da vírgula binária e substituir cada grupo pelo seu equivalente hexadecimal.

$$0100\ 1111\ 1100_2 = 4FC_{16}$$

Conversão de base hexadecimal para binária

Para converter hexadecimal em binário, deve-se substituir cada dígito hexadecimal pelo binário equivalente de quatro dígitos.

$$DEA_{16} = 1101\ 1110\ 1010_2$$

UNIDADE 7

Operações Binárias

Objetivo: Conhecer as técnicas para efetuar operações matemáticas com os números binários.

As operações em base 2 podem ser feitas utilizando-se as mesmas técnicas empregadas na aritmética em base 10 - o raciocínio é exatamente o mesmo. O método apresentado é o mais fácil para os seres humanos resolverem problemas envolvendo números binários.

Para os computadores, a adição e multiplicação são implementadas utilizando-se circuitos que são análogos às técnicas utilizadas pelos seres humanos.

A operação de soma de dois números em base 2 é efetuada levando-se em conta que só há dois algarismos disponíveis (0 e 1). Abaixo, todas as possibilidades:

$$0 + 0 = 0$$

$$1 + 0 = 1$$

$$0 + 1 = 1$$

$$1 + 1 = 0, \text{ e vai "um" ou } 1\ 0_2$$

Um exemplo da soma de dois números na base binária: $1001_2 + 0101_2$:

$$\begin{array}{r}
 & & 1 & & \leftarrow \text{Transporte do Bit (vai 1)} \\
 & 1 & 0 & 0 & 1 & \\
 + & 0 & 1 & 0 & 1 & \\
 \hline
 & 1 & 1 & 1 & 0 & \\
 & & & & & 14_{10} \\
 \end{array}$$

Um mais um é igual a zero e “vai um”. O resultado corresponde exatamente ao valor esperado da base.

A subtração em base 2 é mais complexa por dispormos somente dos algarismos 0 e 1 e, com isso, $0 - 1$; necessita de “pedir emprestado”, de um valor igual à base, no nosso caso 2, obtido do próximo algarismo diferente de zero à esquerda. A operação é exatamente a mesma de uma subtração na base decimal.

Exemplo:

-1 (“vai um”)

1 1 0 1 1 0

- 1 1 0 0

1 0 1 0 1 0

Overflow ou estouro do limite

Nos exemplos acima não é levado em consideração qualquer limite ou quantidade máxima de algarismos permitida para um dado registrador. Quando uma operação aritmética resulta em um valor acima do limite máximo possível, trata-se do problema denominado **overflow** ou estouro do limite.

Quando resultado de uma adição pode ter um número de bits maior que o da palavra utilizada esta condição é denominada *overflow*. Quando ocorre um *overflow*, o processador deve sinalizar este fato, para que o resultado não seja usado.

Exemplo:

Somar os números $15_{10} + 49_{10}$ num registrador de 6 bits:

1 1 1 1 1 1 (“vai um”)

$$15 = (0\ 0\ 1\ 1\ 1\ 1)_2$$

$$\underline{49} = (\underline{1}\ \underline{1}\ 0\ 0\ 0\ 1)_2$$

$$64 = (0\ 0\ 0\ 0\ 0\ 0)_2$$

Neste caso o valor esperado (64) não pode ser representado em um registrador de 6 bits.

UNIDADE 8

Representação de Sinal Negativo I

Objetivo: Entender Sinal e Magnitude; Complemento a um.

Todos os exemplos citados nas unidades anteriores se referenciam a números **positivos**. Os **números positivos** possuem sempre a mesma representação (conforme descrito nas unidades anteriores). Para representar números **negativos**, há três formas convencionadas:

Sinal e Magnitude

Na representação sinal e magnitude o bit mais à esquerda de um número binário é reservado para indicar se o número é positivo ou negativo e é conhecido como bit de sinal, onde, convencionalmente, “0” indica um número positivo e “1” um número negativo. O restante indica a magnitude do número (ou seu valor absoluto).

| | |
|-------|-----------|
| Sinal | Magnitude |
|-------|-----------|

Figura 15 – Representação de Números Inteiros em Sinal e Magnitude

Por exemplo, para representar o número - 7 em um registrador de 6 bits:

$$-7 = 1\ 0\ 0\ 1\ 1\ 1$$

O primeiro bit foi usado para representar o sinal (negativo) e apenas 5 bits foram usados para representar o valor absoluto (7).

De forma geral pode-se expressar a faixa de um número da seguinte forma:

$$-(2^{(n-1)} - 1) \leq X \leq + (2^{(n-1)} - 1), \text{ onde } n \text{ é a quantidade de bits do número}$$

Como exemplo, para um registrador de 6 bits ($n = 6$), os limites de representação serão:

$$\text{de } -(2^{6-1} - 1) \text{ a } +(2^{6-1} - 1)$$

$$\text{de } -(2^5 - 1) \text{ a } +(2^5 - 1)$$

Faixa: de -31 a +31

A representação sinal-magnitude apresenta vantagens em relação à inversão de um número (basta inverter o bit de sinal) e em relação à determinar se o número é positivo ou negativo (basta verificar o bit de sinal). Esta representação apresenta algumas desvantagens. O número zero pode ser representado de duas maneiras distintas:

$$+0_{10} = 0000_2$$

$$-0_{10} = 1000_2$$

Por conta desta desvantagem, a representação sinal/magnitude normalmente não é usada na implementação da ULA (Unidade Lógica Aritmética).

Há também a possibilidade de ocorrer overflow sem a ocorrência do “vai um” para fora do limite do registrador. Sem que haja um estouro do limite máximo do registrador. Por exemplo, se quisermos representar a soma de “ $15 + 17 = 32$ ” com um registrador de 6 bits ocorrerá um overflow, pois como visto, no primeiro exemplo desta unidade, a faixa de valores possíveis para 6 bits é de -31 a +31.

$+15 = 0\ 0\ 1\ 1\ 1\ 1$ $\underline{+17 = 0\ 1\ 0\ 0\ 0\ 1}$ $1\ 0\ 0\ 0\ 0\ 0$

Ocorreu overflow, pois os bits da magnitude não foram suficientes para representar o valor desejado. Mas não houve estouro da capacidade do registrador, apenas mudou o bit do sinal.

Representação em complemento a um

Para se achar a representação em complemento a um, deve-se inverter todos os bits do número, incluindo o do sinal.

Por exemplo, o número +3 para -3, representado em “complemento a um” em um registrador de 6 bits:

 $+3 = 0\ 0\ 0\ 0\ 1\ 1$ $-3 = 1\ 1\ 1\ 1\ 0\ 0 \rightarrow (\text{complemento a um} - \text{todos os bits invertidos})$

Esta representação possui vantagem sobre o sinal e a magnitude, pois há apenas um componente para soma e subtração, diminuindo custo e o tempo de execução. Entretanto, assim como em sinal e magnitude, há duas representações para o número zero.

UNIDADE 9

Representação de Sinal II

Objetivo: Entender: Complemento de dois.

Representação em complemento de dois

Nesta representação, o número *negativo* (ou a negação de um número) é obtido em dois passos:

- **Primeiro passo:** Obtém-se o complemento de todos os bits do número positivo (trocando 0 por 1 e vice-versa) incluindo o bit do sinal (complemento a um)
- **Segundo passo:** Ao resultado obtido no primeiro passo soma-se 1 (em binário), desprezando-se o último transporte, se existir.

Por exemplo: Abaixo a representação em Complemento de 2 do número +10 para -10 em um registrador de 8 bits:

$$\begin{array}{rcl}
 +10 & = & 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \\
 \text{Primeiro passo} \quad -10 & = & 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \\
 \text{Segundo passo} & + & \qquad \qquad \qquad 1 \\
 \text{Resultado} \quad -10 & = & 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0
 \end{array}$$

As propriedades importantes da representação em complemento de 2 são:

- O sinal é determinado pelo bit mais significativo;
- Negar um número duas vezes produz o número original;
- O número zero tem apenas uma representação;
- Somar as representações em complemento de 2 de um número positivo e de um negativo (descartando-se o transbordo) fornece o resultado correto na representação em complemento de 2. Além disto, a subtração é tratada como uma adição.

Um número representado em complemento de 2 pode assumir a seguinte faixa de valores:

$$(-2^{(n-1)}) \leq X \leq (2^{(n-1)} - 1), \text{ onde } n \text{ é a quantidade de bits do número}$$

A faixa de representação neste caso é assimétrica:

- Para o caso de 8 bits (byte), a faixa é: $-128 \leq X \leq 127$
- Para 16 bits (Word), a faixa é: $-32768 \leq X \leq 32767$
- Para 32 bits (double word), a faixa é: $-2147483648 \leq X \leq 2147483647$

A principal vantagem é ter uma única representação para o número 0. Por exemplo, para representar o número zero em um registrador de 6 bits:

$$0 = 0\ 0\ 0\ 0\ 0\ 0$$

Primeiro passo -0 = 1 1 1 1 1 1

Segundo passo + 1

Resultado -0 = 4 0 0 0 0 0

O último transporte ou transbordo (4) é desprezado. Portanto, o 0 e o -0 tem uma mesma representação. O método de representação em complemento de dois é o mais utilizado para representar números negativos.

UNIDADE 10

Sistema de Memória

Objetivo: Reconhecer a hierarquia das memórias: organização e características.

Memória tem papel crítico no desempenho de um sistema computacional. Ela recebe dados e transfere para processador, e vice-versa. Como a cada ciclo de instrução em uma máquina Von Neumann requer que o processador obtenha o código da instrução e seus operandos, a velocidade de transferência de itens da memória para o processador é, de fato, crítica.

Devido a grande variedade de características desejadas e dos diferentes tipos de memória não é possível implementar um sistema de computação com uma única memória. Na realidade, há muitas memórias no computador, as quais se interligam de forma bem estruturada; constituindo um sistema em si, parte do sistema global e podendo ser denominada subsistema de memória.

As principais características tecnológicas que diferenciam os vários tipos de dispositivos de memória incluem: custo, tempo e modo de acesso e persistência do armazenamento.

Hierarquia de Memórias

Para um funcionamento correto e eficaz da manipulação das informações (instruções e dados de um programa) de e para a memória de um computador, verifica-se necessidade de diferentes tipos de memória. Para certas atividades, a transferência de informações é a mais rápida possível (Como por exemplo, execução das instruções no processador), em outros casos, a rapidez não é fator fundamental, mas sim a capacidade de armazenamento.

No sentido de atender as diferentes características necessárias a um sistema de armazenamento, com uma relação custo/benefício atraente, as diferentes tecnologias de fabricação e armazenamento de dados são utilizadas e organizadas de forma hierárquica.

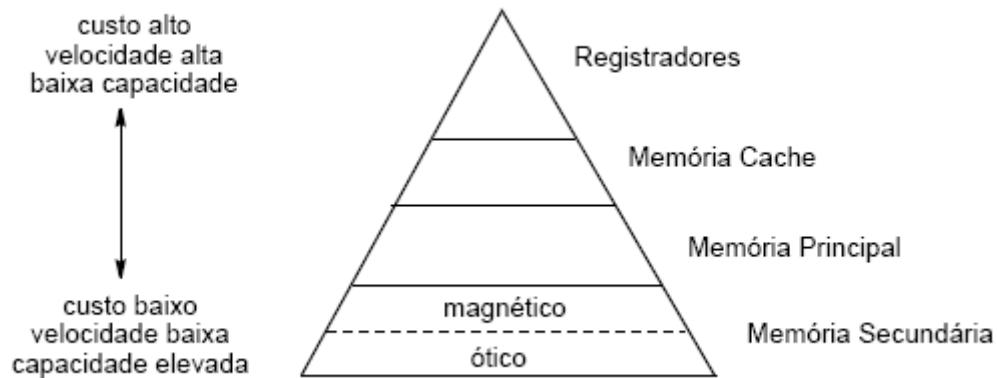


Figura 16 - Hierarquia de sistemas de memórias.

Os principais parâmetros de análise das características das memórias serão apresentados como segue:

- **Tempo de acesso:** indica quanto tempo à memória gasta no barramento de dados após uma determinada posição ter sido endereçada. É o período de tempo decorrido desde o instante em que foi iniciada a operação, até que a informação seja disponibilizada.
- **Tempo de ciclo de memória:** período de tempo decorrido entre duas operações sucessivas de acesso de leitura ou escrita na memória.
- **Capacidade:** é a quantidade de informação que pode ser armazenada na memória; a unidade de medida mais comum é o byte, embora também possam ser usadas outras unidades como células, setores, etc. O tamanho da memória (sua capacidade) indica

o valor numérico total de elementos de forma simplificada, através da inclusão de K (kilo), M (mega), ou T (tera).

- **Volatilidade:** memórias podem ser do tipo volátil ou não volátil. Uma memória não volátil é a que retém a informação armazenada quando a energia elétrica é desligada. Memória volátil é aquela que perde a informação armazenada, quando a energia elétrica é desligada.

Registradores e memória RAM são memórias voláteis. Memórias magnéticas, óticas e também as do tipo ROM, EPROM são memórias do tipo não volátil.

- **Tecnologia de fabricação:** ao longo do tempo diversas tecnologias vêm sendo desenvolvidas para fabricação de memórias. Entre as mais conhecidas temos as memórias de semicondutores, as memórias de meio magnético as memórias de meio óptico.
- **Temporariedade:** diz respeito ao tempo de permanência da informação em um dado tipo de memória.
- **Custo:** o custo de fabricação de uma memória é bastante variado em função de diversos fatores, entre os quais se pode mencionar, principalmente: tecnologia de fabricação, que resulta em um maior ou menor tempo de acesso, ciclo de memória, etc. Uma boa unidade de medida de custo preço por byte armazenado, em vez do custo total da memória em si. Isto porque, devido às diferentes capacidades de armazenamento, seria irreal considerar, para comparação, custo pelo preço da memória em si.



Atividades

Atenção! Não dê continuidade aos seus estudos sem antes acessar sua SALA DE AULA e fazer a Atividade 1 através do “link” ATIVIDADES.



UNIDADE 11

Tipos de Memórias I

Objetivo: Classificar as Memórias quanto à leitura e escrita.

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

- **R/W - Read and Write** (memória de leitura e escrita), comumente chamada de RAM (*Random Access Memory* ou memória de acesso aleatório).

Esta memória permite operações de escrita e leitura pelo usuário e pelos programas. Seu tempo de acesso é da ordem de poucos nano segundos (ns) e 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 e não volátil, portanto também é uma RAM. É utilizada geralmente por fabricantes para gravar programas que não se deseja permitir que o usuário possa alterar ou apagar acidentalmente (por ex: a BIOS - *Basic Input Output System* e microprogramas de memórias de controle).

Quando se liga uma máquina, é da ROM que vem os programas que são carregados e processados no "boot" (na inicialização o hardware aponta automaticamente para o primeiro endereço da ROM). Desta forma, parte do espaço de endereçamento da MP é ocupada pela ROM. A ROM é mais lenta que a R/W e é barata, porém o processo produtivo depende de ser programada por máscara ("*mask programmed*") em fábrica

e devido ao alto custo da máscara somente se torna econômica em grandes quantidades.

- **PROM** - *Programmable Read Only Memory* ou memória apenas de leitura, programável. Esta memória é uma ROM programável (em condições e com máquinas adequadas, chamadas queimadores de PROM) e geralmente é comprada "virgem" (sem nada gravado), sendo 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 (com queimadores de PROM) e apagável (com máquinas adequadas, à base de raio ultravioleta). Esta memória é uma PROM apagável. Tem utilização semelhante à da PROM, para testar programas no lugar da ROM, ou sempre que se queira produzir ROM em quantidades pequenas, com a vantagem de poder ser apagada e reutilizada.
- **EEPROM** (ou E2PROM) - *Electrically Erasable Programmable Read Only Memory* ou memória apenas de leitura, programável e eletronicamente alterável. Também chamada EAROM (*Electrically Alterable ROM*). Esta memória é uma EPROM apagável por processo eletrônico, sob controle da UCP, com equipamento e programas adequados. É mais cara e, geralmente, utilizada em dispositivos nos quais se deseja permitir a alteração, via modem; 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* (p.ex., a reconfiguração de teclado ou de modem, programação de um terminal, etc.).

UNIDADE 12

Tipos de Memórias II – Memória Principal

Objetivo: Conhecer o funcionamento e as características da Memória Principal do computador.

Memória Principal é a parte do computador onde programas e dados são armazenados para processamento. A informação permanece na Memória Principal apenas enquanto for necessário para seu emprego pela UCP, sendo então a área da MP liberada para ser posteriormente utilizada por outra informação. Quem controla a utilização da Memória Principal é o Sistema Operacional.

Uma das principais características definidas no projeto de arquitetura de Von Neuman consistia no fato de ser uma máquina “de programa armazenado”. O fato das instruções, uma após a outra, poderem ser imediatamente acessadas pela UCP é que garante o automotismo do sistema e aumenta a velocidade de execução dos programas.

A memória precisa ter uma organização que permita ao computador guardar e recuperar informações quando necessário. Não teria nenhum sentido armazenar informações que não fosse possível recuperar depois. Portanto, não basta transferir informações para a memória. É preciso ter como 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, este mecanismo é conhecido como **Célula**.

Célula é a unidade de armazenamento do computador. A Memória Principal é organizada em células que é a menor unidade da memória que pode ser endereçada (não é possível buscar uma "parte" da célula) e tem um tamanho fixo (varia conforme a arquitetura da máquina). As memórias são compostas de um determinado número de células ou posições e cada célula é composta de um determinado número de bits.

Cada célula é identificada por um endereço único, pela qual é referenciada pelo sistema e pelos programas. As células são numeradas sequencialmente, uma a uma, de 0 a ($N-1$), chamado o **endereço da célula**.

A estrutura da Memória Principal é um problema do projeto de hardware:

- mais endereços com células menores ou
- menos endereços com células maiores?

O tamanho mais comum de célula era 8 bits (1 byte); hoje são comuns células contendo vários bytes.

| Endereço da célula | Célula de dados (8 bits) |
|--------------------|--------------------------|
| 0 | 0 1 1 0 0 0 1 0 |
| 1 | 1 0 1 1 1 0 0 1 |
| 2 | 0 1 1 0 0 0 1 0 |
| 3 | 0 1 1 0 0 0 1 0 |
| 4 | 1 0 1 1 1 0 0 1 |
| 5 | 0 1 1 0 0 0 1 0 |
| ... | ... |
| N | N |

Capacidade da Memória Principal

A capacidade da MP em bits é igual ao produto do nº de células pelo total de bits por célula.

$$T = N \times M$$

T = capacidade da memória em bits

N = nº de endereços

M = nº de bits de cada célula

Para encontrar a capacidade em bytes, bastaria encontrar a capacidade em bits e depois dividir por 8 (cada byte contém 8 bits) ou então converter o tamanho da célula para bytes e depois multiplicar pelo número de células.

O último endereço na memória é o endereço N-1 (os endereços começam em zero e vão até N-1).

UNIDADE 13

Tipos de Memórias III - Memória Cache

Objetivo: Conhecer o funcionamento da memória cache e a sua relação com a memória principal.

Cache é uma memória especial de alta velocidade projetada para acelerar o processamento das instruções dadas à Memória Principal pela CPU. A CPU pode acessar instruções e dados localizados na memória cache muito mais rapidamente do que na memória principal.

Para a CPU processar uma instrução é necessário que esta seja carregada no registrador de instrução e seus operandos sejam trazidos da memória para os demais registradores. Enquanto estes acessos à Memória Principal (MP) são feitos, a CPU tem que ficar esperando que os dados sejam disponibilizados. Para que a MP não seja um gargalo que limite a velocidade de processamento da CPU; foi criado um esquema de utilização de memória Cache, que é muito mais rápida que a principal, com tempos de acesso compatíveis com a velocidade da CPU, e que serve para guardar as últimas posições de memória acessadas.



Figura 17 - Memória cache e memória principal.

Como a experiência nos mostra que a reutilização dos mesmos dados ou instruções é muito intensa, este processo aumenta o desempenho do sistema de computação.

O chamado princípio da localidade está por trás da maneira como os programas operam. Este princípio estabelece que os programas acessem uma parte relativamente pequena do seu espaço de endereçamento em um instante qualquer. Existem dois diferentes tipos de localidade:

- Localidade temporal: se um item é referenciado, ele tende a ser referenciado novamente dentro de um curto espaço de tempo.
- Localidade espacial: se um item é referenciado, itens cujos endereços sejam próximos a eles tendem a ser logo referenciados.

No exemplo abaixo, certo programa é constituído de um grupo de instruções iniciais realizadas em sequência (Sequência1) dois loops e o resto do código (Sequência 2). Assim, quando o programa é iniciado as instruções da sequência1 são consecutivamente executadas uma após a outra até que a região de código constituída pelo loop 1 seja executada. A localidade espacial aparece explicitamente na execução de cada uma das sequências. A localidade temporal, por sua vez, pode ser percebida quando da execução de qualquer um dos loops, cuja sequência de instruções nele contidas será executada de tempos em tempos (a cada repetição do laço).

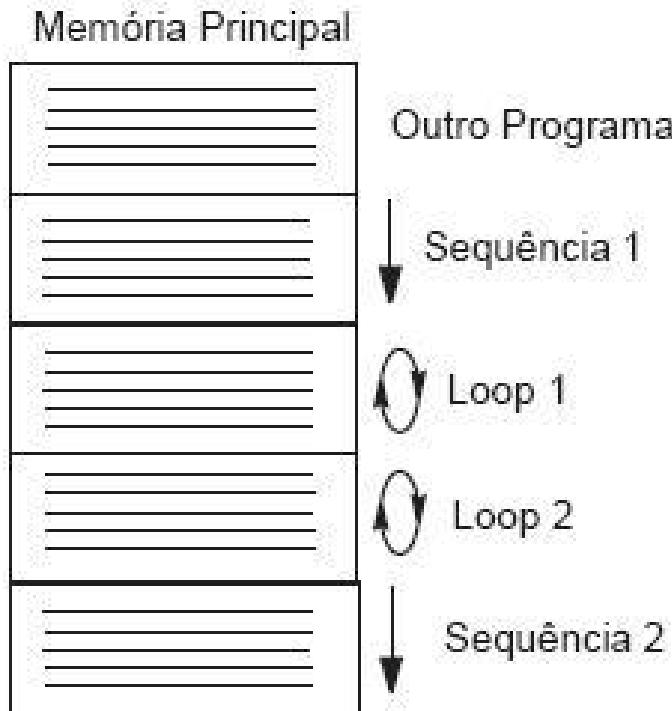


Figura 18 – Exemplo de Princípio de localidade.

Exemplo dos princípios de localidade espacial e temporal

Quando o processador solicita um item de dado gerando uma referência para seu endereço, que pode ser físico ou virtual, o gerenciador de memória requisita este item do cache. Duas situações podem ocorrer:

1. cache hit - Acerto: o item está presente no cache, é retornado para o processador praticamente sem período de latência;
2. cache miss - Falta ou Falha: o item não está presente no cache e o processador deve aguardar item ser buscado da memória principal.

A taxa de acertos, ou razão de acertos, corresponde à fração dos acessos à memória encontrados no nível superior, e com frequência é usada como medida de performance da

hierarquia de memória. A taxa de faltas é a fração de acessos à memória não encontrados no nível superior (na cache).

Define-se como tempo de acerto o tempo necessário para acessar o nível superior da hierarquia (cache), que inclui o tempo necessário para determinar se o acesso a informação vai gerar um acerto ou uma falta.

A penalidade por falta é o tempo necessário para substituir um dos blocos do nível superior, pelo bloco do nível inferior contendo a informação desejada, mais o tempo para enviar a informação ao processador. Em razão do nível superior (cache) ser menor e de ser constituído de memórias mais rápidas, o tempo de acerto é muito menor que o tempo necessário ao nível mais baixo da hierarquia (MP). Justamente este tempo de acesso é o maior tempo dentro da penalidade por falta.

Para que haja aumento de desempenho, com a inclusão da memória cache, é necessário que exista mais acertos do que faltas. Isto implica no dimensionamento adequado da memória cache como um todo e principalmente do tamanho de cada bloco.



Dica

Vídeo

Assista aos vídeos “Como um computador funciona (parte 1 e 2)” através do link “Estudo Complementar”.

Este vídeo é uma animação com humor sobre o funcionamento de um computador.

Vale à pena assistir!



UNIDADE 14

Registradores

Objetivo: Conhecer o funcionamento dos registradores especiais do processador, responsáveis por organizarem o controle da execução das instruções.

Dentro da UCP existe um conjunto de registradores que tem duas funções:

- Possibilitar ao programador de linguagem de montagem ou de máquina minimizar referências à memória. São os registradores visíveis para o usuário.
- Controlar as operações da CPU. São usados por programas privilegiados do sistema operacional e pela unidade de controle, para controlar a execução de programas. São os Registradores de controle e estado.

Registradores visíveis para o usuário

Registradores visíveis ao usuário são aqueles que armazenam as informações que estão sendo processadas em um determinado instante e podem ser referenciados pela linguagem de máquina que a CPU executa. Podem ser classificados nas seguintes categorias:

- Registradores de Uso Geral
- Registradores de Dados
- Registradores de Endereços

A quantidade e função de registradores existentes variam de acordo com o modelo do processador e de acordo com o fabricante.

Os registradores de uso geral podem ser usados para uma variedade de funções, em alguns casos, podem ser usados para endereçamentos. Em outros, existe uma separação clara ou parcial entre registradores de dados e registradores de endereços.

Registradores de dados podem ser usados apenas para conter dados e não podem ser usados no cálculo de endereçamentos dos operandos.

Registradores de endereços podem ser empregados, até certo ponto, como registradores de: Propósito Geral ou podem ser dedicados para um determinado modo de endereçamentos. Alguns exemplos são:

- **Registradores de Segmento:** tem como finalidade indicar ao processador o endereço de um determinado segmento.
- **Registradores de índices:** são usados para endereçamento indexado, possivelmente com autoindexação.
- **Apontador de topo de pilha:** se houver endereçamento de operandos na pilha visível para o usuário, então a pilha será alocada na memória e existirá um registrador dedicado que aponta para o topo da pilha. Com isso, as operações de empilhar e desempilhar não requer um operando explícito.

Uma questão importante de projeto do conjunto de registradores é decidir se serão de uso geral ou se terão uso específico. Com o uso de registradores especializados, o tipo de registrador referenciado como operando de uma instrução geralmente é implícito, sendo determinado pelo código de operação. Com isso o campo do operando apenas identifica um registrador de um conjunto de registradores especializados, com economia de alguns bits de instrução. O outro lado desta questão é que a especialização limita a flexibilidade de programação.

Outra questão importante é a quantidade de registradores a serem disponibilizados, seja para uso geral, seja para registradores de dados e de endereços. Um número muito pequeno de registradores resulta em mais referências à memória. Um número muito grande, por outro lado, não reduz significativamente o número de referências à memória. O número adequado parece estar entre 8 e 32 registradores (STALLINGS, 2002).

O tamanho do registrador deve ser suficiente para acomodar o maior endereço usando no sistema, no caso de registrador de endereços. No caso de registrador de dados, deve ser capaz de conter a maioria dos tipos de dados.

Registradores de Controle e Estado

São registradores empregados para controlar a operação da CPU e são usados para a transferência de dados entre a CPU e a memória. Quatro registradores são essenciais para a execução das instruções:

- ***Contador de Instrução (CI)***: tem a finalidade de indicar a próxima instrução a ser executada, sendo automaticamente atualizado pelo processador após a busca da instrução. A instrução buscada é carregada no RI.
- ***Registrador de Instrução (RI)***: contém a última instrução buscada. É onde o código de operação e as referências a operando são analisadas.
- ***Registrador de Endereçamento à Memória (MAR)***: contém o endereço de uma posição de memória. A troca de dados com a memória é feita usando o MAR e o MBR. Em um sistema com barramento, o MAR é conectado diretamente ao barramento de endereço.
- ***Registrador de armazenamento temporário de dados (MBR)***: contém uma palavra de dados a ser escrita na memória ou a palavra lida mais recentemente.

UNIDADE 15

Representação e Ciclo de Instrução

Objetivo: Identificar a Representação da Instrução e a análise do Ciclo da Instrução.

Representação da Instrução

Quem executa um programa é o hardware e o que ele espera encontrar é um programa em linguagem de máquina (uma sequência de instruções de máquina em código binário). Um programa em linguagem de alto nível não pode ser executado diretamente pelo *hardware*, pois ele tem que ser transformado (traduzido) para linguagem de máquina, antes de ser carregada para memória, para que o hardware possa executá-lo. A linguagem de máquina é composta de códigos binários, representando instruções, endereços, dados e está totalmente vinculada ao conjunto (*set*) de instruções da máquina.

Funcionalmente, as operações do computador são:

- Matemáticas (aritméticas, lógicas, de complemento, de deslocamento.)
- Movimentação de dados (entre memória e registrador)
- Entrada-saída (leitura e escrita em dispositivos externos - dispositivos de Entrada / Saída)
- Controle (desvio da sequência de execução, parar, entre outros)

Cada uma das instruções tem um código binário associado, que é o Código da Operação.

Formato das Instruções

A instrução é composta de Código de Operação e zero, um ou mais Operandos:



- **Código de Operação ou OPCODE** - identifica a operação a ser realizada pelo processador. É o campo da instrução cujo valor binário identifica a operação a ser realizada. Cada instrução deverá ter um código único que a identifique.
- **Operando(s)** - é o campo 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 onde está contido o dado que será manipulado. Um operando pode também indicar um Registrador (que conterá o dado propriamente dito ou um endereço de memória onde está armazenado o dado). Os operandos fornecem os dados da instrução.

Obs.: Existem instruções que não têm operando. Ex.: Instrução HALT (PARE).

Há diversos formatos de instruções, com características particulares, vantagens e desvantagens.

O conjunto de instruções de uma máquina pode ser constituído por instruções de diversos formatos. Esta flexibilidade permite a escolha da instrução adequada para aplicação em cada caso.

Conjunto de instruções que pode ser analisado sob alguns aspectos, por exemplo:

- quantidade de instruções
- quantidade de operandos
- modo de endereçamento (é a forma de sinalizar a localização de um dado, conhecido como “Modos de Endereçamento”).

Ciclo de Instrução

O processamento necessário para a execução de uma instrução é chamado de ciclo de instrução. De forma simplificada pode-se entender o ciclo de instrução como composto de dois passos básicos: ciclo de busca e o ciclo de execução. De forma mais geral, o ciclo de instrução é formado pelos seguintes passos:

- Busca da próxima instrução no endereço da memória principal apontado pelo CI e armazenamento da mesma no RI.
- Atualização do CI, fazendo-o apontar para a instrução seguinte ($CI = CI + 1$).
- Determinação do tipo de instrução armazenada no RI.
- Se a instrução precisa de operandos armazenados na memória principal, os seus endereços devem ser determinados.
- Caso necessário, busca os operandos na memória principal.
- Execução da instrução.
- Retorno ao primeiro passo, para iniciar a execução da instrução seguinte.

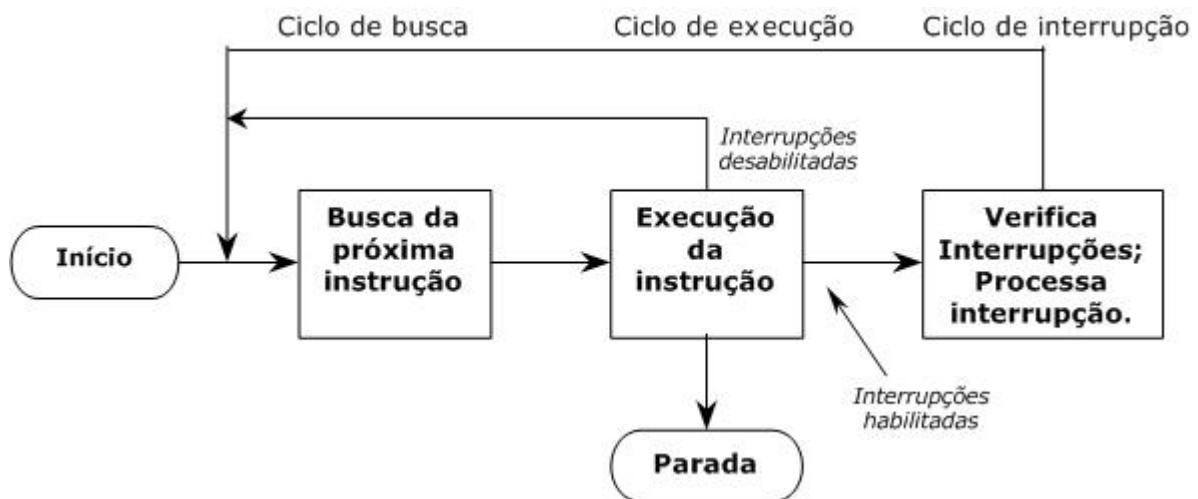


Figura 19 - Ciclo de instrução básico.

UNIDADE 16

Modos de Endereçamento

Objetivo: Conhecer os tipos de modos de endereçamento.

Na unidade anterior foi descrito o formato básico de instruções de máquina e o ciclo de execução de cada instrução, concluindo que:

- 1) O endereçamento de uma instrução é sempre realizado através do valor armazenado no Contador de Instrução (CI). Todo o ciclo de instrução é iniciado pela transferência da instrução para o Registrador de Instrução (RI);
- 2) Toda instrução consiste em uma ordem codificada (código de operação), para o processador executar uma operação qualquer sobre os dados;
- 3) A localização do(s) dado(s) pode estar explicitamente indicada na própria instrução por um ou mais conjuntos de bits, denominados campo do operando.

Todos os exemplos apresentados, até esse ponto, definiram o campo operando da instrução como sendo um endereço da MP onde está localizado o dado; no entanto, essa não é a única maneira de indicar a localização dos dados, havendo outros modos de endereçamento.

Dentre os diversos modos de endereçamento desenvolvidos para processadores, os principais são:

Modo Imediato

O método mais simples e rápido de obter um dado é indicar seu próprio valor no campo operando da instrução, em vez de buscá-lo na memória. A vantagem desse método reside

no curto tempo de execução da instrução, pois não gasta ciclo de memória para sua execução, exceto o único requerido para a sua busca.

Desvantagens

- O tamanho do dado fica limitado ao número de bits do operando (campo operando da instrução). A limitação de tamanho do campo operando reduz o valor máximo do dado que pode ser armazenado.
- Este modo de endereçamento não permite flexibilidade para alterar dados que variam a cada execução do programa, portanto não é adequado para variáveis repetidamente operadas com diferentes valores a cada execução do programa.

Modo Direto

Nesse método, o valor binário contido no campo operando da instrução indica o endereço de memória onde se localiza o dado.

Vantagens

- É aplicado em mais situações que o modo imediato;
- Requer apenas uma referência à memória para busca do dado (além de uma para a busca da instrução), sendo mais rápido que o modo indireto.

Desvantagens

- Limitação do endereço da MP que pode ser indicado pelo tamanho do campo operando.
- É mais lento que o modo imediato.

Modo Indireto

Nesse método, o valor binário contido do campo operando representa o endereço de uma célula, mas o conteúdo da referida célula não é o valor de um dado (como no modo direto), é outro endereço de memória, cujo conteúdo é o valor do dado. A grande desvantagem desse método é, obviamente, a maior quantidade de ciclos de memória requerida para completar o ciclo de instrução, pois para se acessar um dado, no modo indireto, é necessário efetuar dois acessos à memória. Um para buscar o endereço do dado e outro para, efetivamente, buscar o dado.

Vantagens

- Permite implementar estruturas de organização de dados mais complexas, mais sofisticadas.
- Elimina a limitação de células endereçáveis.

Modo de endereçamento por registrador

Esse método tem característica semelhante aos modos direto e indireto, exceto que a célula de memória referenciada na instrução é substituída por um dos registradores do processador. Com isso, o endereçamento mencionado na instrução passa a ser o de um registrador, e não mais de uma célula da MP.

A primeira vantagem, logo observada, consiste no menor número de bits necessários para endereçar os registradores, visto que estes existem em muito menor quantidade que as células de memória. Há duas maneiras de empregar o modo de endereçamento por registrador:

Modo por registrador direto

O registrador endereçado na instrução contém o dado a ser manipulado.

Modo por registrador indireto

O registrador referenciado armazena o endereço de uma célula de memória onde se encontra o dado.

UNIDADE 17

Pipelining

Objetivo: Entender o que significa Pipelining.



MEDIATECA

VÍDEO

Atenção! Antes de dar continuidade aos seus estudos. Vá ao ambiente CAMPUS ON-LINE e assista ao vídeo referente à UNIDADE 17.



Pipelining é uma técnica desenvolvida para melhorar o desempenho de processadores. O *pipelining* permite que um processador sobreponha a execução de diversas instruções de modo que mais instruções possam ser executadas no mesmo período de tempo.

Uma *pipelining* de instruções é semelhante a uma linha de montagem, de uma indústria. Na linha de montagem pode-se começar a fazer o segundo produto antes do primeiro estar concluído. De forma análoga, em uma *pipelining* de instruções, novas entradas são aceitas em uma extremidade, antes que entradas previamente apareçam como saídas na outra extremidade.

Por exemplo: Em uma lavanderia, onde cada sacola de roupa a ser limpa é equivalente a uma instrução, cada etapa do processo de limpar a roupa (lavar, secar e dobrar) é equivalente a um ciclo de processamento; e a quantidade de sacolas de roupa limpas num determinado

período de tempo era equivalente ao *throughput* do processador. Em uma lavanderia, sem *pipeline*, embora tenha uma máquina de lavar, uma de secar e uma pessoa para dobrar a roupa; somente uma sacola de roupa é limpa de cada vez. Enquanto não lavar, secar e dobrar as roupas desta sacola, nenhuma outra sacola de roupa pode ser limpa. Nada ocorre simultaneamente.

Se esta lavanderia utilizasse um processo em *pipeline*, logo que as roupas da primeira sacola estivessem lavadas, elas passariam para a secadora liberando a máquina de lavar para uma nova sacola de roupas. Assim que as roupas da primeira sacola estivessem secas, então elas passariam a ser dobradas, as da segunda sacola que estavam sendo lavadas passariam para o secador e uma nova sacola poderia ser posta na máquina de lavar.



Figura 20 - Fluxo de instruções em um processador com pipeline

Na execução em pipeline, cada tarefa individualmente ainda requer “ n ” segundos e o tempo total para a execução de **uma** operação em *pipeline* é, em geral, ligeiramente maior que o tempo para executar a mesma operação monolicamente (sem *pipeline*).

Um dos *overheads* associados à operação de um *pipeline* é decorrente da necessidade de se transferir dados entre os estágios. Há duas estratégias básicas para controlar a transferência de dados entre os estágios de um *pipeline*: o método assíncrono e o método síncrono.

Método Assíncrono

No método assíncrono, os estágios do *pipeline* comunicam-se através de sinais de *handshaking*, indicando a disponibilidade de dados do estágio corrente para o próximo estágio (RDY); indicando a liberação do estágio corrente para o estágio anterior (ACK).

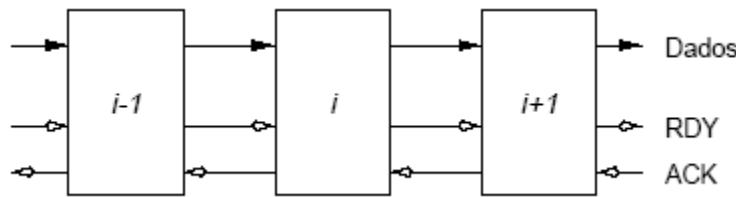


Figura 21 – Método assíncrono

Método Síncrono

No método síncrono, os estágios do *pipeline* são interconectados por *latches* (registradores cujo objetivo é armazenar dados) que armazenam os dados intermediários durante a transferência entre estágios, que é controlada por um sinal de relógio. Neste caso, o estágio com operação mais lenta determina a taxa de operação do *pipeline*.

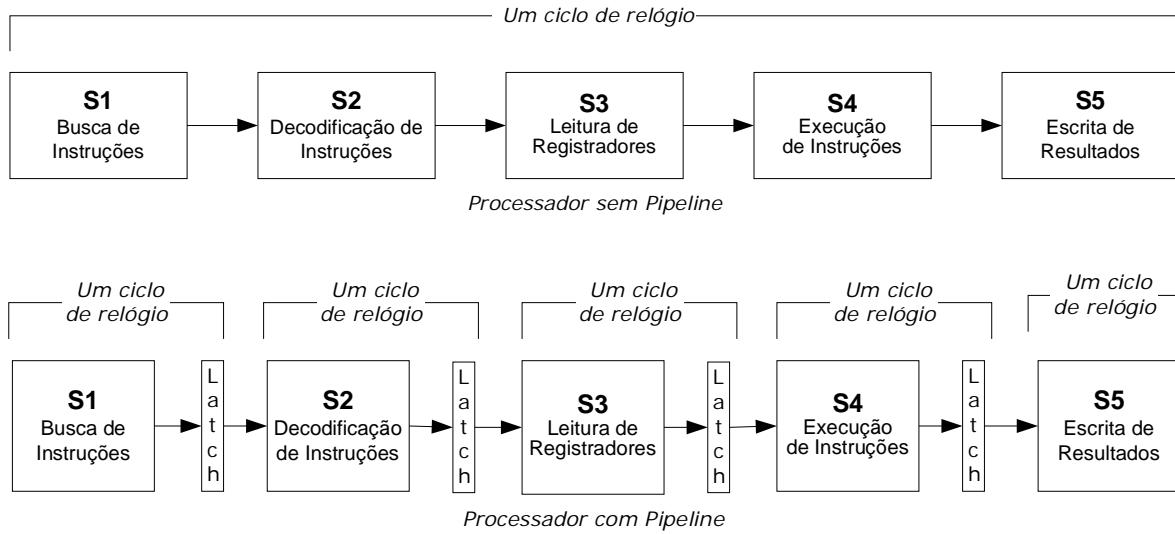


Figura 22 – Método síncrono

O método assíncrono é o que permite maior velocidade de operação do *pipeline*. Entretanto, o método síncrono é o mais adotado devido à sua simplicidade de projeto e operação.

UNIDADE 18

Organização de uma Interface de E/S (Comunicação entre a Memória e UCP – Barramentos)

Objetivo: Conhecer a organização da interface de E/S.

A função primordial de uma interface de E/S é realizar controles adequados sobre os dispositivos periféricos, de modo que o processador possa se relacionar com estes periféricos de forma transparente. A organização de uma interface de E/S pode ser entendida em duas partes, conforme mostrado na figura a seguir.

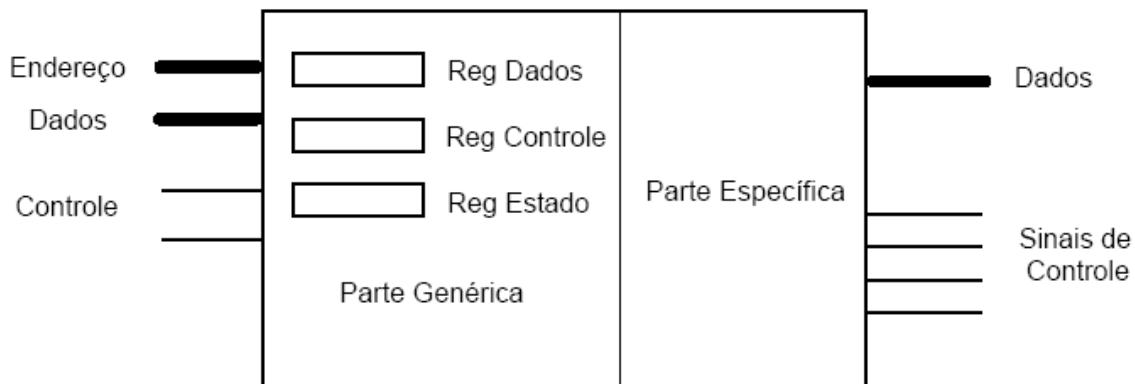


Figura 23 - Organização típica de uma interface de E/S.

Qualquer que seja o tipo de interface, a parte genérica é muito semelhante. Nesta parte são encontrados, usualmente, alguns registradores que variam em quantidade de acordo com o tipo de periférico acoplado. Pelo menos um **registraror de dados**, um **registraror de**

controle e um **registraror de estado** estão inclusos, sendo o acesso a estes registradores feitos através de endereços de E/S diferentes.

Para operação de saída, o processador escreve um dado no registrador de dados e a interface procede o envio ao periférico. Na operação de entrada, a interface armazena o dado recebido do periférico no registrador de dados, que por sua vez é lido pelo processador.

Na operação de saída, o processador envia comandos através do registrador de controle, para que a interface possa interpretar e executar a operação solicitada; esta operação pode ser interna à interface ou sobre o periférico a ela conectado.

O registrador de estado é usado para veicular informações gerais sobre uma operação de E/S. Tipicamente, este registrador possui *bits* para indicar o término de uma operação e para indicar condições de erro que eventualmente possam acontecer durante a operação.

A parte específica tem interação direta com o periférico, sendo desta forma muito diferente de um periférico para outro. Apesar das diferenças, geralmente na parte específica temos dois conjuntos de sinais, um é a via através da qual os dados são transferidos; outro são os sinais de controle com o periférico.

Barramento de E/S

Um barramento, ou bus, nada mais é do que um caminho comum pelo qual os dados trafegam dentro do computador. Este caminho é usado para comunicações e pode ser estabelecido entre dois ou mais elementos do computador.

O tamanho de um barramento é importante, pois ele determina quantos dados podem ser transmitidos em uma única vez. Por exemplo, um barramento de 16 bits pode transmitir 16 bits de dado, e um barramento de 32 bits pode transmitir 32 bits de dados a cada vez.

A grande vantagem do uso do barramento de E/S é ser um padrão de comunicação entre o dispositivo e o processador, ou seja, uma interface. Isso faz com que sistemas que utilizem barramentos de E/S sejam muito flexíveis, em oposição a conexões diretas entre o

processador e cada dispositivo de E/S, permitindo que um sistema suporte muitos dispositivos de E/S diferentes; dependendo das necessidades dos seus usuários, e permitindo que estes mudem os dispositivos de E/S, que estão conectados em seus sistemas, à medida que as suas necessidades mudam. A principal desvantagem dos barramentos em geral (e também dos barramentos de E/S) é que tem uma largura de banda fixa que precisa ser compartilhada por todos os dispositivos que estão sobre ele.

O barramento pode ser dividido em três conjuntos:

1. **Barramento de endereços:** por onde trafegam os endereços de memória ou dispositivos de E/S. Este barramento é **unidirecional**: somente o processador fornece endereços. Tem como função conduzir um endereço fornecido pelo processador para que este endereço possa ser acessado.
2. **Barramento de dados:** por onde trafegam os dados do processador para memória e dispositivos de E/S e vice-versa. Portanto, tal barramento é **bidirecional**: o processador tanto envia como recebe dados.
3. **Barramento de controle:** bidirecional. Por tal barramento trafegam os sinais de controle do microprocessador para memória e dispositivos, bem como da memória e dispositivos para o processador. Tem como função indicar que o processador quer executar determinada ação (como por exemplo, ler ou gravar de um endereço de memória) ou receber um sinal indicando determinado estado (por exemplo, dado já disponível no barramento de dados).

UNIDADE 19

Entrada/Saída

Objetivo: Identificar o controle das conexões entre o processador, a memória e os outros dispositivos.

Em sistemas como computadores pessoais e estações de trabalho as *interfaces de E/S* estão ligadas ao processador através de barramentos de endereço, dados e controle, de maneira semelhante à conexão entre memória principal e processador. A organização típica de um computador incluindo o subsistema de E/S é mostrada na figura a seguir.

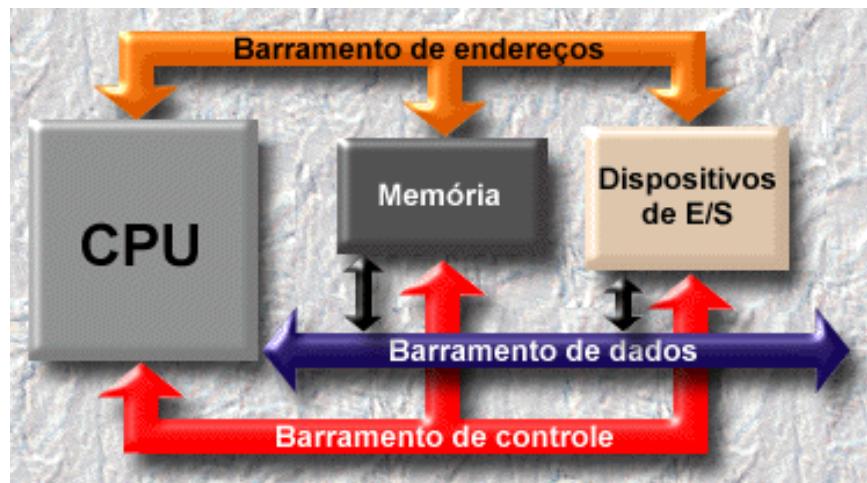


Figura 24 - Arquitetura de um computador incluindo o subsistema de E/S.

O que todos os dispositivos de entrada têm em comum é que eles precisam codificar (converter) a informação de algum tipo em dados que podem ser processados pelo sistema digital do computador. Dispositivos de saída por outro lado, decodificam os dados em

informação que é entendida pelo usuário do computador. Neste sentido, um sistema de computadores digital é um exemplo de um sistema de processamento de dados.

O processador realiza acessos de leitura ou de escrita a uma *interface* de E/S. Em um acesso de leitura, o processador obtém um dado recebido do dispositivo periférico conectado à *interface*, ou então uma informação de estado sobre uma operação de E/S em andamento ou recém-completada. Em um acesso de escrita, o processador fornece à *interface* um dado que deve ser enviado ao dispositivo periférico, ou então o código de um comando que inicia uma operação de E/S ou uma operação de controle sobre o dispositivo periférico.

Nos acessos às *interfaces*, o processador executa ciclos de barramento semelhantes aos descritos no capítulo anterior. Cada *interface* de E/S é identificada por um endereço único. Em um acesso de leitura, o processador coloca o endereço da *interface* no barramento de endereço e ativa um sinal de leitura. Após certo intervalo de tempo, a *interface* coloca a informação desejada no barramento de dados. O processador finaliza o ciclo de barramento lendo a informação presente no barramento de dados e retirando o endereço e o sinal de controle.

Em um acesso de escrita, o processador coloca o endereço da *interface* e o dado nos respectivos barramentos, e ativa um sinal de escrita. A *interface* selecionada armazena a informação presente no barramento de dados. No final do ciclo de barramento, o processador retira o endereço e o dado e desativa o sinal de controle. Assim como nos ciclos de barramento com a memória, todos estes eventos são comandados pelo processador e ocorrem em sincronismo com o sinal de *clock*.

UNIDADE 20

Formas de Comunicação

Objetivo: Conhecer as diferentes formas de Comunicação da Informação, pelos dispositivos do computador.

As portas de comunicação de um microcomputador permitem a interligação física dele com os diversos periféricos como: impressoras, modem, mouse, scanners, etc.

Há duas maneiras básicas de comunicação de dados entre o computador e outros equipamentos. Temos a comunicação paralela e a comunicação serial.

Comunicação Serial

Na comunicação Serial, o byte é enviado por apenas uma via ou fio. Para que isso seja possível, o byte é desmembrado em bits e cada um é enviado separadamente, um após o outro. No local da recepção, os bits são "montados" novamente, recompondo o byte. Os sinais de controle são enviados separadamente.

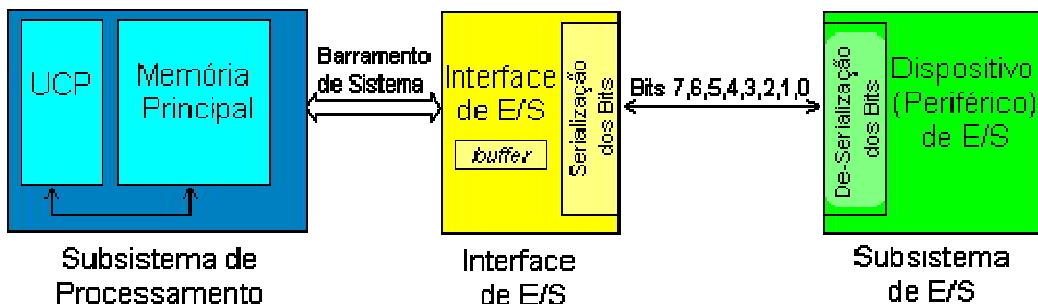


Figura 25 - Esquema de comunicação serial.

A porta serial contém um chip UART (Universal Asyncronous Reciver/Transmiter) e alguns componentes de apoio. Como o nome indica, este chip recebe dados provenientes do barramento do PC e os traduz para o formato utilizado nas transmissões seriais.

O chip também executa o procedimento inverso: receber uma string de dados, remover os caracteres de enquadramento e transferir os bytes de dados para o PC.

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 (comunicação via linha telefônica usando *modem*).

A transmissão serial tem recebido aperfeiçoamentos importantes (seja de protocolo, de *interface* e de meio de transmissão) que vem permitindo 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 hoje ligar até 128 dispositivos a taxas muito altas (centenas de kbps).

Comunicação Paralela

Comunicação Paralela é aquela em que os bits, que compõem um byte ou palavra de dados, são enviados ou recebidos simultaneamente bem como os sinais de controle de comunicação. Para que isso seja possível, faz-se necessário um meio físico (fio) para cada informação, seja ele de dado ou de controle.

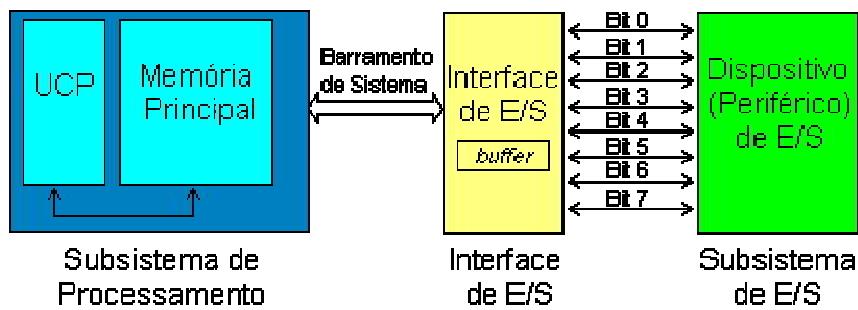


Figura 26 - Esquema de comunicação paralela.

O processo de transferência em paralelo envolve um controle sofisticado e é razoavelmente complexo, o que o torna mais caro. A propagação dos sinais no meio físico é um dos problemas mais importantes neste tipo de comunicação. A propagação dos sinais deve ser tal que todos os bits cheguem juntos à outra extremidade do cabo. Cada condutor que compõe o cabo tem pequenas diferenças físicas, o que pode tornar a velocidade de propagação dos sinais ligeiramente diferente para cada condutor. Para cabos mais longos, um bit pode chegar a um determinado condutor pode chegar mais adiantado ou atrasado em relação aos demais, tornando a informação irreconhecível. Este fenômeno chama-se *skew*. Em face deste problema, há limites para o comprimento do cabo que interliga um dispositivo ao computador, quando se usa o modo paralelo.

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 que 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 a até um máximo de 1,5 metros. O conector de uma interface paralela é do tipo DB-25 e contém 25 pinos.

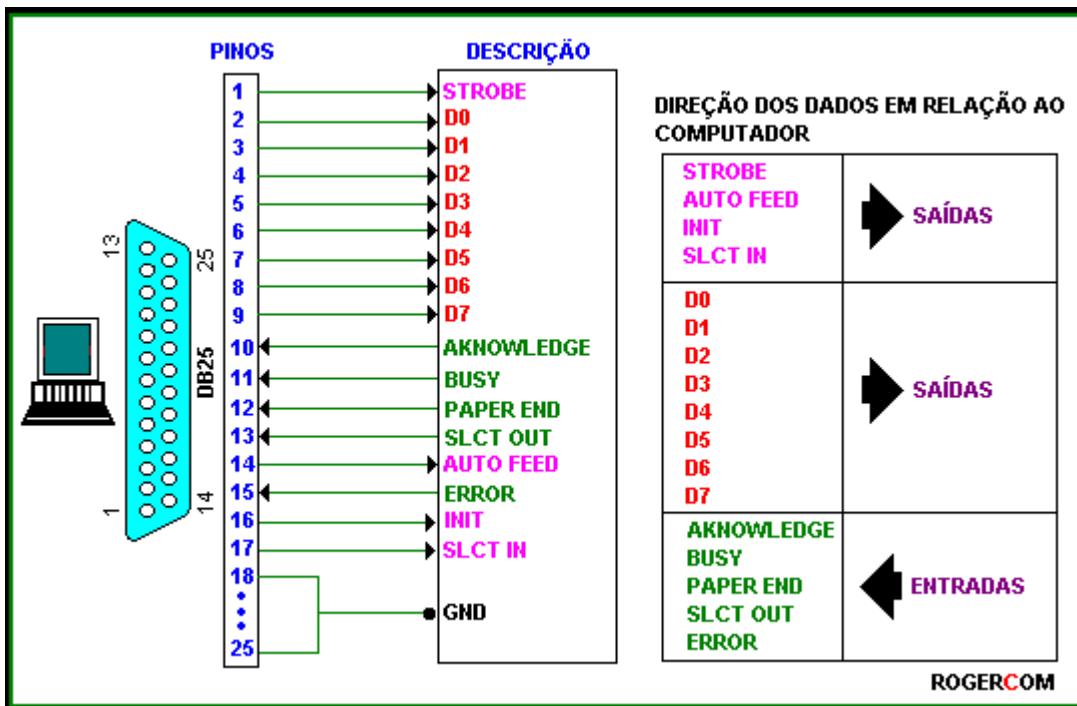


Figura 27 - conector de uma interface paralela do tipo DB-25



Atividades

Antes de dar continuidade aos seus estudos é fundamental que você acesse sua SALA DE AULA e faça a Atividade 2 no “link” ATIVIDADES.



UNIDADE 21

Técnicas de Transferência de Dados I

Objetivo Saber utilizar as E/S com Polling e Interrupção.

Em geral, uma operação de E/S envolve a transferência de dados entre a memória e a interface de E/S. Existem basicamente três técnicas para realizar a transferência de dados: **polling, interrupção e acesso direto à memória**. Descrição a seguir:

E/S com Polling

Na E/S com *polling*, o processador controla toda a transferência de dados entre a memória e a interface de E/S. Para entender como é o procedimento desta técnica, considere o exemplo de uma operação de escrita em um setor de disco. Suponha que a interface controladora de disco é semelhante àquela mostrada na abaixo. Normalmente, o registrador de estado possui um *bit*, chamado ***done bit***, que é desativado quando um dado é escrito no registrador de dados, sendo ativado quando este dado é escrito no setor do disco. O diagrama abaixo mostra como acontece à escrita de um setor de disco usando-se E/S com *polling*.

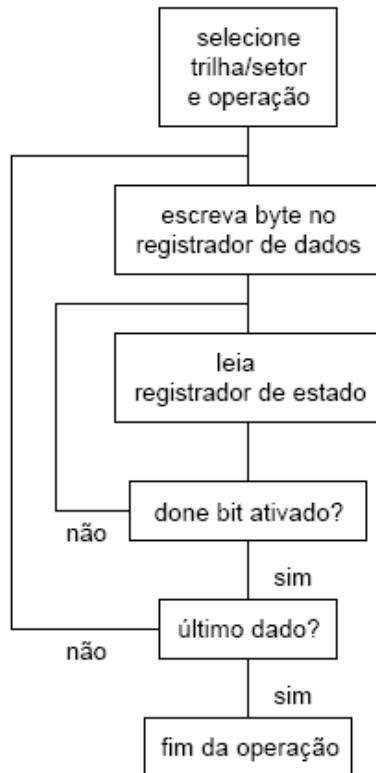


Figura 28 - Exemplo de E/S com polling.

Após escrever um dado no registrador de dados, o processador lê o registrador de estado e testa o *done bit*, para verificar se o mesmo já foi escrito no setor do disco. Este teste do *bit* de estado é chamado *polling*. O processador continua realizando o *polling* até encontrar o *done bit* ativado, o que indica que o dado já foi escrito no setor do disco. Quando isto acontece, e se ainda existe algum dado a ser enviado, o processador escreve o novo dado no registrador de dados e reinicia o *polling*. Este ciclo é repetido até que todos os dados tenham sido escritos no setor do disco.

A principal vantagem da E/S com *polling* é a sua simplicidade. No entanto, esta técnica possui a desvantagem de que o processador fica dedicado à operação de E/S. Isto pode ser extremamente ineficiente, sob o ponto de vista da utilização do processador. Considere uma operação de envio de um bloco de caracteres para uma impressora. O tempo de impressão de um caracter é infinitamente maior que o tempo de execução de uma instrução. Manter o

processador em *polling* durante o tempo de impressão de cada caractere é um desperdício, já que durante este intervalo de tempo o processador poderia executar alguns milhões de instruções de outro programa. Devido ao fato que o processador fica dedicado à operação de E/S até o seu término, o uso da técnica de E/S com *polling* é restrito apenas a sistemas onde apenas um programa pode se encontrar em execução a cada instante.

E/S com Interrupção

Na E/S com *polling*, o processador fica dedicado à operação de E/S porque ele é o responsável por determinar quando um novo dado pode ser transferido entre a memória e a interface de E/S. O mesmo não acontece na E/S com interrupção. Nesta técnica, a *interface* é responsável por notificar o processador quando um novo dado pode ser transferido. Enquanto a E/S com *polling* é uma técnica puramente de *software*, a E/S com interrupção requer um suporte de *hardware*. A *interface* deve gerar um sinal de interrupção, através do qual ela notifica o processador quando uma operação de E/S foi concluída.

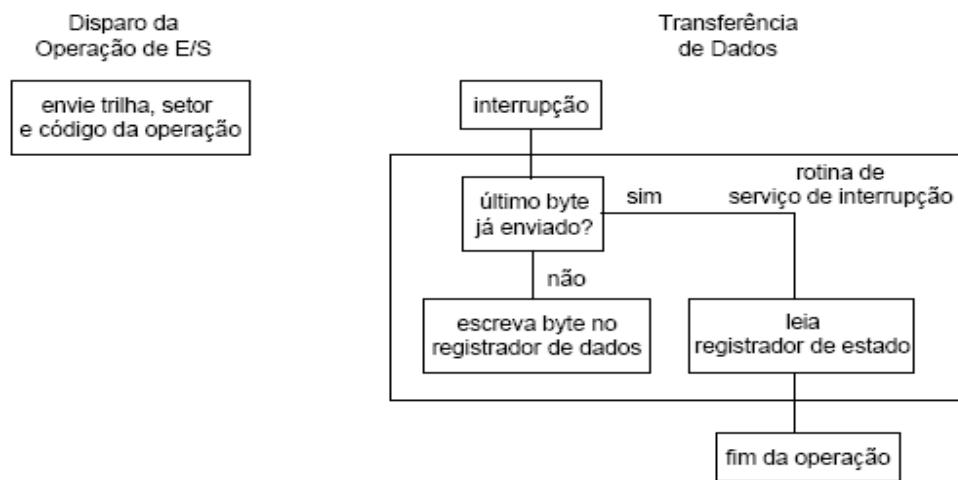


Figura 29 - Exemplo de E/S com interrupção.

UNIDADE 22

Técnicas de Transferência de Dados II

Objetivo: Conhecer as técnicas E/S com Interrupção e influências no funcionamento do computador.

Considerando o exemplo da operação de escrita de um setor de disco. O diagrama na Figura 29 mostra como esta operação é realizada através de E/S com interrupção. A operação é dividida em duas fases. Na fase de disparo da operação, o processador envia para a *interface* o comando, o número da trilha e do setor. Ao final da fase de disparo, o processador passa a executar outra atividade qualquer, por exemplo, parte de outro programa.

A *interface* inicia a fase de transferência de dados fazendo um pedido de interrupção ao processador, através do sinal de interrupção. Ao receber o pedido de interrupção, o processador suspende a execução do programa corrente e passa a executar uma rotina especial, chamada rotina de serviço de interrupção (também chamada *device driver* ou *device handler*). Nesta rotina, o processador verifica inicialmente se o último dado já foi enviado. Se este é o caso, o processador conclui a escrita do setor do disco lendo o registrador de estado da *interface*. Caso contrário, o processador envia um novo dado e retorna para o programa que se encontrava em execução.

Durante a fase de transferência de dados, a *interface* faz um pedido de interrupção a cada dado escrito no setor do disco. O processador responde ao pedido de interrupção executando a rotina de serviço e enviando um novo dado. Isto se repete até que todos os dados tenham sido escritos no setor do disco. Normalmente, a *interface* de disco conhece o tamanho do setor e mantém uma contagem dos dados já recebidos, de forma que ela pode determinar quando deve encerrar a sequência de pedidos de interrupção.

Em um sistema é comum existirem várias *interfaces* diferentes que fazem pedidos de interrupção ao processador. Cada *interface* deve ser atendida por uma rotina de serviço de interrupção específica para aquela *interface*. Assim, ao receber um pedido de interrupção, o processador deve determinar qual a rotina de serviço a ser executada. Além disso, quando duas ou mais *interfaces* fazem pedidos de interrupção simultâneos, é necessário decidir qual o pedido de interrupção que será atendido. Estas duas funções são suportadas por um componente do subsistema de E/S, chamado controlador de interrupção (*interrupt controller*).

Como mostra a figura abaixo, o sinal de interrupção de cada *interface* é ligado ao controlador de interrupção. O controlador de interrupção atribui um número e uma prioridade a cada um destes sinais. Quando um pedido de interrupção acontece, o controlador de interrupção envia para o processador, através do barramento de dados, o número do pedido. No caso de dois ou mais pedidos simultâneos, o controlador decide qual é o pedido com maior prioridade e envia para o processador o número correspondente.

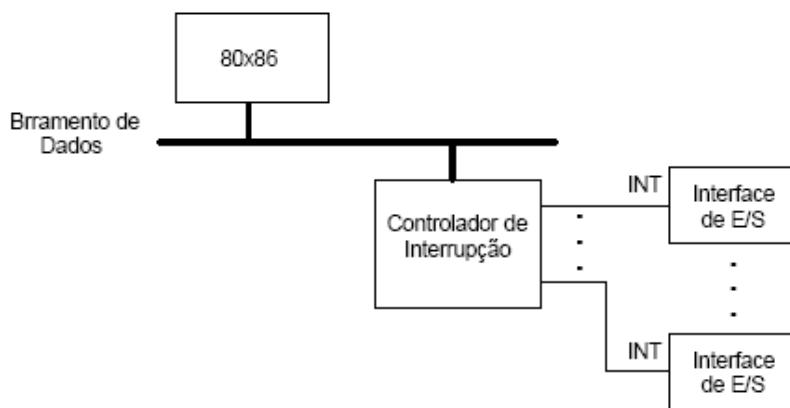


Figura 30 - O controlador de interrupção.

O processador usa o número recebido do controlador para indexar uma tabela armazenada na memória, chamada tabela de vetores de interrupção (*interrupt vector table*). Cada entrada desta tabela contém o ponteiro, ou vetor, para uma rotina de serviço. Ao receber um número de interrupção n , o processador lê o vetor contido na posição n da tabela e passa a executar a rotina de serviço de interrupção apontada por este vetor.

UNIDADE 23

Técnicas de Transferência de Dados III

Objetivo: Saber utilizar as técnicas E/S com acesso direto à memória.

Na E/S com interrupção, o processador não fica dedicado à operação de E/S. O processador é alocado somente quando realmente deve ser transferido um dado entre a memória e a *interface*, resultando em uma utilização mais eficiente do processador. No entanto, esta técnica apresenta uma desvantagem quanto à velocidade de transferência dos dados. Note que a transferência de um dado envolve a arbitragem pelo controlador de interrupção, a comunicação entre o controlador e o processador, o acesso à memória para a leitura do vetor de interrupção e finalmente o desvio para a rotina de serviço. Todas estas etapas acrescentam um retardo antes que o dado seja realmente transferido. Este retardo é chamado de tempo de latência de interrupção (*interrupt latency time*).

Em alguns tipos de periféricos, a taxa de transferência de dados entre o periférico e a *interface* é muito alta, ou em outras palavras, o intervalo de tempo entre a transferência de dois dados consecutivos entre o periférico e a *interface* é muito pequeno. Devido ao tempo de latência, o intervalo de tempo entre acessos do processador à *interface* pode tornar-se maior que o intervalo de tempo com que os dados chegam à *interface*. Se isto acontece, um novo dado chega à *interface* antes que o processador leia o dado anterior, e assim o dado anterior é perdido.

Na realidade, o que contribui para aumentar o tempo de latência é o fato de que o processador ainda é o responsável por controlar a transferência de dados. Para atender periféricos com alta taxa de transferência, usa-se a técnica de E/S com acesso direto à memória, onde o processador não participa da fase de transferência de dados. Esta técnica é analisada a seguir.

E/S com Acesso Direto à Memória

Na E/S com DMA (*Direct Memory Access*) um componente do subsistema de E/S, chamado controlador de DMA, é responsável por transferir os dados entre a memória e a *interface* de E/S. A Figura 5.6 mostra como o controlador de DMA é ligado ao resto do sistema.

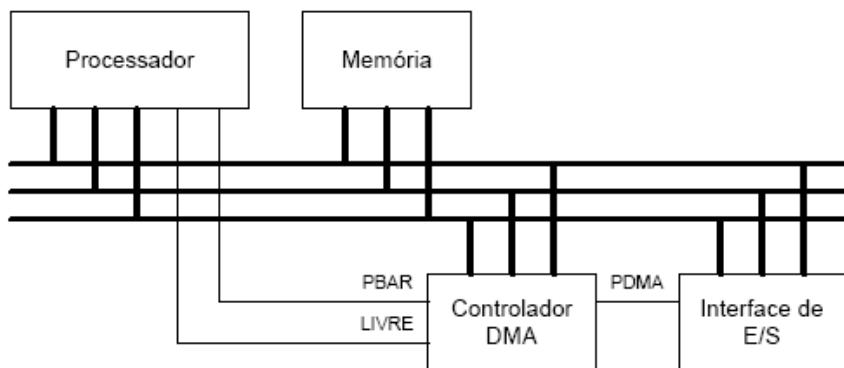


Figura 31 - Sistema com controlador de DMA.

Considerando novamente o exemplo da operação de escrita de um setor de disco. Na fase de disparo da operação, o processador informa ao controlador de DMA o número de dados a ser transferido; o endereço do primeiro dado e o sentido da transferência (no caso do exemplo, o sentido de transferência é da memória para a *interface* de E/S). Em seguida, o processador envia para a *interface* controladora de disco o número de trilha, o número de setor e o comando da operação.

O processador participa apenas da fase de disparo. Na fase de transferência de dados, o controlador de DMA assume o controle dos barramentos para realizar a transferência entre a memória e a *interface*. Para tanto, o controlador de DMA coloca o processador em um estado, chamado *hold state*, no qual o processador fica impedido de iniciar ciclos de barramento. Mais detalhadamente, a fase de transferência de dados envolve os seguintes passos:

Após receber o comando do processador, a *interface* de disco faz um pedido de DMA ao controlador de DMA através do sinal PDMA. Por sua vez, o controlador faz um pedido de

barramento ao processador, através do sinal PBAR. Ao liberar os barramentos, o processador responde ativando o sinal LIVRE, indicando ao controlador de DMA que este já pode usar os barramentos.

O controlador de DMA coloca no barramento de dados o endereço do primeiro dado e ativa o sinal de leitura de memória. A memória responde colocando o dado endereçado no barramento de dados. O controlador de DMA ativa o sinal de escrita em *interface* de E/S, fazendo com que a *interface* de disco capture o dado presente no barramento de dados.

Ao escrever o dado no setor do disco, a *interface* faz um novo pedido de DMA. O controlador de DMA inicia uma nova transferência, colocando o endereço do próximo dado no barramento de endereço e ativando os sinais de controle apropriados. Este passo se repete até que todos os dados tenham sido transferidos. Ao concluir a última transferência, o controlador de DMA retira o pedido de barramento, permitindo que o processador volte à operação normal.

Note que na E/S com DMA a transferência de cada dado envolve apenas uma leitura de memória e uma escrita de *interface* de E/S, realizadas pelo próprio controlador de DMA. A E/S com DMA efetivamente elimina o tempo de latência associado a cada dado transferido, que existe na E/S com interrupção. Isto permite que a E/S com DMA atinja taxas de transferência bem maiores que as técnicas de E/S que envolvem o controle do processador.

Em geral, é possível ter várias *interfaces* de E/S operando com a técnica de acesso direto à memória. Para tanto, o controlador de DMA possui várias entradas para pedido de DMA. O controlador de DMA associa a cada uma destas entradas um conjunto independente de registradores para armazenar o número de dados a serem transferidos, o endereço inicial e o sentido da transferência. Um grupo de sinais de controle com seus respectivos registradores formam o chamado canal de DMA. O controlador de DMA se encarrega de arbitrar entre *interfaces* que fazem pedidos de DMA simultâneos, usando um esquema de prioridades atribuídas aos canais de DMA.

UNIDADE 24

Padrões de Barramento I

Objetivo: Identificar os tipos de barramentos e suas diferenças.

Com o desenvolvimento de CPU's mais rápidas, com a maior demanda de software e maiores requisitos de vídeo, necessita-se de barramentos que atendam essas exigências. Por outro lado, os barramentos devem ser padronizados e, ainda assim, ter um custo compatível com o bolso dos usuários.

Os principais barramentos

ISA – Industry Standard Architecture: Criado em 1984 para os micros IBM PC/AT. É capaz de executar transferências de dados de 8 ou 16 bits operando a 8 MHz. Apesar de estar ultrapassado, este padrão ainda é suficiente para a conexão de placas de áudio, modems e outros dispositivos que não demandam grandes pré-requisitos de desempenho.



Figura 32 - Slot ISA.

EISA – Extended Industry Standard Architecture: Padrão de barramento que amplia o barramento tradicional (ISA) de 16 bits para 32 bits. Além disso, este padrão permite que mais de um processador compartilhe o barramento. Projetado como resposta ao MCA, o EISA aceita as placas de expansão ISA.

MCA – MicroChannel Architecture: Barramento proprietário de 32 bits lançado pela IBM em 1987 para os computadores da linha PS/2. Projetado visando multiprocessamento, este barramento permite que as placas de expansão se identifiquem para o sistema, evitando, desta forma, os conflitos que surgem nas configurações manuais necessárias nos barramentos convencionais. Diferente do EISA, o MCA não é compatível com as placas de expansão ISA.

VESA Local Bus: O padrão VESA foi desenvolvido por um grupo de fabricantes denominado *Video Electronic Standards Association*. Surgiu como uma opção para acomodar periféricos (principalmente as placas controladoras de vídeo e de disco) capazes de executar transferências de dados de 32 bits. Permaneceu durante alguns anos como uma alternativa boa e barata para viabilizar a melhoria da performance do sistema computacional como um todo. Um slot VLB (VESA Local Bus) é um slot ISA 16 bits com um terceiro e quarto slots no final. Desta maneira, uma placa ISA pode ser conectada no VLB.

UNIDADE 25

Padrões de Barramento II

Objetivo: Identificar os tipos de barramentos e suas diferenças.

PCI - Peripheral Component Interconnect: Este padrão permite transferências de dados de 32 ou 64 bits. Foi desenvolvido com o objetivo de maximizar a performance das placas-mãe equipadas com os microprocessadores Pentium. Trabalha com *clock* de 25 a 33 Mhz e permite taxas de transferências de até 132 MB/s. A vantagem chave do PCI sobre seu predecessor, o VLB, é a existência de um circuito no chipset que controla o barramento. Enquanto o VLB era basicamente uma extensão do barramento do processador 486. O PCI e seu chipset fornecem funcionalidades para controle, que habilitam o PCI a fazer mais coisas que o VLB poderia fazer.



Figura 33 - Slot PCI.

O barramento PCI opera concorrentemente com o barramento do processador. A CPU pode processar os dados como um cache externo enquanto o barramento PCI está ocupado transferindo informação entre outras partes do sistema. Além de ser mais eficiente, o barramento PCI incorpora o recurso Plug and Play, não necessitando que o usuário configure as placas adaptadoras.

As taxas de transferência chegam a 132 MB/s para 32 bits e 264 MB/s para 64 bits, para um clock de 33 Mhertz.

AGP: Accelerated Graphic Port: Criado pela Intel para acelerar o trabalho das placas de vídeo, fazendo com que elas se comuniquem direto com a memória RAM através da Ponte barramento local – barramento PCI. É utilizado exclusivamente para placas de vídeo 3D. Pode trabalhar com três taxas de transferência: x 1, 266 MB/s, x 2, com taxa de transferência de 533 MB/s e x 4, com taxa de transferência de 1 GB/s. Tais taxas dependem da placa de vídeo e da frequência FSB da placa-mãe.



Figura 34 - Slot AGP.

USB: Universal Serial Bus: A interface externa USB fornece uma comunicação serial de 12 Mbps, apenas sobre uma conexão de 4 fios. Um único porto USB pode ser usado para conectar até 127 periféricos, tal como mouse, modems, teclados, scanners, câmeras. E A

USB também atende às especificações Plug and Play da Intel, inclusive de poder conectar os dispositivos com a máquina ligada e sem precisar reiniciá-las. Simplesmente se conecta o dispositivo e a USB irá detectar automaticamente e alocar os recursos necessários para o seu funcionamento. Entre os sinais transportados pelo cabo USB, existe uma corrente de alimentação auxiliar de 5 V que permite energizar pequenos dispositivos USB.



Figura 35 - USB.

Firewire (IEEE 1394): O firewire é um barramento serial padrão externamente rápido que suporta taxas de transferências de dados de até 400 MBps. Um único porto 1394 pode ser usado para conectar até 63 dispositivos externos. Além da alta velocidade, o firewire também suporta dados isócronos, transmitindo dados com uma taxa garantida. Isto é ideal para dispositivos que necessitam altas taxas de transferências em tempo real, tal como dispositivos de vídeo. Embora muito rápido, o firewire é muito caro. Tal como o USB, o firewire suporta Plug and Play, e também fornece potências aos periféricos.



Figura 36 - Firewire.

UNIDADE 26

Execução de Programas I

Objetivo: conhecer as diversas formas do computador executar um programa.



MEDIATECA

VÍDEO

Atenção! Antes de dar continuidade aos seus estudos. Vá ao ambiente CAMPUS ON-LINE e assista ao vídeo referente à UNIDADE 26.



Programa em Linguagem de Máquina

Para executar qualquer tarefa, um computador precisa receber instruções precisas sobre o que fazer. Uma sequência adequada de instruções de computador, para a realização de uma determinada tarefa, se constitui num 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. As instruções eram escritas diretamente na linguagem do computador (formada apenas com 1 e 0).

Um programa em linguagem de máquina é uma longa série de 0 e 1, 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 - 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.

Então, um programa extenso escrito apenas usando 1 e 0; imagine que para cada diferente marca ou modelo de computador as regras para entender esses códigos serão totalmente diferentes e, finalmente. Se as instruções tivessem que ser escritas uma a uma; os dados adequadamente codificados e ordenados, perfurar todos os programas 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 tipo "erro no cartão X" (e mais nada!). Um programa escrito nessa linguagem era difícil de ser escrito sem que se cometesse muitos erros, processo esse longo, difícil, entediante e principalmente caro.

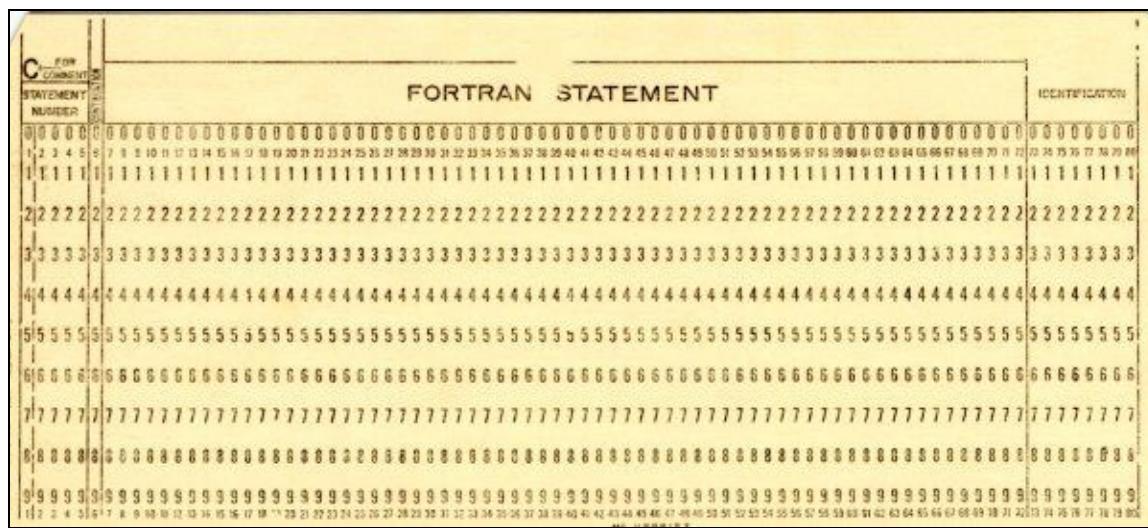


Figura 37 – Cartão em branco.

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

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

Linguagem de Montagem

A primeira tentativa bem-sucedida para resolver o problema acima descrito foi à criação de uma linguagem em que os códigos numéricos foram substituídos por mnemônicos (palavras ou símbolos), como por exemplo, LOAD = carregar e ADD = somar.

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, bastando 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 SALÁRIO para o local onde seriam armazenados os salários, etc., o que sem dúvida facilita enormemente o trabalho.

É importante lembrar que um computador entende única e exclusivamente a sua própria linguagem de máquina. Portanto, para escrever um programa em outra linguagem (mnemônicos), e ela ser entendido e processado no computador (linguagem de máquina); é preciso haver algum outro programa que leia o programa escrito nessa linguagem alternativa e o traduza para a linguagem nativa do computador.

O processo de tradução da linguagem de montagem para a linguagem de máquina 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 em 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 foi possibilitar o desenvolvimento de programas de crítica de sintaxe (os debuggers), facilitando o processo de depuração de erros de programação. No entanto, o processo continuava lento e complexo, exigindo do programador uma grande compreensão do processo e profundo conhecimento da máquina que ele estava programando.

Um programa de computador ainda era difícil de ser escrito, caro, e dependente do computador para o qual foi escrito, já que um programa escrito em linguagem de máquina para um determinado computador só poderá ser processado em computadores 100% compatíveis com ele.

UNIDADE 27

Execução de Programas II

Objetivo: Conhecer as linguagens de Programação e Tradutores.

Linguagens de Programação

Esses problemas de complexidade no desenvolvimento de softwares e compatibilidade com hardware levaram a uma busca por linguagens que fossem mais simples de programar e entender; mais rápidas e eficientes, levando a programas mais enxutos, com menos instruções, menos dependente 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. Essas linguagens foram estruturadas buscando refletir melhor os processos humanos de solução de problemas. Essas linguagens orientadas a problema são também chamadas linguagens de alto nível, por serem afastadas do nível de máquina.

As primeiras linguagens foram FORTRAN (1957), usada basicamente para manipulação de fórmulas; ALGOL (1958), para manipulação de algoritmos; COBOL (1959), para processamento comercial e ainda hoje bastante usada, especialmente em computadores de grande porte (mainframes) em bancos.

Nas décadas de 60 e 70, podemos citar Pascal, a primeira linguagem de alto nível estruturada; 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 80, surgiu o C e depois o C++ (com suporte a objetos), que estão entre as linguagens mais utilizadas hoje.

Cada nova linguagem criada visa atingir níveis de abstração mais altos, pois afastam cada vez mais o programador do nível de máquina. Se por um lado essas novas linguagens facilitam muito o trabalho dos programadores, pois reduzem a necessidade de conhecer o hardware da máquina; elas cobram um alto preço em termos de desempenho, pois são cada vez mais lentas, ao consumir cada vez mais ciclos de máquina e espaço em memória.

Esse aumento de exigência ao poder de processamento dos computadores é compensado pelo aumento acelerado do poder de processamento dos novos *chips* (exemplificado pela chamada Lei de *Moore*, que afirma que o poder de processamento dos *chips* dobra a cada 18 meses) e pelos avanços na arquitetura dos computadores.



Dica

Caso queira saber mais sobre a Lei de Moore não deixe de acessar o link:

http://pt.wikipedia.org/wiki/Lei_de_Moore



Tal como na linguagem humana, as linguagens de computadores proliferaram e sempre há problemas que ainda persistem, continuando a busca por uma linguagem ideal - a solução "definitiva". As duas tecnologias mais utilizadas atualmente são Java e o .NET da Microsoft.



Dica

Caso queira saber mais sobre .NET, acesse:

<http://msdn.microsoft.com/pt-br/default.aspx> ou

<http://pt.wikipedia.org/wiki/.NET>

Caso queira saber mais sobre JAVA, acesse:

[http://pt.wikipedia.org/wiki/Java_\(linguagem_de_programa%C3%A7%C3%A3o\)](http://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o))



Tradução

Programas em linguagem de alto nível, a exemplo dos programas escritos em linguagem de Montagem, precisam ser traduzidos para linguagem de máquina para poderem ser entendidos e processados pelo computador.

O processo de tradução do programa pode ser classificado como “Montagem”, “Compilação” e “Interpretação”, conforme detalhado a seguir.

Montagem

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.

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 definidos 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 dados.
- d) Converter valores de constantes em binário.

UNIDADE 28

Execução de Programas III

Objetivo: Conhecer o significado de Compilação, Ligador e Interpretação.

Compilação

Compilação é o processo de tradução de um programa escrito em linguagem de alto nível para código em linguagem de máquina. Compilação é um processo 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 na compilação a relação é múltipla, cada instrução do código fonte gerando várias instruções de máquina.

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 e atribui-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.

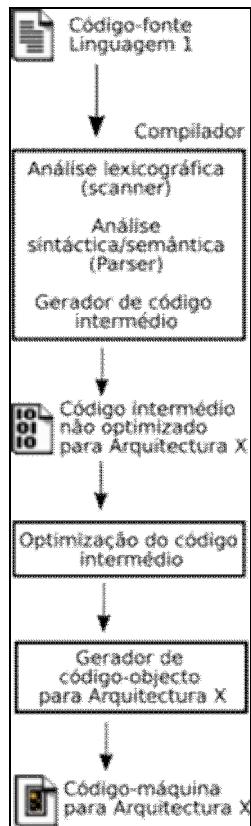


Figura 38 – Processo de Compilação.

Bibliotecas

O desenvolvimento de um programa certamente utilizará diversas operações que são comuns a muitos outros programas. Por exemplo, a execução de uma instrução de entrada e saída, a classificação dos dados de um arquivo, o cálculo de funções matemáticas, etc.

Uma linguagem de alto nível geralmente incorpora diversas rotinas prontas (que fazem parte da linguagem) e que compõem bibliotecas (libraries) de funções pré-programadas que poderão ser utilizadas pelo programador, poupando tempo, aumentando a eficiência e evitando erros. Dessa forma, um programa em alto nível possivelmente conterá diversas chamadas de biblioteca.

Ligaçāo (*Linkedição*)

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 devam 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 (Link Editor). O resultado da execução do Ligador é o Código Final pronto para ser executado pelo computador, chamado módulo de carga ou código executável.

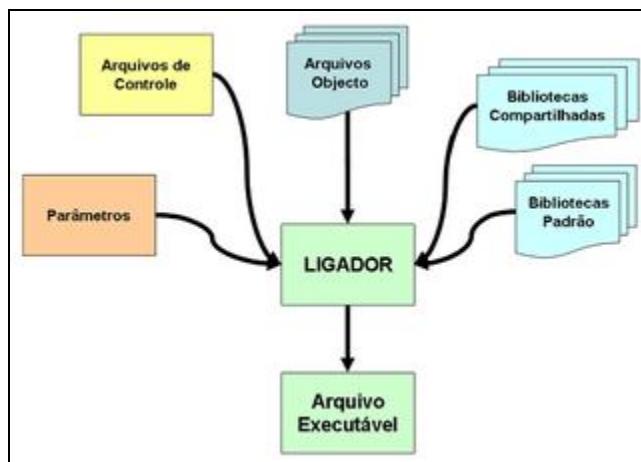


Figura 39 – Lingador ou linkedição.

O módulo de carga após ser testado e depurado é armazenado em memória de massa para ser executado quando necessário. 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.

Interpretação

Com o processo de execução de um programa em fases distintas (compilação / ligação / execução) apresentado, um programa para ser executado precisa primeiro ter sido convertido para código objeto pelo compilador e depois ter passado pelo ligador. Esse processo é o mais largamente utilizado, porém não é o único.

O método alternativo chama-se de 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. Cada comando é lido, verificado, convertido em código executável e imediatamente executado, antes que o comando seguinte seja sequer lido.

As linguagens voltadas para Web (como ASP, PHP, ColdFusion, etc) são interpretadas. A imagem a seguir exemplifica o processo de interpretação.

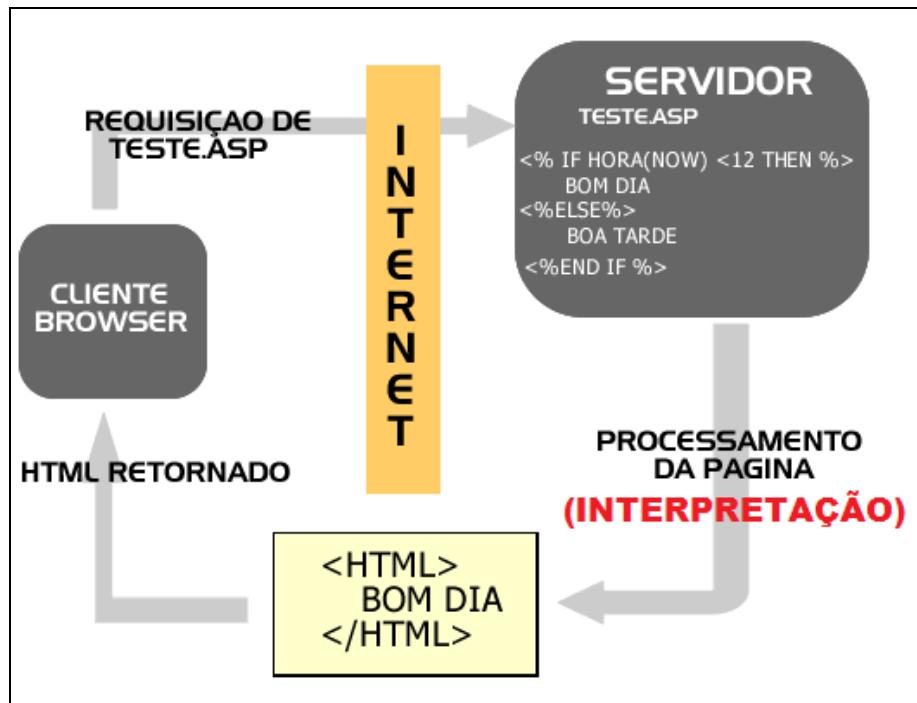


Figura 40 – Processo Interpretação.

Linguagens como C, Pascal, COBOL, etc., são linguagens tipicamente compiladas, enquanto o BASIC foi desenvolvido como linguagem interpretada (hoje também existem linguagens BASIC compiladas e o programador pode optar).

UNIDADE 29

Execução de Programas IV

Objetivo: Identificar com clareza Comparação entre Compilação e Interpretação.

Compilação e Interpretação - comparação

Sempre que houver duas opções, haverá vantagens e desvantagens para cada uma delas, pois se assim não fosse, a que apresentasse sempre desvantagem seria abandonada. A seguir uma comparação entre os dois métodos:

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.

Consumo de memória

No método de interpretação, o interpretador é um programa geralmente grande e que precisa permanecer na memória durante todo o tempo que durar a execução do programa, pois 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.

Desta forma, o método de interpretação acarreta um consumo de memória muito mais elevado durante a execução do programa.

Repetição de interpretação

No método de compilação, um programa é compilado e ligado apenas uma vez, e 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 ser executado.

Outro aspecto 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 se faz uma única vez, em tempo de compilação e ligação.

Estas características levam a um maior consumo de tempo no método de interpretação, que é, portanto mais lento.

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 sentido. 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 variável causadora do problema.

Plataforma Cliente

Um computador somente é capaz de executar programas compilados que tenham sido desenvolvidos para ele. Assim, um programa desenvolvido para rodar em PC's rodando Windows não funciona em PC's com UNIX ou em Macintosh. Imagine então uma página na Internet, com textos, imagens e programas que podem ser visualizados e processados por quase qualquer computador.

Páginas WEB utilizam linguagens padronizadas, tais como HTML - para a escrita das páginas - e linguagens interpretadas como ASP, PHP, ColdFusion, entre outras para codificação dos aplicativos. Assim, cada uma das plataformas através dos programas visualizadores de páginas Internet, conhecidos como browsers ou mesmo através de seus respectivos sistemas operacionais, pode interpretar corretamente qualquer página feita e hospedada em qualquer computador.

UNIDADE 30

Execução de Programas V

Objetivo: Identificar Máquinas Virtuais e Java Bytecode

Máquinas Virtuais

Levando o conceito de interpretação um pouco mais adiante, imagine desenvolver um programa conversor que pegasse 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 máquina. Um PC "virtual" emulado em um Macintosh, que estaria assim apto a rodar qualquer programa escrito para PC. Esse programa emulador criaria um ambiente chamado de Máquina Virtual, isto é, uma máquina que se comporta como outra máquina diferente, não compatível.

Desta forma, é possível ter no PC doméstico uma máquina virtual com Linux, Windows 2008 Server, ou qualquer outro sistema operacional que desejar.



Dica

Caso queira saber mais sobre máquinas virtuais acesse o link:

http://pt.wikipedia.org/wiki/M%C3%A1quina_virtual

O “Virtual PC” é um aplicativo gratuito e de fácil manuseio para quem quiser fazer os primeiros testes com máquinas virtuais. Você pode fazer o download através do link:

<http://www.microsoft.com/downloads/details.aspx?FamilyId=04D26402-3199-48A3-AFA2-2DC0B40A73B6&displaylang=en>



Bytecode Java

O código de um programa de computador escrito na linguagem Java é compilado para uma forma intermediária de código denominada bytecode, que é interpretada pelas Máquinas Virtuais Java (JVMs). É essa característica que faz com que os programas Java sejam independentes de plataforma, executando em qualquer sistema que possua uma JVM. Cada opcode tem o tamanho de um byte — daí o seu nome — e assim o número de diferentes códigos de operação está limitado a 256. Os 256 possíveis valores para códigos de operação não são todos utilizados. Na verdade, alguns dos códigos foram inclusive reservados para nunca serem implementados.

Um programador Java não precisa entender — e nem tomar conhecimento — dos bytecodes Java para ser proficiente na linguagem, da mesma forma que um programador de qualquer linguagem de alto nível, compilada para linguagem de máquina, não precisa conhecer a linguagem de montagem do computador hospedeiro, para escrever bons programas naquela linguagem.

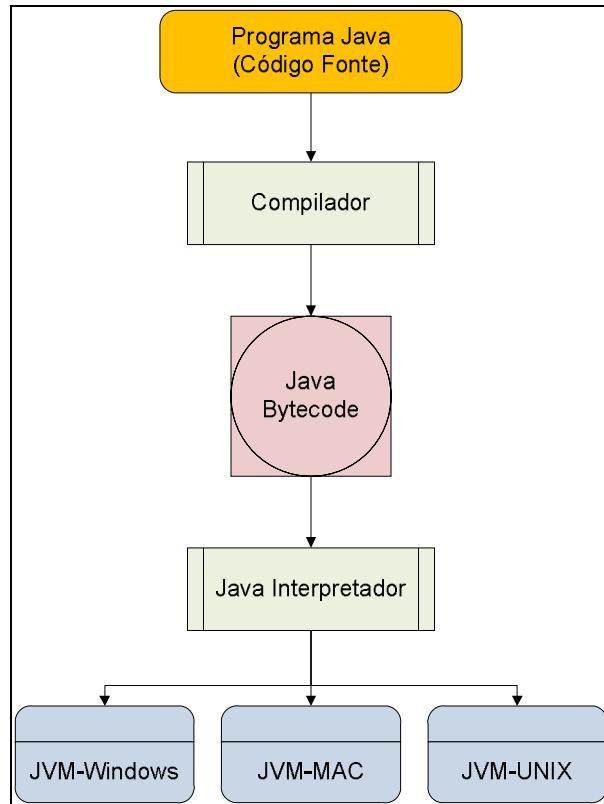


Figura 41 – Java Bytecode.



Dica

Caso queira saber mais sobre Java Bytecode acesse o link:

http://en.wikipedia.org/wiki/Java_bytecode





Atividades

Antes de iniciar sua Avaliação Online, é fundamental que você acesse sua SALA DE AULA e faça a Atividade 3 no “link” ATIVIDADES.



Atividades

Atividades dissertativas

Acesse sua sala de aula, no link “Atividade Dissertativa” e faça o exercício proposto.

Bons Estudos!



GLOSSÁRIO

Caso haja dúvidas sobre algum termo ou sigla utilizada, consulte o link Glossário em sua sala de aula, no site da ESAB.

BIBLIOGRAFIA

Caso haja dúvidas sobre algum termo ou sigla utilizada, consulte o link Bibliografia em sua sala de aula, no site da ESAB.